Universidade de Aveiro Departamento de Matemática
2006

Gladys Castillo Jordán

Algoritmos de Aprendizagem Adaptativos para Classificadores de Redes Bayesianas

Adaptive Learning Algorithms for Bayesian Network Classifiers

Universidade de Aveiro Departamento de Matemática
2006

Gladys Castillo Jordán

Algoritmos de Aprendizagem Adaptativos para Classificadores de Redes Bayesianas

Adaptive Learning Algorithms for Bayesian Network Classifiers

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Matemática, realizada sob a orientação científica do Doutor João Manuel Portela da Gama, Professor Auxiliar do Departamento de Economia da Universidade do Porto e da Doutora Ana Maria Reis d'Azevedo Breda, Professora Associada com Agregação do Departamento de Matemática da Universidade de Aveiro

To Leslie, Beatriz and Arturo

In memory of my dad.

o júri

presidente

Doutora Nilza Maria Vilhena Nunes da Costa
Professora Catedrática da Universidade de Aveiro

vogais

Doutor Pedro Larrañaga
Professor Catedrático da Universidade do País Basco, Espanha

Doutor Domingos Moreira Cardoso
Professor Catedrático da Universidade de Aveiro

Doutora Ana Maria Reis d'Azevedo Breda
Professora Associada com Agregação da Universidade de Aveiro (Co-Orientadora)

Doutor André Carlos Ponce de Leon Ferreira de Carvalho
Professor Associado da Universidade de São Paulo, Brasil

Doutor Carlos Manuel Robalo Lisboa Bento
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Doutor João Manuel Portela da Gama
Professor Auxiliar da Faculdade de Economia da Universidade do Porto (Orientador)

acknowledgments

"When you want something, all the universe conspires in helping you to achieve it"
Paulo Coelho

I want to express my sincerest gratitude to my supervisor Prof. João Gama who introduced me in such an interesting research area like Machine Learning and Bayesian Networks. Thank you for your guidance and constructive advices on various levels given during the entire period I have prepared the content of this thesis. I am also very grateful to my co-supervisor Prof. Ana M. Breda for her invaluable friendship, support and encouragement in achieving this long-time dream of doing my PhD.

During the last nine years my work at the University of Aveiro in the Department of Mathematics were marked by a wonderful wealth of friends and colleagues. It is difficult for me to enumerate the many ways in which the people of the Department of Mathematics have supported me through the various stages of my career all these years. I wish, in general, to thank all my friends and colleagues for believing in me and giving me the opportunity to work with them and to achieve my goals and dreams. Without such a nice working atmosphere this thesis would not have been possible. Many thanks are due to the actual head of the Department of Mathematics for the computer support, which helped me to run all the experiments.

Thanks to LIACC, the FCT and the projects Adaptive Learning Systems I and II for providing me with financial support to attend very interesting conferences where I have the opportunity to receive helpful feedback on my work. I would also like to thank the researchers of the Intelligent System Group of the Department of Computer Science and Artificial Intelligence at the University of the Basque Country for their interest in my work, reading papers, and providing invaluable feedback and advice.

I am especially very grateful to Dorabella Santos for her unconditional help in the revision of the English spelling and grammar.

I want also to express my foremost gratitude to all my family and friends who, although very distant, have always supported and encouraged me over these hard years. I am especially very grateful to my aunt Beatriz, who has encouraged me at any time, no matter how far away from me she was. Thanks also to my younger cousin Stephanie, who was always ready to help me with the English revision.

Finally, all these years of many ups and downs associated with this thesis would not have been bearable without the love and never-ending encouragement and support of my husband Leslie and my children Arturo and Beatriz. Thank you for being so patient with me and for providing me with a lovely atmosphere at home where I could work hard to write this thesis.

palavras-chave

classificadores de redes Bayesianas, Naive Bayes, sistemas de aprendizagem on-line adaptativos, modelo prequencial de avaliação estatística, mudança de conceito, aprendizagem de redes Bayesianas, criterios para selección de modelos, algoritmo de aprendizagem de ascenso a colina (hill-climber)

resumo

Nesta tese consideramos o desenvolvimento de algoritmos adaptativos para classificadores de redes Bayesianas (BNCs) num cenário *on-line*. Neste cenário os dados são apresentados sequencialmente. O modelo de decisão primeiro faz uma predição e logo este é actualizado com os novos dados. Um cenário on-line de aprendizagem corresponde ao cenário "prequencial" proposto por Dawid. Um algoritmo de aprendizagem num cenário prequencial é eficiente se este melhorar o seu desempenho dedutivo e, ao mesmo tempo, reduzir o custo da adaptação. Por outro lado, em muitas aplicações pode ser difícil melhorar o desempenho e adaptar-se a fluxos de dados que apresentam mudança de conceito. Neste caso, os algoritmos de aprendizagem devem ser dotados com estratégias de controlo e adaptação que garantem o ajuste rápido a estas mudanças.

Todos os algoritmos adaptativos foram integrados num modelo conceptual de aprendizagem adaptativo e prequencial para classificação supervisada designado AdPreqFr4SL, o qual tem como objectivo primordial atingir um *equilíbrio* óptimo entre *custo-qualidade* e controlar a *mudança de conceito*. O equilíbrio entre custo-qualidade é abordado através do controlo do viés *(*bias*)* e da adaptação do modelo. Em vez de escolher uma única classe de BNCs durante todo o processo, propomo-nos utilizar a classe de classificadores Bayesianos k-dependentes (k-DBCs) e começar com o seu modelo mais simples: o classificador Naïve Bayes (NB) (quando o número máximo de dependências permissíveis entre os atributos, k, é 0). Podemos melhorar o desempenho do NB se reduzirmos o bias produto das restrições de independência. Com este fim, propomo-nos incrementar k gradualmente de forma a que em cada etapa de aprendizagem sejam seleccionados modelos de k-DBCs com uma complexidade crescente que melhor se vai ajustando ao actual montante de dados. Assim podemos evitar os problemas causados por demasiado viés (*underfitting*) ou demasiada variância (*overfiting*). Por outro lado, a adaptação da estrutura de um BNC com novos dados implica um custo computacional elevado. Propomo-nos reduzir nos custos da adaptação se, sempre que possível, usarmos os novos dados para adaptar os parâmetros. A estrutura é adaptada só em momentos esporádicos, quando é detectado que a sua adaptação é vital para atingir uma melhoria no desempenho. Para controlar a mudança de conceito, incluímos um método baseado no Controlo de Qualidade Estatístico que tem mostrado ser efectivo na detecção destas mudanças.

Avaliamos os algoritmos adaptativos usando a classe de classificadores k-DBC em diferentes problemas artificiais e reais e mostramos as vantagens da sua implementação quando comparado com as versões no adaptativas.

keywords

Bayesian network classifiers, Naïve Bayes, adaptive on-line learning systems, prequential framework, concept drift, learning bayesian networks, model selection criteria, hill-climbing learning algorithm

abstract

This thesis mainly addresses the development of adaptive learning algorithms for Bayesian network classifiers (BNCs) in an on-line leaning scenario. In this scenario data arrives at the learning system sequentially. The actual predictive model must first make a prediction and then update the current model with new data. This scenario corresponds to the Dawid's prequential approach 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, a problem known as concept drift. In changing environments, learning algorithms should be provided with some control and adaptive mechanisms that effort to adjust quickly to these changes.

We have integrated all the adaptive algorithms into an adaptive prequential framework for supervised learning called **AdPreqFr4SL**, which attempts to handle the **cost-performance trade-off** and also to **cope with concept drift**. The cost-quality trade-off is approached through **bias management** and **adaptation control**. The rationale is as follows. Instead of selecting a particular class of BNCs and using it during all the learning process, we use the class of k-Dependence Bayesian classifiers and start with the simple Naïve Bayes (by setting the maximum number of allowable attribute dependence k to 0). We can then improve the performance of Naïve Bayes over time if we trade-off the bias reduction which leads to the addition of new attribute dependencies with the variance reduction by accurately estimating the parameters. However, as the learning process advances we should place more focus on bias management. We reduce the bias resulting from the independence assumption by gradually adding dependencies between the attributes over time. To this end, we gradually increase k so that at each learning step we can use a class-model of k-DBCs that better suits the available data. Thus, we can avoid the problems caused by either too much bias (*underfitting*) or too much variance (*overfitting*). On the other hand, updating the structure of BNCs with new data is a very costly task. Hence some adaptation control is desirable to decide whether it is inevitable to adapt the structure. We reduce the cost of updating by using new data to primarily adapt the parameters. Only when it is detected that the use of the current structure no longer guarantees the desirable improvement in the performance, do we adapt the structure. To **handle concept drift**, our framework includes a method based on Statistical Quality Control, which has been demonstrated to be efficient for recognizing concept changes.

We experimentally evaluated the AdPreqFr4SL on artificial domains and benchmark problems and show its advantages in comparison against its non-adaptive versions.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

The need for understanding large, complex collected data is becoming increasingly important to many fields of business, science and engineering in today's competitive world. Bayesian networks (also known as *belief networks*, *probabilistic networks*) have become one of the most popular probabilistic models used in Artificial Intelligence to model data. Bayesian networks were introduced by Pearl in [108] and provide a sound theoretical framework to represent and manipulate probabilistic dependencies between random variables. A Bayesian network is composed of two components: the *qualitative* part (its structure) and the *quantitative* part (the set of parameters that quantifies the network). The *structure* is a *directed acyclic graph* (DAG) in which *nodes* represent random variables, and the *arcs* represent dependencies between these variables. The *parameters* are conditional probabilities that represent the strength of the dependencies.

In the last years there has been an increasing interest in inducing Bayesian networks which are efficient for classification. *Classification* means the task of assigning one of predefined classes to objects described by a set of attribute values. Data classification can be applied to a wide variety of domains: *medical diagnosing*, *loan applications*, *user modeling*, *pattern recognition*, *biomedical informatics*, *fault diagnosing*, etc. Bayesian networks when used as classifiers are known as *Bayesian network classifiers*, BNCs from now on. BNCs are becoming increasingly popular as a classification tool due to recent developments in learning Bayesian networks. As pointed out in [6] learning Bayesian networks from data offers a lot of advantages when compared with other models. First, Bayesian networks are an effective representation for decision making and reasoning. This allows the learned model to be easily integrated as a component of a complex system. Second, they have a compact and comprehensible representation, which allows human experts to provide prior knowledge in the form of strong

constrains on the initial structure of the network. Third, the output of the learning process is also very comprehensible to humans.

The task of learning a Bayesian network from data is two-fold: learning the *network structure* and learning *the set of parameters*. Learning 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. The notion of "*best fits*" is defined via a *model selection criterion*. Model selection can be approached as a discrete optimization problem where the *model selection criterion* that measures the quality of each candidate model is optimized in the space of feasible hypotheses. The choice of an appropriate model selection criterion according to the learning goals is crucial for model selection tasks. In the case when a Bayesian network is induced for classification, the main goal is to build an accurate classifier. Hence, the model selection task is to choose the model which yields the most accurate classifications with respect to a given loss function (as a rule, the *zero-one loss* is used) [78].

In recent years the issue of the selection of an appropriate model selection criterion for learning BNCs has received a lot of attention [32, 38, 43, 52, 78]. It has been suggested that search strategies for learning BNCs should select among models using selection criteria specialized for classification; otherwise it can result in suboptimal choices during the search process. In this thesis, instead, we are more interested in exploring other aspects of model selection criteria that can affect the performance of BNCs. All model selection criteria that are used in practice either implicitly or explicitly choose a trade-off between *goodness of fit* and *complexity* of the models involved [53]. Indeed, model selection makes a *bias-variance* trade-off in order to select a model with the appropriate complexity for the amount of data available [25, 57]. Van Allen et al. in [4] have carried out an empirical comparison of several model selection criteria in order to identify how each one handles the *bias-variance* trade-off in learning Bayesian networks. Whereas their work explored the behaviour of different criteria for small samples and in the general framework of Bayesian networks, we have investigated these issues for the particular case when Bayesian networks are induced for classification in a *prequential learning scenario*.

During the past several years there has been an explosive growth of methods for learning BNCs from data [8, 11, 23, 43, 44, 49, 52, 66, 86, 121, 141]. Nevertheless, most of them are

implemented in a *batch learning scenario*, where all training examples are given at the same time to the learning algorithms and the induced classifier is no more revised with future data. However, nowadays in many current real-world applications the data processing system is organized in the form of a *data stream* rather than a static data repository. Since data often needs to be processed in an *on-line manner*, on-line learning systems are becoming increasingly important in today's real-world applications. An *on-line*, predictive learning scenario corresponds to the situation where the data arrives at the learning system sequentially, not at the same time. The actual predictive model must first make a prediction and then the current model is updated with new data. This philosophy about on-line learning paradigms has been exposed by Dawid [35] in his *prequential* approach for statistical validation of models.

Efficient learning algorithms in a *prequential learning scenario* involve an artful trade-off between the *gain in the quality of the model* and the *cost of updating* the model in the light of new data. Since the quality of a BNC is determined by its *predictive capability*, an efficient learning algorithm for BNCs must be able, above all, to improve its predictive accuracy over time while optimizing the cost of updating. Bayesian networks suffer from several drawbacks for updating purposes. While sequential updating of the parameters is straightforward (if data is complete); updating the structure is a more costly task [61]. We can reduce the cost of updating if we try to use new data to primarily adapt the parameters and only if this is really necessary, do we adapt the structure. On the other hand, in many real-world situations it may be difficult to improve and adapt 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 that the learning system is trying to learn. In changing environments the predictive model should be adapted quickly to these changes in order to maintain its performance level. Learning systems that track concept drift are often called adaptive systems. Many adaptive systems employ regular model updates while new data arrives. However, a better approach is to provide the system with some controlling mechanisms aimed at selecting the best adaptive actions according to the current learning goal.

The main purpose of this work has been the development of *adaptive algorithms* for BNCs in a *prequential learning scenario*, which attempt to handle the *cost-quality* trade-off and cope with *concept drift*. We have integrated all the adaptive algorithms into an unified, *adaptive*

prequential framework for supervised learning called `AdPreqFr4SL`. During all the learning process the `AdPreqFr4SL` aims to satisfy the following goals:

1. **Improvement** in the predictive accuracy while reducing the cost of updating, thus performing an artful *cost-quality* trade-off.
2. **Recoverability** if the performance goes down due to concept drift or another causes, trying to improve it back to a level, at least, no worse than the previous best performance.
3. **Stability** when a desirable level of performance has been achieved.

The *Naïve Bayes* classifier [39, 83] is one of the most used classifiers in real-world on-line applications mainly due to its *simplicity*, *effectiveness*, *easy interpretability* and *incremental nature*. Naïve Bayes significantly simplifies learning by assuming that attributes are independent given the class. This can be viewed as a Bayesian network with a simple structure that has the class node as the parent node of all other attribute nodes. In spite of the fact that Naïve Bayes presents a *high bias*, it shows a good performance due to a *high variance management*, thus producing accurate classifications [13]. However, in practice, the independence assumption is violated which can lead to a poor predictive performance. We can improve Naïve Bayes if we trade-off the *bias reduction* which leads to the addition of new attribute dependencies, and, consequently, to the estimation of more parameters, with the *variance reduction* by accurately estimating the parameters [72]. Different classes of BNCs attempt to reduce the bias of the Naïve Bayes by relaxing the attribute independence assumption. For instance, a **T***ree* **A***ugmented* **N***aïve Bayes* (TAN) classifier [43, 44] extends the Naïve Bayes structure by allowing the attributes to form a *tree*. A **B***ayesian network* **A***ugmented* **N***aïve Bayes* (BAN) [22, 23, 43, 44] extends the Naïve Bayes structure by allowing the attributes to form an arbitrary DAG. We can also use as a classifier a **G***eneral* **B***ayesian* **N***etwork* (GBN) without any restriction about how the class node is treated [22, 23, 32, 43, 44, 52, 86, 93]. Nevertheless, not always do the more complex BNCs outperform the simple Naïve Bayes. More complex classifiers allow for a better representational power, but suffer from a decreased ability to generalize to unseen data. They can *overfit* the training data. In turn, simpler classifiers cannot capture the true structure in the data. They can *underfit* the data. Both, *overfitting* and *underfitting* can lead to a deterioration of the performance.

Finding the appropriate balance between *complexity* and *performance* is a matter of handling the *bias-variance trade-off*. Increasing the model's complexity decreases bias but increases the variance in the parameter values. 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. One of the main differences among the different class-models of BNCs is the way each of them restrict the number of parents for each attribute node, and hence, the search space. We can control the complexity of BNCs if we choose *an appropriate class-model* for the available training data. Thus, if we scale up the complexity of BNCs slowly enough, the use of more training data will reduce bias at a rate that also reduces variance and consequently the classification error. 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 avoiding the problems caused by either too much bias (*underfitting*) or too much variance (*overfitting*).

We chose the class of k-**D**ependence **B**ayesian **C**lassifiers (k-DBCs) introduced by Sahami in [121] for illustrating our approach. A k-DBC is a Bayesian network, which contains the structure of Naïve Bayes and allows each attribute to have a maximum of k attribute nodes as parents. For instance, Naïve Bayes is a 0-DBC, TAN is a 1-DBC, etc. k-DBCs are very suitable for our adaptive proposal. By varying k we can obtain classifiers that move smoothly along the spectrum of feature dependencies, thus providing a flexible control over the model's complexity.

The adaptive strategy in the `AdPreqFr4SL` for incorporating new data leads to a more artful trade-off between the *cost of updating* and the *gain in performance*, even in changing environments. This is based upon two main policies: *bias management* and *gradual adaptation*. 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 Naïve Bayes. 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 dependencies among the attributes. The rationale is as follows. 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 simplest Naïve Bayes (by setting $k = 0$). Whenever we obtain new data, we first try to perform the less costly FIRST LEVEL of adaptation, that is, we adapt only the parameters. Only when

we obtain some evidence indicating that the performance using the current structure stops improving in the desirable tempo, do we move to a more costly SECOND LEVEL of adaptation and start *adapting the structure* by searching for new dependencies between the attributes. If after searching, the resulting structure remains the same, then do we move to the THIRD LEVEL of adaptation. If it is still possible, k is increased by one, and the current structure is once again adapted by searching for new attributes dependencies but now in the augmented search space. In addition, 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.

The `AdPreqFr4SL` integrates some monitoring tools for *bias management* with a method for *handling concept drift* based on *Statistical Quality Control* first presented in [21]. In order to achieve a desirable performance even when dealing with concept drift, the `AdPreqFr4SL` is provided with simple controlling mechanisms based on the observation of some performance indicators. If during the monitoring process a concept drift is detected, some actions to adapt the learner to these changes are taken. Only in the case when an *abrupt concept change* is identified, the adaptation process is re-launched from its INITIAL LEVEL and a new Naïve Bayes classifier is built using the examples suspected to belong to a new target concept. Although the `AdPreqFr4SL` is presented in this thesis for the class of k-DBCs, we believe that its adaptive and control strategies can be easily adapted to other families of classifiers based on discrete search with a hierarchical and increasing control over the complexity of its induced hypotheses.

We conclude the introduction with a brief outline of the contents of this thesis. In the first part of this thesis (Chapter 1 and Chapter 2), we give an introduction to the more general problem of learning Bayesian networks that due to its nature is more mathematically precise than the remaining of the chapters. The following chapters deal with the more restricted class of Bayesian network classifiers that are the subject of this thesis.

Chapter 1 presents the basic concepts, mathematical formalisms and philosophies underlying the problem of learning probabilistic models from data in statistical inference. Two related problems are introduced: *parameter estimation* and *model selection*. The focus here is on the derivation of different model selection criteria. Because we are interested in compar-

ing the performance of different BNCs induced with different model selection criteria (scoring functions), a good understanding of the philosophies adopted under the different approaches to derive the model selection criteria for the general problem of learning probabilistic models from data is essential for a good understanding and objective evaluation of how these criteria can affect the performance of score-based approaches to learn BNCs from data.

Chapter 2 describes the learning problem for the particular framework of Bayesian networks along with the more relevant issues on the score-based approach. We overview the three factors that affect the performance of score-based approaches for learning Bayesian networks: *i*) *the parameter estimator*; *ii*) *the scoring function*; and *iii*) *the search method*; and give the derivation of the different estimators and scoring functions for Bayesian networks. Finally, we introduce heuristic search methods and the *hill-climbing* search procedure for learning the structure of Bayesian Networks.

Chapter 3 begins with an introduction to the general problem of the supervised learning. Next, it introduces the Naïve Bayes classifier and summarizes various reasons why we might expect the surprisingly good performance of Naïve Bayes in practice. Through an overview of previous work which attempted to improve the predictive accuracy of Naïve Bayes, different classes of BNCs are then introduced. The chapter concludes with a broad discussion concerning how the choice of the *scoring function* and the *class-model* can affect the performance of BNCs learned from data.

Chapter 4 provides the results of a conducted experimental study using the class of k-DBCs along with an in-depth analysis. The main goal is to compare how the choice of different scores and class-models (varying k) affect the performance of k-DBCs induced with the same underlying learning algorithm at different time points in a *prequential learning framework*. This comparative study was basically motivated to test whether it makes sense to gradually increase the k value in order to adjust the complexity of the k-DBC class-model, and hence, the complexity of the induced k-DBCs to the current amount of training data.

Chapter 5 is the core of our contribution. The chapter first introduces the main factors that are required for drawing up an efficient adaptive learning framework. Then it describes the adaptive and control strategies that we have adopted to handle the *cost-quality* trade-off and *concept drift* for learning BNCs in a *prequential* scenario. Finally, this chapter describes the `AdPreqFr4SL` as a whole learning framework that integrates all the developed adaptive

algorithms.

Chapter 6 describes the experiments we conducted that demonstrate the effectiveness of our adaptive approach. We evaluated the adaptive algorithms for BNCs, using both, *artificially generated* problems and *benchmark problems* from the UCI repository [102]. The use of artificial domains allowed us to test the two main issues that the algorithm exhibit: *bias management* and *concept drift management* knowing the true degree of attribute dependence and whether concept drift actually occurs. Benchmark problems allow us to test our adaptive algorithms in some real-world problems. Results in conducted experiments showed that adaptive learning algorithms for BNCs work as expected. They are able to perform a more artful *cost-quality* trade-off when compared against its non-adaptive versions and also efficient to cope with drifting concepts.

Finally, we provide the conclusions with a summary of the main contributions of our work in the area of machine learning and finish with an outlook on future research taking into account the general open issues arising from the present work.

Chapter 1

Learning Probabilistic Models

1.1 Introduction

Almost all work in learning Bayesian Networks can be viewed in terms of learning a probabilistic model from data. Given the available data and some background knowledge the main goal is to build probabilistic models that best fit the data in some sense. Probability theory and statistical inference [91] give a natural framework for the problem of learning a probabilistic model from data. As described in [4], most learning problems can be solved in the following form. First, a suitable class of model (*class-model*) such as *Bayesian Network*, neural network, decision tree, etc. is chosen, based on our domain knowledge. Next, within this class-model a *structure* is chosen. This structure defines a *parametric class* of models. Finally, the parameters are estimated based on a sample. The problem of choosing the structure, or a parametric class of models, is called the *model selection problem*.

This chapter provides an overview of some basic definitions and concepts of statistical inference applied to *model selection*, which are involved in the problem of learning probabilistic models from data. Model selection can be approached as a discrete optimization problem where a model selection criterion that measures the quality of each candidate model is optimized in the space of feasible hypotheses. Thus, model selection can be viewed as an optimization problem with two separate issues. First, how to search the space of candidate models. Second, what model selection criterion to optimize for. The main purpose of this chapter is to give a theoretical background in order to contrast the fundamental philosophies

underlying the derivation of different model selection criteria which are not always found all together in the specialized literature. However, the range of concepts covered here is slightly wider than strictly necessary for the rest of the thesis, in order to point out some connections between the problem of learning probabilistic models and the particular problem of learning Bayesian Networks.

1.2 Statistical Preliminaries

The following is a list of references to the material covered in this section. The notational conventions are mainly based on the tutorials [53, 58, 74]. The basic concepts of parametric probabilistic models are mainly based on the tutorial [53] and the book [103]. The comparison between the *frequentist* and *Bayesian* approaches to model selection is based on [58, 89, 90, 114]. Through all this thesis we will use capital letters to denote random variables (e.g. X, Y) and lowercase letters to denote specific values for these variables (e.g. x, y). Bold uppercase letters denote sets of variables (e.g. $\mathbf{X}, \mathbf{Y}$) and bold lowercase letters denote assignments (configuration) of values to the variables in these sets (e.g. $\mathbf{x}, \mathbf{y}$).

1.2.1 Basic Concepts

Probability theory views the world as a set of random variables $X_1, X_2, \ldots, X_n$, each of which has a domain of possible values. The key concept is the *joint probability distribution* $P(X_1, X_2, \ldots, X_n)$, which specifies a probability for each possible combination of values for all the random variables. Let us consider a finite set $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ of random variables where each variable X_i may take on values from its domain Ω_{X_i}. Suppose that we have a random experiment for $\mathbf{X}$, we run the experiment and at time t we observe a particular outcome $\mathbf{x}^{(t)} = (x_1, x_2, \ldots, x_n)$. We call this outcome a *case* or *example*, that is, an instance of the random vector $\mathbf{X}^{(t)}$. A case is said to be *complete* if every variable from $\mathbf{X}$ has a state assigned to it. Otherwise, the case is said to be *incomplete*.

Definition 1. A **dataset** or **data** $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ is the result from a random experiment in which N observed cases are sampled independently from some joint probability distribution $P(X_1, X_2, \ldots, X_n)$ over $\mathbf{X}$. In this case we say that data $\mathcal{D}$ is a random sample of N independent and identically distributed (*i.i.d.*) examples.

Definition 2. A dataset is **complete** if all its cases are complete. Otherwise, we say that the dataset has *missing values*.

A probabilistic model M for a set $\mathbf{X}$ of random variables is a set of joint probabilistic distributions of a same functional form. Usually, a probabilistic model M is specified using a parameter set $\mathbf{\Theta} \in \Omega_{\mathbf{\Theta}}$, where $\Omega_{\mathbf{\Theta}}$ is some parameter space. Almost all probabilistic models are in fact *parametric families* of models; that is, they are models governed by one or more parameters that can be adjusted to fit the random process being modeled. We further consider parametric families of models and define a probabilistic model more formally as follows:

Definition 3. A **probabilistic model** M for a set $\mathbf{X}$ of random variables is a set of joint probability distributions parameterized by a set of parameters $\mathbf{\Theta}_M \in \Omega_{\mathbf{\Theta}}$, that is,

$$M \equiv \{P(\mathbf{X} \mid \mathbf{\Theta}) \mid \mathbf{\Theta} \in \Omega_{\mathbf{\Theta}}\}$$

For example, the family of all normal distributions on $\Re$ is a probabilistic model parameterized by $\mathbf{\Theta} = (\mu, \sigma^2)$ where μ is the *mean* and σ^2 is the *variance* of the distribution. The family of all Bayesian Networks for all possible structures is a *class-model*, but not a *probabilistic model*. Instead, for some fixed structure S, the set of all possible joint probability distributions $P(\mathbf{X} \mid S, \mathbf{\Theta}_S)$ indexed by $\mathbf{\Theta}_S$ is a probabilistic model.

In learning probabilistic models from data we are interested in finding the best explanations for the data from a set of possible explanations and any prior knowledge held by the learner. These explanations are specified by a set of hypotheses that we are considering for the current learning task.

Definition 4. Let M be a probabilistic model parameterized by a set of parameters $\mathbf{\Theta}_M$. A **point hypothesis** M^h is a probability distribution from M with a particular parameter setting that can be tested.

Thus, each legal assignment of values to the parameters in $\mathbf{\Theta}_M$ defines a *point hypothesis* $M^h \in M$, that is, a single probability distribution. For example, the standard normal distribution $\mathcal{N}(0, 1)$ is a point hypothesis. A Bayesian network with a fixed structure S is a probabilistic model containing a set of point hypotheses, each of them corresponding to different choices of the probability parameters.

To facilitate the reading, occasionally we may use the word "*model*" instead of probabilistic model and the word "*hypotheses*" as a generic term, referring to some possibilities that we are considering in the current learning task (this can refer to both *models* and *point hypotheses* according to the actual context).

1.2.2 Likelihood

The concept of likelihood plays a central role in the classical statistical inference. The likelihood measures the "*goodness of fit*" provided by the observed data given a model. Let us consider a probabilistic model M that defines a probability distribution over datasets $\mathcal{D}$.

Definition 5. The **likelihood** of the model M given $\mathcal{D}$ is the probability of $\mathcal{D}$ given that model, that is,

$$\mathcal{L}(M : \mathcal{D}) \equiv P(\mathcal{D} \mid M) \tag{1.1}$$

The likelihood viewed as a function of point hypothesis for a fixed data $\mathcal{D}$ is said to be the *likelihood function*. Obviously, if we say that a point hypothesis M^h fits the data $\mathcal{D}$ well, this means that the likelihood $\mathcal{L}(M^h : \mathcal{D})$ is high. The importance of the likelihood function is summarized by the Fisher's *likelihood principle* [41] that states that "*a hypothesis is more plausible or likely than another, in the light only of the observed data if it makes those data more probable*".

A likelihood function can provide no absolute statement about the validity of any candidate models, but only relative comparison among them [91]. A criterion that is commonly used to compare two hypotheses $M^{h_1}, M^{h_2} \in \mathcal{M}$ is the *likelihood ratio* defined by:

$$\mathcal{LR}(M^{h_1}, M^{h_2} : \mathcal{D}) \equiv \frac{\mathcal{L}(M^{h_1} : \mathcal{D})}{\mathcal{L}(M^{h_2} : \mathcal{D})} \tag{1.2}$$

Often we use the natural logarithm (*log*) of the likelihood instead of the likelihood since products converted to summations reduce problems of numerical underflow.

Definition 6. The **log-likelihood** of a model M given the observed dataset $\mathcal{D}$ is the natural logarithm of the likelihood of M given $\mathcal{D}$, that is,

$$l(M : \mathcal{D}) \equiv log\ \mathcal{L}(M : \mathcal{D}) \tag{1.3}$$

We further introduce the concept of *sufficient statistic*, which will turn out to be very useful to compute the likelihood of a model. Suppose we observe the dataset $\mathcal{D}$ with a view of gaining information about the parameter $\mathbf{\Theta}_M \in \Omega_{\mathbf{\Theta}_M} \subset \Re^k$. A *statistic* is a function $\mathbf{T} : \mathcal{D} \rightarrow \Re^k$ that gives us an useful information about the data. Therefore, a statistic is an observable, real-valued function of the observations. The term observable means that the function should not contain any unknown parameters.

A *sufficient statistic* is just a function of dataset that summarizes the relevant information for computing the likelihood. More precisely, a sufficient statistic summarizes all the available information in $\mathcal{D}$ that allows to make inference on the parameter $\mathbf{\Theta}$. Therefore, if we gather and store sufficient statistics from a dataset, we can drop the original data that we do not lose any information related to the parameter. The concept of sufficient statistics is widely discussed, for instance, in [74, 89, 91].

1.2.3 Frequentist versus Bayesian Approach to Statistical Inference

There are nowadays two main approaches of statistical thought: the *frequentist, Fisherian* approach and the *Bayesian* approach. The former is named after Sir Ronald Fisher and combines the *frequency approach* [41] (unbiased estimators, hypothesis tests, etc.) with *likelihood* methods. The *Bayesian* approach is named after the Reverend Thomas Bayes and refers to such concepts as *prior and posterior knowledge*, *prior and posterior predictive distributions*, *Bayes estimators*, etc. Both, *Fisherian* and *Bayesian* statistics have different definitions of what it means to be a probability. Whereas a *frequentist* probability is a physical property of the world measured by repeated trials (e.g., the probability that a coin will land heads), a *Bayesian* probability describes the person's degree of belief in that event (e.g. your degree of belief that the coin will land heads).

We further assume that some data is observed and we wish to make inferences about one or more unknown features of the physical system which have been generated these data. These unknown features may be expressed in terms of a discrete set of hypothesis (in *model selection* problems) or in terms of a parameter space (in *parameter estimation* problems).

Bayesian inference methods are distinguished from *frequentist* methods by the fact that Bayesian inference is based on a different view of what it means to learn from data. In fre-

quentist inference the unknown features take single values whereas in Bayesian learning the unknown features are treated as *random variables* which have both, *prior* and *posterior distribution*. Before observing the data, our prior beliefs can be expressed in a prior probability distribution that represents the knowledge we have about the unknown features. After observing the data our revised beliefs are captured by a posterior distribution over the unknown features. *Bayes' theorem* (also known as *Bayes' rule*) is the main tool in Bayesian inference.

Theorem 1.2.1. *(Bayes) Given two events* $\mathbf{E}$ *and* $\mathbf{F}$ *such that* $P(\mathbf{E}) \neq 0$ *and* $P(\mathbf{F}) \neq 0$*, we have*

$$P(\mathbf{E} \mid \mathbf{F}) = \frac{P(\mathbf{F} \mid \mathbf{E})P(\mathbf{E})}{P(\mathbf{F})}$$

In Bayesian inference, Bayes' theorem can be expressed in its more memorable form:

$$posterior \propto prior \times likelihood$$

by combining the *prior distribution* and the *likelihood* of the observed data in order to derive the *posterior distribution*. This expression summarizes the way in which we should modify our beliefs about the unknown quantities in order to take into account the observed data [89].

Informally speaking, the Bayesian methodology briefly comprises the following steps to make inference:

1. **Modeling Priors**: Assign priors to all the unknown quantities (parameters or models).

2. **Compute the Likelihood**: Observe the data and compute the likelihood of a hypothesis given the data.

3. **Prior-to-Posterior Computation**: Apply Bayes' theorem to derive the posterior probability of the unknown quantities given the data.

4. **Make Inference**: Use the posterior probability to derive appropriate inference: to compute a point estimate of the parameters, to do model selection, or in general, to obtain the predictive distributions.

In the Bayesian methodology, therefore, inference is a continuous, dynamic process in which new data are used to revise the current knowledge. As pointed out in [114] when

Bayesian methods are coupled with the conditional independence of the data given the parameters, the inference procedure can process data as a whole (in a batch mode), as frequentist methods do, but it can also process data sequentially, one at the time, and the final results will be the same. This incremental nature is a crucial advantage of Bayesian methods. On the light of new data they do not require reprocessing all the data seen so far.

Bayesian methods are distinguished from frequentist methods also by the fact that they can be applied to every type of problem and often represent the optimal method to use. However, practical implementation of Bayesian methods usually requires substantial computation. This computational requirement is essential to calculate summaries of the posterior distribution. The combination of the likelihood and prior generally produces a posterior distribution too complex to summarization, even if the two constituents separately are sufficiently simple. For some cases, however, the prior distributions and the likelihood have sufficiently convenient forms that enable the necessary results to be obtained in closed-form solutions. In spite of this, in practice, we need to work with much more complex models. On the other hand, Bayesian methods require the specification of prior distributions to accurately reflect the available prior information, which can also lead to complex modeling.

1.3 The Parameter Estimation Problem

The uncertainty in parameters is an issue in model selection. In the *frequentist* approaches to parameter estimation the parameters are regarded as having a "*true*" but unknown value which can be estimated from the data, using, for instance, the *maximum likelihood estimator*. The obtained estimate then is used as the predictive probability. Bayesian approaches, alternatively, regard the parameters as random variables and the uncertainty about parameters is expressed in terms of a prior distribution. Bayes' theorem is then used to compute the posterior distribution given the observed data, which can be further used to make predictions. Bayesian methods usually involve integrals over the parameters to make inference, whereas frequentist methods rely on optimization procedures. In the next subsections we briefly overview the derivation of the mostly used parameter estimators in statistical inference, under both the *frequentist* and *Bayesian* framework[1].

[1] All the material exposed in this section is mainly based on the references [58, 74, 114].

1.3.1 Maximum Likelihood Estimator

Definition 7. An **estimator** for a parameter θ is a function that computes a putative value for $\hat{\theta} = \hat{\theta}(\mathcal{D})$ given the observed data $\mathcal{D}$.

The intuitive requirement that $\hat{\theta}$ should be "*close*" to the actual value is formalized by the notions of *bias* and *consistency*. We can quantify the bias of an estimator as the expected value of the error, that is, $bias(\hat{\theta}) \equiv E(\hat{\theta}(\mathcal{D}) \mid \theta) - \theta$. An estimator is said to be *unbiased* if its bias is 0 and *consistent* if it converges to the true value as data increases.

A *frequentist* approach to parameter estimation is to maximize the likelihood function. This estimator is known as the *Maximum Likelihood estimator* [41]. Let us consider a probabilistic model M parameterized by some parameter set $\mathbf{\Theta}_M \in \Omega_{\mathbf{\Theta}_M}$. We can consider the likelihood as a function of point hypothesis $M^h \in M$, where each hypothesis corresponds to a different assignment of the parameter values over $\Omega_{\mathbf{\Theta}_M}$. Hence, we can define the likelihood function over the parameter space as follows:

Definition 8. The **likelihood function** of a parameter $\mathbf{\Theta}_M \in \Omega_{\mathbf{\Theta}_M}$ given data $\mathcal{D}$ is the probability of $\mathcal{D}$ given that parameter, that is

$$\mathcal{L}(\mathbf{\Theta} : \mathcal{D}, M) \equiv P(\mathcal{D} \mid M, \mathbf{\Theta}) \tag{1.4}$$

Definition 9. The **Maximum Likelihood (ML) estimate** of a parameter $\mathbf{\Theta}_M$ is that value (parameter settings) the maximizes the likelihood function of $\mathbf{\Theta}_M$ given $\mathcal{D}$, that is

$$\hat{\mathbf{\Theta}}_{\mathcal{ML}}(\mathcal{D}, M) = \arg \max_{\mathbf{\Theta}_M \in \Omega_{\mathbf{\Theta}_M}} \mathcal{L}(\mathbf{\Theta}_M : \mathcal{D}, M) \tag{1.5}$$

ML estimators are in most cases *consistent*, but in general *biased*. However, when $N \to \infty$ they become *unbiased* and *efficient* [41]. Hence, the ML estimator converges to the best possible value, as close to the true value as possible as the number of examples grows given a particular dataset. Moreover, the variance of the ML estimator is no greater than that of any other unbiased estimator.

The ML estimate can be viewed as a point estimate. Because the likelihood function is bounded above, the ML estimate always exists. However, the likelihood function may not

have a unique maximum in the parameter space, which means that several hypotheses are equally likely to describe the data [91]. Before going further, let us give an example. This will allow us to illustrate the behavior of the likelihood function for the single parameter of the thumbtack example[2]. If we throw the thumbtack up N times in the air, it will come to rest either on its points (*heads*) or its head (*tails*). Thus, each observation is the realization of a Bernoulli trial with one of two outcomes: *head* (`H`) or *tail* (`T`). The single parameter θ represents the probability with which an individual toss lands heads. The likelihood function of the parameter θ for the sequence `H, T, T, H, H` is given by $\theta^3(1-\theta)^2$. Figure 1.1 depicts this likelihood function. In general, to compute the likelihood in the thumbtack example we only need to obtain the sufficient statistics from data, that is the number of heads N_h and the number of tails N_t. The likelihood function is then $\mathcal{L}(\theta : \mathcal{D}) = \theta^{N_h}(1-\theta)^{N_t}$.

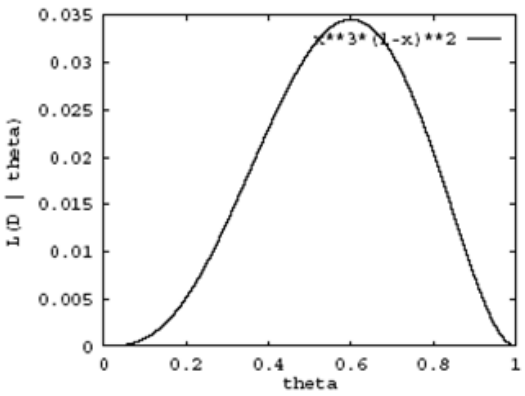

Figure 1.1: A likelihood function for the single parameter of the thumbtack example

The likelihood function is monotonically related to the log-likelihood. Therefore, maximizing the one is equivalent to maximizing the other. However, the log-likelihood is more convenient to work with, since products converted to summations reduce problems of numerical underflow. In particular, for the thumbtack example we have:

$$l(\theta : \mathcal{D}) = N_h \; log \; \theta + N_t \; log(1-\theta)$$

The value that maximizes the likelihood, that is, the ML estimate of θ is given by the observed relative frequencies:

$$\hat{\theta}_{\mathcal{ML}} = \frac{N_h}{(N_h + N_t)}$$

[2]This example is taken from [58, 74] and Figure 1.1 from [74].

1.3.2 Bayesian Estimators

In the Bayesian approach to parameter estimation we express our uncertainty on the parameters by regarding $\mathbf{\Theta}_M$ as a random vector over $\Omega_{\mathbf{\Theta}_M}$. The first step is to assign a prior distribution $P(\mathbf{\Theta}_M \mid M)$ in order to reflect our prior beliefs in the possible values of the parameters. The next step is to derive the *posterior distribution* of $\mathbf{\Theta}_M$ given the observed data $\mathcal{D}$ by means of Bayes' theorem:

$$P(\mathbf{\Theta}_M \mid \mathcal{D}, M) = \frac{P(\mathcal{D} \mid M, \mathbf{\Theta}_M) P(\mathbf{\Theta}_M \mid M)}{P(\mathcal{D} \mid M)} \tag{1.6}$$

The resulting posterior distribution $P(\mathbf{\Theta}_M \mid \mathcal{D}, M)$ gives the probability of the parameters after observing the data. The term $P(\mathcal{D} \mid M, \mathbf{\Theta}_M)$ is the likelihood $\mathcal{L}(\mathbf{\Theta}_M : \mathcal{D}, M)$. Whereas the likelihood function plays a central role in classical methods for parameter estimation, for the Bayesian approach it is only the instrument to perform the *prior-to-posterior* computation via Bayes' theorem [114]. The last term $P(\mathcal{D} \mid M)$ in Equation 1.6 is the probability of the data given the model and is computed by integrating over the parameter space:

$$P(\mathcal{D} \mid M) = \int P(\mathcal{D} \mid M, \mathbf{\Theta}_M) P(\mathbf{\Theta}_M \mid M) \, d\mathbf{\Theta}_M \tag{1.7}$$

This term is called the *marginal density of data* [114] to point out the fact that it is no longer conditioned on $\mathbf{\Theta}_M$. In fact, it is just a *normalization constant* that is independent of the values of the parameters, and hence, this is usually ignored. However, the *marginal density of data* is very important when comparing different models. The last step is to use the obtained posterior distribution in order to compute the parameter estimates.

The Full Bayesian Approach

The key idea in Bayesian statistics is to work with full distributions of parameters instead of single estimates. In computations that require a value for a certain parameter, instead of choosing a single "*best value*", we must obtain an estimate by averaging (integrating) the probabilities over the parameter space weighting their results by the resulting posterior probabilities. This method is called *marginalising* over the parameter space and this approach is known as the *full Bayesian approach*. A standard point estimate of an individual parameter $\theta \in \mathbf{\Theta}_M$ is the posterior expectation $E_{P(\mathbf{\Theta}_M \mid \mathcal{D}, M)}(\theta)$, that is, the expectation of θ with respect

to the posterior distribution [114]. We call this estimate the **Bayesian estimate** and denote it by $\hat{\theta}_{Bayes}$.

Definition 10. The **Bayesian estimate** of an individual parameter $\theta \in \mathbf{\Theta}_M$ is the posterior expectation of θ with respect to the posterior distribution, that is,

$$\hat{\theta}_{Bayesian}(\mathcal{D}, M) = E_{P(\mathbf{\Theta}_M|\mathcal{D},M)}(\theta) = \int \theta \; P(\mathbf{\Theta}_M \mid \mathcal{D}, M) \; d\mathbf{\Theta}_M \tag{1.8}$$

Although the full Bayesian approach gives the optimal method for performing statistical inference, the exact use of those tools can become impractical. However, for the exponential family of distribution (e.g. *binomial*, *multinomial*, *normal*, etc.) under some determined assumptions, these computations can be done efficiently and in a closed form [58].

The MAP Approach

When the exact Bayesian inference is impossible, there are many methods that approximate it. The simplest method is to approximate the posterior with a discrete distribution concentrated at its maximum value. This gives a single estimate for all the parameters. This approach is called *maximum a posteriori estimation*. The estimator is called the MAP estimator.

Definition 11. The **MAP estimate** of a parameter vector $\mathbf{\Theta}_M$ is the value of the parameter $\mathbf{\Theta}$ that maximizes the posterior distribution of $\mathbf{\Theta}_M$ given data $\mathcal{D}$, that is,

$$\hat{\mathbf{\Theta}}_{\mathcal{MAP}}(\mathcal{D}, M) = \arg \max_{\mathbf{\Theta}_M \in \Omega_{\mathbf{\Theta}_M}} P(\mathbf{\Theta}_M \mid \mathcal{D}, M) \tag{1.9}$$

The MAP estimator is the Bayesian counterpart to the ML estimator, and they become equivalent when the prior is uniform. In this case only the likelihood term is maximized. We can consider the Bayesian and MAP estimates as different ways of summarizing the posterior distribution $P(\mathbf{\Theta}_M \mid \mathcal{D}, M)$ around its peak. Note that the MAP estimate only takes into account the location of the peak, while the Bayesian estimate takes into account the location as well as the sharpness of the peak [128]. As stated in [31], the advantages of bayesian estimates is that for smaller data sets, the results tend to be more robust and generally, less sensitive to the presence of zeroes in marginal counts.

Conjugate Priors

The Bayesian approach requires the specification of prior distributions. A prior distribution is said to be *informative* if this is known; otherwise it is *non-informative*. The requirement of a prior distribution can be both an advantage and a disadvantage. It is an advantage when the prior is informative because the shape of the prior is taken into account to obtain the posterior. It is a disadvantage when it is *non-informative* because we need to asses a great number of parameters. Even when no information is available with respect to the parameter being estimated, it is necessary to choose a prior. In such cases it is desirable to select a prior that will have the least influence on the posterior. A commonly used approach is to use the *uniform distribution*.

Modeling priors has been traditionally a compromise between a realistic assessment of beliefs and choosing a convenient approximation to simplify the analytic calculations [58]. A well-known strategy is to choose a prior with a suitable form so that the posterior belongs to the same functional family as the prior. Prior and posterior chosen in this way are said to be *conjugate*. The choice of the conjugate family depends on the likelihood. Conjugate families are useful because for many distributions they allow the sequential updating of the posterior distribution. In many cases we have a *closed-form* solution for the *prior-to-posterior* computations [58]. In Section 2.5.2 we derive the Bayesian estimates for a Bayesian network with discrete variables by using Dirichlet priors as conjugate to the multinomial distribution.

1.4 The Model Selection Problem

Model selection is a subject of statistical inference concerned with the selection among a set of competing models. Let $P(X_1, X_2, ..., X_n)$ denote the true, an unknown joint probability distribution over a finite set $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ of random variables for a domain under study. Let $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ be a set of candidate probabilistic models, each of them containing a set of point hypotheses M^h. The model selection problem can be formally posed as follows. Given a training dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ of N *i.i.d.* examples of $\mathbf{X}$ sampled from the unknown joint distribution $P(\mathbf{X})$, and some prior information ξ (background knowledge), find the model $M \in \mathcal{M}$ containing the hypothesis $M^h \in M$ that "*best fits*" $\mathcal{D}$. The notion of "*best fits*" is defined via a *model selection criterion* [17].

Let us give a standard example[3] that allows us to better illustrate a model selection problem. Suppose that we choose the class-model of polynomials to fit a given number of points in the plane. Let the set of candidate models $\mathcal{M}$ be the set of different polynomials where each $M_k \in \mathcal{M}$ represents the set of k^{th} degree polynomials, each containing a set of point hypotheses (e.g. individual polynomials). If we are interested in selecting both the degree of a polynomial and its corresponding parameters (a hypothesis $M^h \in \mathcal{M}$), it is a *hypothesis selection problem*. If we are only interested in selecting the degree of the polynomial (a model $M_k \in \mathcal{M}$) it is a *model selection problem* [53]. In the context of machine learning, learning problems are rather posed as hypothesis selection problems.

Further we consider a model selection problem. We assume that some data is observed, a set of models is given and that statistical inference is model-based. In this context, model selection can be viewed as a discrete optimization problem where there are two separate issues. First, how to search over the space of models. Second, what model selection criterion to optimize for. The choice of an appropriate *model selection criterion* (also called *scoring function* or *score*) is crucial for model selection tasks. In this chapter we focus on model selection criteria.

All model selection criteria that are used in practice either implicitly or explicitly choose a trade-off between *goodness of fit* and *complexity* of the models involved [53]. Indeed, model selection makes a *bias-variance* trade-off in order to select a model with the appropriate complexity [25, 57]. Models that are too simple has too much bias, they therefore fit the data poorly and not are able to describe the essential features of the data. They will *underfit* the data. On the contrary, models that are too complex (with too much parameters) fit the training data very well but have too much *variance*. They will *overfit* the data. Intuitively, we need the right balance. This *bias-variance* trade-off is automatically regularized by the model selection criterion.

1.5 Model Selection Criteria

There are many competing approaches on how to choose the model selection criterion reflecting different paradigms for inductive inference:

[3] This example is taken from the tutorial [53].

1. The *frequentist* approach.
2. The *Bayesian* approach.
3. The *information-theoretic* approach.
4. The *predictive* approach.

As pointed out in [4], the choice of a model selection criterion reflects not only a paradigm for inductive inference, but also our prior beliefs and intuitions about the domain of induction according to the learning goals. However, it is often difficult to express in the mathematical formalism of a model selection criterion our beliefs and intuitions. It is therefore of interest to empirically compare how different model selection criteria perform the model selection task in a particular domain. Toward this end, one of the main goals of this thesis was to compare several model selection criteria in the task of learning Bayesian network classifiers from data using class-models of increasing complexity. In the next sections we briefly review the commonly used model selection criteria, which we used in our experiments:

1. Maximum likelihood criterion (MLC).
2. Bayesian score.
3. Bayesian information criterion (BIC).
4. Akaike's information criterion (AIC).
5. Minimum description length (MDL) score .
6. Cross-validation (k-Fold-CV) score.
7. Prequential (Preq) score.

MLC is derived from the frequentist *maximum likelihood principle*. Both AIC and MDL score are derived from *information-theoretic* arguments. AIC is based on two basic concepts of information theory such as the *entropy* and the *Kullback-Leibler (K-L) divergence*. Minimizing AIC is approximately equivalent to minimizing the *expected K-L divergence* between the true distribution and the approximating distribution. MDL principle attempts to describe the data using a *minimum encoding approach*. When given a choice between models that model the

data "*similarly well*", MDL will choose the one with the least complexity. Both Bayes and BIC scores are derived from the Bayesian framework. The Bayesian score is the *log of the (relative) posterior probability*. But assuming uniform priors over models we obtain the *log marginal likelihood*, one of the most used scores in learning Bayesian networks. BIC is a large-sample approximation to the *log marginal likelihood* derived from the Laplace approximation. Hence BIC asymptotically corresponds to choosing the model with the largest posterior probability. MLC, MDL, BIC and AIC are based on the *log-likelihood* while the Bayesian score is based on the *log marginal likelihood*. Although derived from different frameworks, AIC, BIC and MDL are all *penalized* log-likelihood scores. Moreover, BIC has an alternative formulation in terms of information-theoretic concepts: under some conditions maximize BIC is equivalent to minimize MDL. All the log-likelihood based scores are easy to use and does not require evaluation of prior distributions. The Bayesian score, instead, use prior knowledge to set priors over model's structures and parameters.

Likelihood based scores such as MLC prefer more complex models. They can overfit the data, specially if we have few data to learn. Penalized likelihood scores, such as AIC, BIC and MDL can be viewed as a mathematically precise form of *Occam's Razor* [9] that states that "*given two equally predictive theories, choose the simpler*". According to this principle, we should seek simpler models over complex ones. These scores are most frequently applied to model selection problems dealing with overfitting. In practice, penalized likelihood scores allow finding a more optimal trade-off between the *complexity* of models and the *goodness-of-fit* to the data than MLC. However as the number of examples N grows very large, the emphasis of the MDL/BIC score is on the log-likelihood term. Therefore, asymptotically MDL will pick the same model as MLC does.

An alternative approach to model selection is to choose a model for future prediction. In this case the model selecion task is to choose a model so that the resulting predictive distribution yields the most accurate predictions for future data. This approach requires the definition of a *loss function* in order to assess the quality of a model in terms of its predictive accuracy. One natural way to measure the predictive performance is provided by *cross-validation* [132]. The cross-validation method evaluates the predictive performance of data models by repeatedly splitting the data into k subsets of equal size. Each subset is used in turn as a validation set, while the union of the remaining k-1 sets are used as training

set. The cross-validation score k-Fold-CV is the resulting averaged expected loss. Other alternative approach is based on the Dawid's *prequential* approach [35] where the model selection criterion is computed *predictively* and *sequentially* through a sequential updating of the predictive distribution. The *prequential* score is the resulting cumulative loss.

Interested readers can follow some of the following references for an in-depth study of the derivation of different model selection criteria. The book [92] provides an overview of the general problem of model selection, covering AIC and cross-validation scores. The book [116] gives a detailed development of the MDL score. Schwarz in [125] gives the derivation of BIC. Bozdogan in [12] gives the derivation of AIC and a discussion of its use. In [53] Grunwald provides a very nice tutorial of MDL. A survey of cross-validation approaches is given in [110]. In [17] the authors provide an interesting discussion about the philosophies and general principles that should guide model-based inferences on the science and compare AIC with BIC. Finally, Ghahramani in [51] provides an overview of the model selection problem for the particular problem of unsupervised learning with emphasis on Bayesian criteria and approaches to approximate the posterior distributions of a model.

1.5.1 Maximum Likelihood Criterion (MLC)

When models are learned in a *frequentist* approach, they are compared on the basis of the likelihood they attain. The maximum likelihood criterion (MLC) is simple the maximized log-likelihood of a model M given the training data $\mathcal{D}$. From Definition 9 of the ML estimate we know that the hypothesis the maximizes the likelihood is that hypothesis that we obtain when parameters are set to the ML estimate $\hat{\mathbf{\Theta}}_{\mathcal{ML}}$.

Definition 12. The **Maximum likelihood (ML) hypothesis** of a probabilistic model M is the hypothesis $M^h \in M$ that maximizes the log-likelihood of M given data $\mathcal{D}$, that is,

$$M_{\mathcal{ML}} \equiv \arg \max_{M^h \in \mathcal{M}} l(M^h : \mathcal{D}) \tag{1.10}$$

Definition 13. The **Maximum Likelihood Criterion (MLC)** of a model M given data $\mathcal{D}$ is the log-likelihood of its ML hypothesis, that is:

$$Score_{\mathsf{MLC}}(M, \mathcal{D}) \equiv l(M_{\mathcal{ML}} : \mathcal{D}) \tag{1.11}$$

where:

$$l(M_{\mathcal{ML}} : \mathcal{D}) = l(\hat{\mathbf{\Theta}}_{\mathcal{ML}} : \mathcal{D}, M) = P(\mathcal{D} \mid M, \hat{\mathbf{\Theta}}_{\mathcal{ML}}) \tag{1.12}$$

Maximizing the likelihood will usually *overfit* the data and can lead to the selection of models more complex than the optimal or true one. *Overfiting* results because the likelihood is based on a single ML hypothesis in the hypothesis space for each possible model. As the model becomes more complex this space increases, and hence, the mode of the likelihood function cannot decrease, instead it tends to increase [3]. Therefore, ML approaches always chooses a maximally complex model. For avoiding *overfitting* we can use *penalized likelihood scores*, which bias the MLC to prefer simpler models.

Penalized log-likelihood scores

Definition 14. A **penalized log-likelihood score** for a model M given data $\mathcal{D}$ is given by the following general formula [86]:

$$Score_{\mathsf{MLC}}(M, \mathcal{D}) - f(N) \parallel M \parallel \tag{1.13}$$

where $f(N)$ is a non-negative penalty function, N is the number of data examples and $\parallel M \parallel$ is the dimension of the model defined as the number of its free parameters.

This formula for *penalized log-likelihood* scores explicitly shows the trade-off between the first term - *the fitness to data*, and the second term - *the penalty complexity*. In the next sections we show that although BIC, AIC and MDL are derived under different inductive frameworks, in practice, they all can be derived from this general formula. For instance, when $f(N) = 1$ we obtain the AIC score and when $f(N) = \frac{1}{2}\ log\ N$ we obtain the BIC/MDL scores. Therefore, increasingly for larger number of observations, the model with the most BIC/lest MDL will tend to be simpler than the model with the most AIC.

1.5.2 Bayesian Criteria

In the Bayesian approach to model selection we express our uncertainty on the models by regarding $\mathcal{M}$ as a random discrete variable whose states correspond to the candidate probabilistic models $\{M_1, M_2, \ldots, M_m\}$. Following the Bayesian methododology from Section 1.2.3,

for each candidate model $M \in \mathcal{M}$ we need to place a prior distribution $P(M)$ that reflects our relative beliefs about that model. After observing the data $\mathcal{D}$, we can use Bayes' theorem for the *prior-to-posterior* computation. The denominator $P(\mathcal{D} \mid M)$ can be dropped (a normalization constant independent of the parameters). This yields to the classical formulation of Bayes' theorem for model selection:

$$P(M \mid \mathcal{D}) \propto P(M)P(\mathcal{D} \mid M) \tag{1.14}$$

where $P(\mathcal{D} \mid M)$ is the *marginal likelihood* of the model M given data $\mathcal{D}$ defined by the following integral over the parameter space:

$$P(\mathcal{D} \mid M) = \int P(\mathcal{D} \mid M, \boldsymbol{\Theta}_M)P(\boldsymbol{\Theta}_M \mid M)\ d\boldsymbol{\Theta}_M \tag{1.15}$$

According to the marginalisation principle, the correct way to compare different models in the Bayesian framework would be to use the *full Bayesian approach*. This approach, also known as *model averaging*, uses all the candidate models for prediction by weighting their results by their respective posterior probabilities. However, averaging over all the models is a computationally demanding approach. The simplest and therefore most common approach is pick the model with highest posterior probability, that is, the MAP model. A criterion that is often used for numerical convenience is the log of the relative posterior probability. We call this criterion the Bayesian score.

Definition 15. The **Bayesian score** is the log of the relative posterior probability of a model M given the training data $\mathcal{D}$, that is,

$$Score_{\mathsf{Bayesian}}(M, \mathcal{D}) \equiv\ log\ P(M) +\ log\ P(\mathcal{D} \mid M) \tag{1.16}$$

Thus, in practice the MAP approach to model selection could be simple addressed as follows. For each candidate model $M \in \mathcal{M}$: *i*) to assess an appropriate prior distribution $P(M)$; *ii*) to carefully compute the marginal likelihood $P(\mathcal{D} \mid M)$. When the number of possible models is large, the assessment of the priors will be *intractable*. One straightforward solution is to ignore the *log prior* component assuming that all the candidate models are equally probably apriori. As a result, the Bayesian score is simple reduced to the *log marginal likelihood*.

The effect of the model prior is equivalent to penalizing overly complex models. However, this is not strictly necessary, since the marginal likelihood term has a similar effect [101]. As argued in [51], the *marginal likelihood* has a very interesting interpretation: this is the probability of generating data $\mathcal{D}$ from parameters that are *randomly sampled* from the parameter prior $P(\boldsymbol{\Theta}_M \mid M)$. In contrast with the MLC, which tends to increase as the model complexity increases and can *overfit* the data, the marginal likelihood can decrease as the model becomes more complex. In a more complex dataset, sampling random parameter values can generate a wider range of possible datasets, but since the marginal likelihood is a probability over data sets it must integrate to one. Therefore, very complex models can account for many datasets, but distributing the density over all the data sets inevitably results in more modest marginal likelihood. Otherwise simpler models can reach high marginal likelihood, but only for a limited set of datasets. This property of the decreasing in the marginal likelihood as models become more complex is related to the *Occam's Razor* [9]. Consequently, the Bayesian score tends to select models less complex than MLC does, thus performing a more optimal trade-off between *complexity* and *fitness to data*.

1.5.3 Bayesian Information Criterion (BIC)

One of the main difficulties in the implementation of the Bayesian approach is the computation of the marginal likelihood. This integration problem is difficult because the integral 1.15 is typically of high dimension and very expensive to compute. We can alternatively approximate the marginal likelihood using approximating methods such as *stochastic simulation*, *Laplace approximations* and *Monte Carlo Methods*. An overview of these approximating methods for the marginal likelihood can be found, for example, in [51].

The Bayesian Information Criterion (Schwarz, 1978) [125] is a quick and easy way to compute a large sample approximation to the marginal likelihood. BIC is derived from the Laplace approximation to the log marginal likelihood as follows[4]:

$$log\ P(\mathcal{D} \mid M) \approx log\ P(\hat{\Theta}_M \mid M) + log\ P(\mathcal{D} \mid \hat{\Theta}_M, M) + \frac{d}{2}\ log\ 2\pi - \frac{d}{2}\ log\ |A| \qquad (1.17)$$

where d is the number of free parameters in the model, A is the $d \times d$ negative of the Hessian

[4]Here we only depict the main ideas underlying the derivation of the BIC score, which are mainly based on the derivation given in [51]. The full derivation of BIC was given by Schwarz in [125].

matrix which measures the curvature of the log posterior at the MAP estimate (also known as the observed information matrix) and $\hat{\Theta}_M$ is some parameter estimate.

BIC can be derived from the Laplace approximation by dropping all terms that not depend on N (the number of data examples): the first and third term in 1.17. Then we can substitute the term $\frac{d}{2}\log|A|$ by $\frac{d}{2}\,logN$ by assuming that in the limit of large N the Hessian A converges to N times a full-rank matrix. As a result, we obtain the BIC approximation where the likelihood is penalized by a term (the BIC penalty) that depends linearly on the number of the parameters in the model:

$$log\ P(\mathcal{D} \mid M) \approx log\ P(\mathcal{D} \mid M, \hat{\Theta}_M) - \frac{d}{2}\ log\ N \tag{1.18}$$

Thus, BIC approaches the Occam's Razor by penalizing overcomplex models. Moreover, since the BIC approximation does not involve the prior we can use it either with the ML or MAP estimates. Assuming that we choose the ML estimate $\hat{\boldsymbol{\Theta}}_{\mathcal{ML}}$ of M, that is, its $M_{\mathcal{ML}}$ hypothesis, and that $\parallel M \parallel$ is the model's dimension (the number of its free parameters d) we can derive the BIC criterion from the BIC approximation as follows:

Definition 16. The **Bayesian Information Criterion** (BIC) of a model M given the training data $\mathcal{D}$ is defined as follows

$$Score_{\mathsf{BIC}}(M, \mathcal{D}) \equiv l(M_{\mathcal{ML}} : \mathcal{D}) - \frac{1}{2} \parallel M \parallel\ log\ N \tag{1.19}$$

The BIC model selection procedure is to choose the model for which the BIC criterion is maximized. The BIC criterion is very attractive because it is extremely easy to compute. However, this simplicity comes at a cost in accuracy. The basic assumption of BIC (the Hessian converges to N times a full-rank matrix) only holds for models in which all the parameters are identifiable and well-determined. And since this is often not true, it can conduce to more biased models.

Finally note that BIC score can be derived from the penalized log-likelihood formula 1.13 when $f(N) = \frac{1}{2}\ log\ N$, thus:

$$Score_{\mathsf{BIC}}(M, \mathcal{D}) = Score_{\mathsf{MLC}}(M, \mathcal{D}) - \frac{1}{2}\ log\ N \parallel M \parallel \tag{1.20}$$

1.5.4 Akaike's Information Criterion (AIC)

The Akaike's Information Criterion (AIC) [2] is a simple criterion based on two basic concepts of the information theory: the *entropy* and the *K-L divergence*. The *entropy* measures the amount of information of a random variable. *K-L divergence* is a measure that compares two distributions. Other concepts of information theory which are very useful for model selection, particularly for learning Bayesian networks are the *mutual information* and the *conditional mutual information*. These concepts are based on the concept of the *conditional entropy*, a measure related to the entropy. In the next subsection we briefly overview these basic concepts of information theory[5]. Then we present the main issues related to the derivation of the AIC which are mainly based on the work [17]. The original derivation of the AIC was given by Bozdogan in [12].

Information Theory: Basic Concepts

Information theory deals with the efficient and accurate *storage*, *transmission*, and *representation* of information. Suppose that we want to transmit symbols x randomly drawn from a probability distribution $P(X)$ over a digital channel (e.g. a symbol x may be a *message* and the digital channel may be the *Internet*). The information of each symbol x is quantified as the number of bits that we need to encode it. Naturally, we should encode our data so that symbols which occur more frequently use fewer bits to encode them. Shannon's source coding theorem tells us that the optimal number of bits to encode a symbol x with probability $P(x)$ is the *negative logarithm* of this probability $-logP(x)$. Therefore, a probabilistic model can also be used to achieve efficient storage, transmission and data compression.

Definition 17. Let X be a discrete random variable taking values x in a finite subset Ω_X. Let P be a probability distribution over X. The **entropy** of the random variable X is defined as follows:

$$H(X) \equiv -\sum_{x \in \Omega_X} P(x)\ log\ P(x) \tag{1.21}$$

In terms of encoding the entropy $H(X)$ can be interpreted as the the expected number of bits needed to store the values of X.

[5] For an in-depth study of the elements of information theory we refer interested readers to [30].

Definition 18. Let X, Y be two discrete random variable taking values in the finite subsets Ω_X and Ω_Y, respectively. The **conditional entropy** of X given Y is defined as follows:

$$H(X \mid Y) \equiv - \sum_{y \in \Omega_Y} P(y) \sum_{x \in \Omega_X} P(x \mid y) \ \log \ P(x \mid y)$$

$H(X \mid Y)$ measures the entropy of X knowing the values of Y. The higher the conditional entropy the more we can predict the state of a variable, knowing the state of the other variable. In terms of encoding, $H(X \mid Y)$ measures the optimal number of bits needed to encode the value of X when the value of Y is given. Intuitively, $H(X \mid Y) \leq H(X)$. The difference between these two values is the **mutual information** between X and Y, which measures the information that Y provide about X. The *conditional mutual information* measures the information that Y provide about X when the value of other variable is known. More formally:

Definition 19. Let X, Y be two discrete random variable taking values in the finite subsets Ω_X and Ω_Y, respectively. The **mutual information** between X and Y is defined as follows:

$$I(X, Y) \equiv H(X \mid Y) - H(X) \tag{1.22}$$

Definition 20. Let X, Y and Z be three discrete random variable taking values in the finite subsets Ω_X, Ω_Y and Ω_Z, respectively. The **conditional mutual information** between X and Y given Z is defined as follows:

$$I(X, Y \mid Z) \equiv H(X \mid Z) + H(Y \mid Z) - H(X, Y \mid Z) \tag{1.23}$$

The *mutual information* is used in model selection to measure the degree of dependence between two random variables. This tell us not only if two variables are dependent but also how close their relationship is. For instance, if $I(X, Y) = 0$ then X and Y are completely independent. Moreover, the mutual information $I(X, Y)$ increases with the increase of the degree of dependence between X and Y. The *conditional mutual information* measures the degree of dependence between two random variables knowing the state of the other variable. Thus if $I(X, Y \mid Z) = 0$ then X and Y are completely independent given Z.

Kullback-Leibler (K-L) divergence

In terms of encoding the entropy $H(X)$ can be interpreted as the expected coding cost of the distribution P when $-logP(x)$ numbers of bits to encode each symbol x are used. Therefore, the entropy of a random variable X with probability distribution P can be expressed as the expectation value of the negative logarithm of the probability P, that is:

$$H(X) = -E_P[log\ P(x)] \tag{1.24}$$

Suppose now, that the true distribution P of the data is unknown, but we want to learn an approximating distribution Q from data. The optimal code with respect to Q would use $-logQ(x)$ bits for each symbol x. Therefore, the expected number of bits to encode X is given by $E_P\,[log\ Q(x)]$ where expectations are taken with respect to the true distribution. We call this expected value the *cross entropy* and denote it by $H(P, Q)$.

Definition 21. Kullback-Leibler (K-L) divergence (also known as the **relative entropy**)[6] between two distributions P and Q is defined as the difference between the cross entropy $H(P, Q)$ and the entropy $H(X)$

$$KL(P \parallel Q) \equiv H(P, Q) - H(X) = \sum_{x \in \Omega_X} P(x)\ log\ \frac{P(x)}{Q(x)} \tag{1.25}$$

In terms of encoding, *K-L divergence* is the quantity that measures the information loss when the model Q is used to approximate P (the *truth*). It can be shown that $H(P, Q) \geq H(X)$ always, with equality if and only if the two distributions are identical. It follows that $KL(P \parallel Q) \geq 0$, with $KL(P \parallel Q) = 0$ iif $P = Q$. Thus, $KL(P \parallel Q) = 0$ means that there is no information loss when the model Q reflects the truth P perfectly. However, in some real applications, it is more probable that some information will be invariable be lost when a model is used to approximate full reality, thus $KL(P \parallel Q) > 0$. This justifies thinking of K-L divergence as a *pseudo distance* between two distributions. Here the word "*pseudo*" is used because K-L divergence is not symmetric, that is, $KL(P \parallel Q) \neq KL(Q \parallel P)$. Moreover, it does not obey the triangle inequality.

[6] In the literature the terms relative entropy and cross entropy are often used synonymously, to refer to $KL(P \parallel Q)$ and $H(P, Q)$. As argued in [131] the probable reason is that for minimization either can be used.

Derivation of the Akaike's Information Criterion

Consider the model selection task and assume that no hypothesis in the set of candidate models is the true distribution and hence, the goal is the selection of the best approximating hypothesis. Let M^h be a point hypothesis in $\mathcal{M}$ and M_{true} be the unknown true distribution P_{true}. The best point hypothesis based on the *K-L divergence* is the one which losses less information with respect to other candidate hypotheses, that is, the hypothesis $M_{\mathcal{KL}}$ that minimizes the K-L divergence $KL(M_{true} \parallel M^h)$ over the set $\mathcal{M}$ of candidate models. More formally,

$$M_{\mathcal{KL}} \equiv \arg \min_{M^h \in \mathcal{M}} KL(M_{true} \parallel M^h) = \arg \min_{M^h \in \mathcal{M}} H(M_{true}, M^h) - H(M_{true}) \tag{1.26}$$

The *K-L divergence* cannot be used directly as a criterion in model selection because this requires the knowledge of the true distribution M_{true}. Since the second term $H(M_{true})$ in Equation 1.26 depends only on the unknown true distribution, we can treat it as a constant across models. Thus, we only need to estimate the *cross entropy* $H(M_{true}, M^h)$. Let us further define the *relative expected K-L divergence*, denoted by KL_{rel}, as the negative of the cross-entropy, that is, $KL_{rel}(M^h) \equiv -H(M_{true}, M^h)$. Given an observed dataset $\mathcal{D}$, we can use KL_{rel} as a model selection criterion only if we are able to estimate the expectation $E_{P_{true}}[log\ P(D|M, \Theta_M)]$ for each model M. Akaike in [2] found one such estimate based on the *maximized log-likelihood* function. This estimate is the *expected empirical maximized log-likelihood*. Since an unbiased estimator of the expected maximized log-likelihood is simply itself we obtain

$$E_{P_{true}}[log\ P(D|M, \Theta_M)] = log\ P(D|M, \hat{\Theta}_{\mathcal{ML}}) \tag{1.27}$$

From equation 1.12 comes that $KL_{rel}(M, \mathcal{D}) = l(M_{\mathcal{ML}} : \mathcal{D})$. This estimate turns out not to be a suitable criterion because it has two source of errors: *i*) the bias error resulting for the use of the ML estimator; *ii*) the variance error that depends strong on the model's dimension $\parallel M \parallel$. In order to correct for this two sources of errors, Akaike found the following asymptotic bias correction for the estimate of the relative KL divergence:

$$KL_{rel}(M, \mathcal{D}) = l(M_{\mathcal{ML}} : \mathcal{D}) - \parallel M \parallel \tag{1.28}$$

Next, Akaike multiplied this result by -2, and this became *Akaike's information criterion* for model selection.

Definition 22. The **Akaike's Information Criterion** (AIC) of a model M given the training data $\mathcal{D}$ is defined as:

$$Score_{\mathsf{AIC}}(M, \mathcal{D}) \equiv -2\ l(M_{\mathcal{ML}} : \mathcal{D}) + 2 \parallel M \parallel \tag{1.29}$$

Therefore, minimizing AIC is approximately equivalent to minimizing the expected *K-L divergence* between the *true distribution* and the *approximating distribution*. Finally, note that AIC can be also derived from the penalized log-likelihood formula of Equation 1.13 when $f(N) = 1$. By dividing the AIC derived by Akaike from Equation 1.29 by -2, we obtain

$$Score_{\mathsf{AIC}}(M, \mathcal{D}) = Score_{\mathsf{MLC}}(M, \mathcal{D}) - \parallel M \parallel \tag{1.30}$$

1.5.5 Minimum Description Length (MDL)

BIC criterion has an alternative formulation in terms of *information-theoretic* concepts: the *Minimun Descriprion Length* (MDL) principle introduced by Rissanen in [116]. MDL principle attempts to describe the data using a minimum encoding approach: *the more we can compress the data, the more regularity we can detect in the data, the more we can learn about the data.* Thus, the optimal model is the one that compresses the data most.

The material presented in this section is mainly based on the Grunwald's tutorial on the MDL principle [53]. Let us begin with a formal definition of the *code-length* for a code. Suppose we have a countable set $S = \{s_1, s_2, \ldots\ s_m\}$ of symbols.

Definition 23. The **code-length** $CL(s)$ for a symbol $s \in S$ using a code C is defined as the number of bits needed to encode s using the code C.

Let $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ be a set of candidate probabilistic models, each of them containing a set of point hypotheses M^h. Given a training dataset $\mathcal{D}$ of N *i.i.d.* examples of $\mathbf{X}$ sampled from the unknown joint distribution $P(\mathbf{X})$, the best hypothesis based on the MDL principle is the one which minimizes *the total code-length* needed to describe the model and the data using that model. More formally:

$$M_{\mathsf{MDL}} \equiv \arg \min_{M^h \in \mathcal{M}} CL(M^h) + CL(\mathcal{D} \mid M^h) \tag{1.31}$$

where the first term $CL(M^h)$ is the *code-length of the hypothesis* M^h, that is, the number of bits needed to encode M^h and the second term $CL(\mathcal{D} \mid M^h)$ is the *code-length of the data* $\mathcal{D}$ using M^h, that is, the number of bits needed to encode the data when encoded with the help of that hypothesis. The best model to explain $\mathcal{D}$ is the smallest model containing the selected M^h.

The basic principle behind MDL modeling is to find a code that minimizes the code-length over all datasets $\mathcal{D}$ which can be "*well modelled*" by some probabilistic model. Therefore, we need to associate a code with each hypothesis $M^h = P(\mathcal{D} \mid M^h, \Theta_M)$, or more precisely, a *code-length* function over datasets. Since candidate hypotheses are themselves probability distributions over the possible datasets, each in turn defines an optimal code with code-length given by $-\ log\ P(\mathcal{D}|M^h, \Theta_M)$. Hence, the code-length for the second term in Equation 1.31 corresponds to the negative of the *log-likelihood* of the hypothesis M^h given data $\mathcal{D}$

$$CL(\mathcal{D} \mid M^h) = -\ l(M^h : \mathcal{D}) \tag{1.32}$$

Similarly, we need to find a code to compute the code-length $CL(M^h)$ in the first term of Equation 1.31. To this end we need to associate a code for each point hypothesis $M^h \in \mathcal{M}$. In [117] Rissanen proposed to associate a fixed code with each model M instead of encoding each point hypothesis M^h. This fixed code is designed such that whenever there is a hypothesis $M^h \in \mathcal{M}$ that fits the data well, in the sense that $CL(\mathcal{D} \mid M^h)$ is small, then the code-length of this fixed coded will also be small. The hypothesis within the model M for which the code-length $CL(\mathcal{D} \mid M^h)$ is minimal is the ML hypothesis M_{ML}. This code is possible to construct and is called the *stochastic complexity code*.

Definition 24. The **stochastic complexity code** of a dataset D given a model M is the code C_{SC} that minimizes the total code-length, that is,

$$C_{SC} \equiv \arg \min_{c \in \mathcal{C}(\mathcal{D})} -\ l(M_{ML} : \mathcal{D}) + K_N \tag{1.33}$$

Choosing a code such that the constant K_N is as small as possible yields the optimal code. The minimal K_N is called the *parametric complexity* of M, a measure related to the number of degrees of freedom for parameters and also to the geometric structure of the model. We further define the *stochastic complexity of a dataset* as follows:

Definition 25. The **stochastic complexity** of a dataset $\mathcal{D}$ given a model M, denoted by $SC(\mathcal{D} : M)$, is the shortest code-length obtained when the encode is done with the help of M, that is, this is the code-length resulting when $\mathcal{D}$ is encoded using the stochastic complexity code C_{SC}.

Further by setting $K_N = \frac{1}{2}\ log\ N \parallel M \parallel$ where $\frac{1}{2}\ log\ N$ bits are used to represent each parameter in M, we arrive to the definition of the MDL score as the stochastic complexity $SC(\mathcal{D} : M)$.

Definition 26. The **Minumum Description Length**(MDL) score of a model M given the training data $\mathcal{D}$ is defined as:

$$Score_{\mathsf{MDL}}(M, \mathcal{D}) \equiv -\ l(M_{\mathcal{ML}} : \mathcal{D}) + \frac{1}{2}\ log\ N \parallel M \parallel \tag{1.34}$$

Model selection by MDL principle is equivalent to inference by Bayesian approaches. The maximization of the *log of the relative posterior probability* implicit in Bayesian methods is equivalent to minimize the *total description length* of model and data. As depicted in Equation 1.16, the log of the relative posterior probability is the sum of two components: the *log prior* and the *log marginal likelihood*. If we interpret both these terms as *code-lengths*, the negative logarithm of the relative posterior probability can be interpreted as the total description length of model and data.

Finally, from a practical point of view we can also derive MDL from the penalized log-likelihood formula in Equation 1.13. By setting $f(N) = \frac{1}{2}logN$ and then multiplying by -1 we obtain

$$Score_{\mathsf{MDL}}(M, \mathcal{D}) = -\ Score_{\mathsf{MLC}}(M, \mathcal{D}) + \frac{1}{2}\ log\ N \parallel M \parallel \tag{1.35}$$

1.5.6 Predictive Model Selection Criteria

An alternative approach to model selection is to choose a model for predictive purposes. Given a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of candidate probabilistic models, and a training dataset $\mathcal{D}$, the model selection task is to choose a model $M \in \mathcal{M}$ so that the predictive joint distribution $P\{X_1, X_2, \ldots, X_n\}$ yields the most accurate predictions. This approach requires the definition of a *loss function* in order to assess the quality of each model in terms of its predictive

performance [78]. A *loss function* is a function that maps each element of a sample space onto a real number representing the *loss* or *regret* associated with the event. A predictive model selection criterion, therefore, is dependent on the loss function used.

One of the loss functions commonly used in model selection is the *logarithmic loss function* on the joint distribution known as *log-loss*. To compute the *log-loss* we assume that the examples in the training dataset $\mathcal{D}$ are observed *sequentially*, that is, one after the other. When the t^{th} actual observation is received, the learner suffers a loss equal to $-log\ P(\mathbf{x}^{(t)} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(t-1)})$ which is based on the previous cases. Assuming *i.i.d.* cases this individual loss becomes simply $-log\ P(\mathbf{x}^{(t)})$. By cumulating over N cases we obtain the log-loss.

Definition 27. The **log-loss** of a hypothesis M^h given a dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ of N *i.i.d.* cases is the total loss, that is,

$$logLoss(M^h, D) \equiv -\ log\ P(\mathcal{D} \mid M^h) = -\sum_{t=1}^{N} log\ P(\mathbf{x}^{(t)} \mid M^h) \tag{1.36}$$

The model selection task for predictive purposes is to choose the model $M \in \mathcal{M}$ containing the point hypothesis M^h that minimizes the *log-loss*. This measure is just the negative *log-likelihood* $-l(M^h : \mathcal{D})$, which means that now we are treating the score as a *penalty*, rather than a *measure of goodness* [33]. Thus for each model M we must estimate the parameters Θ_M from data so that the *log-loss* of the resulting hypothesis M^h is minimal. We are interested in obtain a good estimate of the predictive performance. To this end we could use the ML estimate $\hat{\mathbf{\Theta}}_{\mathcal{ML}}$ as we did previously. However, if we take into account the effect of sampling error, coupled with the bias induced by the fact that the ML estimate is itself chosen so as to optimize the performance, this will lead to an *over-optimistic* estimate of the performance [33]. On the other hand, if we use the entire available data $\mathcal{D}$ to estimate the parameters and estimate the log-loss, the resulting hypothesis M^h will overfit the training data. This problem is more pronounced with models that have a large number of parameters. A much better idea is to split the given data in a way that we can ensure that in each individual loss in Equation 1.36 the case $\mathbf{x}^{(t)}$ that is being predicted does not in any way contribute to the estimate $\hat{\mathbf{\Theta}}_M^{(t)}$ used in that term.

A popular approach for estimating the predictive performance of a model is *hold-out testing*. In this scheme, the available data is split into a *training* set and a *hold-out testing*

set. For each candidate model $M \in \mathcal{M}$, the *training set* is used to learn a hypothesis M^h, that is, to estimate the parameters for that model M. Then, the *test part* is used to estimate the performance of the resulting hypothesis M^h. The model $M \in \mathcal{M}$ with the lowest *hold-out* loss is chosen. If we further are interested in obtaining the "*best hypothesis*", then all the available data is used to estimate the parameters of the best model. However, the simple model selection criterion based on hold-out validation is not advisable when the data set is not large enough. As empirically shown [4], if we use a hold-out score for small datasets there is a bias toward simplicity in the chosen models. The estimates of the *hold-out loss* are obtained using a hypothesis with parameters estimated from a smaller sample than those sample that will be used to estimate the parameters of the final hypothesis. Moreover, the variance also increases because we use a small sample to estimate the error. A popular approach avoiding this problem is cross-validation.

Cross-validation [132] is a well established data re-sampling method that can be used to reduce the bias of the performance estimates. In the k-fold cross validation scheme, the training data is partitioned into k subsets (*folds*) of equal size. Each subset is used in turn as a *test set*, while the union of the remaining k-1 parts are used as *training set*. For each candidate model $M \in \mathcal{M}$ and each fold k the resulting *training set* is used to learn a hypothesis (i.e. to estimate the parameters) and then the corresponding *test set* is used to compute the loss for that hypothesis. The k-fold-CV score is the final estimate of the *expected loss* given by the averaged value over the k *loss values*. An extreme case is the *leave-one-out cross-validation* when $k = N$ and each test set contains a single example.

An alternative approach to estimate the predictive performance is the Dawid's *prequential* approach [35] where candidate models are compared by measuring their cumulative loss. This approach is equally applicable when the cases are not modeled as *i.i.d.* [33]. The rationale is to compute the performance estimate *predictively* and *sequentially*, so why it is called "*prequential*". For each candidate model $M \in \mathcal{M}$ the Preq score is computed through a *sequential updating* of the predictive distribution. This approach corresponds to an on-line learning paradigm. We assume that the examples in the training dataset $\mathcal{D}$ are observed *sequentially*. At each time point t the actual example $\mathbf{x}^{(t)}$ is evaluated using the hypothesis $M^{h^{(t-1)}}$ with the parameter estimate $\hat{\mathbf{\Theta}}_M^{(t-1)}$ induced from the first $t-1$ examples. Then, the actual example $\mathbf{x}^{(t)}$ is used to update the actual hypothesis $M^{h^{(t)}}$. As a result, the Preq score

is the resulting *cumulative loss*.

Both *cross validation* and *prequential* approaches for model selection are easy to implement and are computationally feasible for datasets with moderate size. However, both methods can be computationally very expensive when applied for large datasets. Hence methods to reduce the computational cost are desirable. On the other hand, *cross-validation* depends on the way the data is partitioned into the k folds [78]. We can always improve an estimate obtained by cross-validation, for instance, by repeating the algorithm over several partitioning and then by averaging the obtained results. The *prequential* approach, instead, is sensitive to the ordering the data is processed with. Various methods for avoiding the effect of examples' ordering in model selection have been addressed, for instance, in [79].

Under suitable smoothness conditions, *cross-validation* based on the *log-likelihood* will be asymptotically equivalent to AIC, and the *prequential log-likelihood* to BIC [33]. On the other hand, maximizing the *log marginal likelihood* leads to choosing the model minimizing the *prequential log-loss*. Thus, the Bayesian score defined as the log marginal likelihood is a special case of the prequential approach for model selection [78]. Moreover, the *prequential approach* can be also regarded as a *predictive coding system* [116]. Finally, the *frequentist*, *information-theoretic* and *Bayesian* approaches to model selection are closed linked to a specific loss (the *log-loss* related to the log-likelihood). *Cross-validation* and *prequential* approaches have the advantage of being easily modified for different loss functions [78].

1.6 Concluding Remarks

In this chapter we have presented some basic concepts and approaches to statistical inference that are involved in the problem of learning probabilistic models from data. The main purpose was to give a theoretical background in order to contrast the fundamental philosophies underlying the derivation of different estimators and model selection criteria. A good understanding of the philosophies adopted under the different approaches to model selection in the general problem of learning probabilistic models from data is essential for a good understanding and objective evaluation of the existing approaches to the particular problem of learning BNCs. In the next chapter we will introduce the Bayesian networks and will formulate the task of learning their structures as a model selection problem.

Chapter 2

Learning Bayesian Networks

2.1 Introduction

Early work on inductive inference in Artificial Intelligence was centered in symbolic manipulation and logical representations until Pearl in 1988 [108] directed the attention to *probabilistic graphical models*, and in particular, to *Bayesian networks*. Probabilistic graphical models provide a compact representation of joint probability distributions. They are graphs in which nodes represent random variables, and the (lack of) arcs represent conditional independence assumptions. Hence, probabilistic graphical models combine *graph theory* and *probability theory*. The graph part provides a data structure by which the human experts can easily model and interpret the inter-relationships among the variables. This graph structure provides the notion of modularity by combining simple parts of a complex system. Instead, the probability theory ensures that the system as a whole is consistent.

As pointed out in the Preface of the Jordan's book [65]:

> "*Graphical Models in general and Bayesian networks in particular provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering:* **uncertainty** *and* **complexity** *and in particular they play an increasingly important role in the design and analysis of machine learning algorithms.*"

In this chapter we introduce the framework of Bayesian networks and the problem of

learning Bayesian networks from data. Next, we focus on score-based approaches and pose the structure learning problem as a search problem. We then briefly present the derivation of the different estimators and model selection criteria for the particular framework of Bayesian networks that we have just reviewed in Chapter 1. Finally, we introduce heuristic search methods and the hill-climbing algorithm for learning the Bayesian network structure upon which we relied to implement our adaptive algorithms for learning k-DBCs.

For further reading in learning Bayesian networks there is a great amount of literature. A classical tutorial [58] introduces the Bayesian methods for learning the parameters and the structure of Bayesian networks. The recent book [103] gives a fine, mathematically precise overview of the subject, and provides an in-depth understanding of both the underlying foundations and the learning algorithms presented. The chapters 9, 10 and 11 in the book [33] also review various issues related to parameter and structure learning of Bayesian networks. The paper [123] surveys the classical algorithms for learning the structure of Bayesian networks. The book [65] presents a collection of papers discussing recent advances.

2.2 Definition of Bayesian Networks

Bayesian networks graphically represent the joint probability distribution of a set of random variables in a problem domain (e.g. cancer diagnosis). A Bayesian network is composed of a *qualitative* part (its structure) and a *quantitative* part (its parameters). Random variables are represented as *nodes* in the graphical structure, and the dependencies between these variables are represented by *directed arcs*. A directed arc can also be used to represent *causal dependencies* (e.g. the dependence between a *disease* and a *symptom*) as illustrated in the Bayesian network for *lug cancer diagnosis* in Figure 2.1[1]. The uncertainty in the domain is represented by conditional probabilities that express our beliefs about the strengths of the direct dependencies between variables. For instance, Table 2.1 gives the conditional probabilities of each possible value of the variable `Dyspnea`, given each possible combination of values of its parent nodes `Has bronchitis?` and `Tuberculosis or Cancer?`.

As pointed out in [101], despite the name, Bayesian networks do not necessarily imply

[1]This Bayesian network first appeared in [88] and was taken from the Hugin software [104] available on-line at http://www.hugin.com.

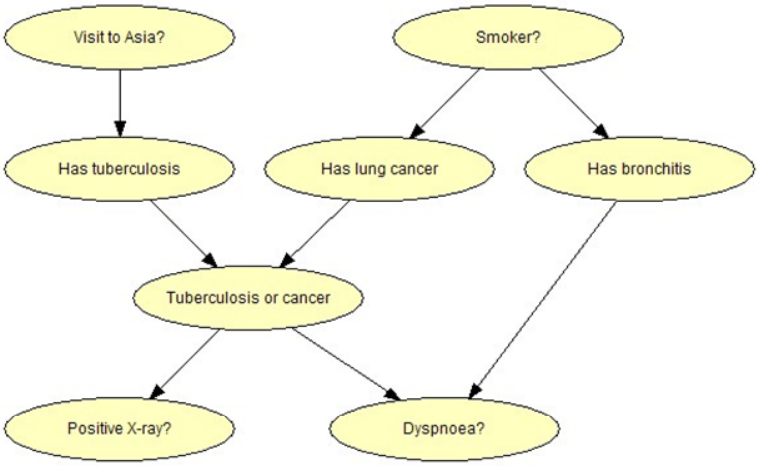

Figure 2.1: The `Asia` Bayesian network for cancer diagnosis

Table 2.1: The CPT for the variable `Dyspnea` in the Bayesian network for cancer diagnosis

Parent 1	Parent 2	**P(`Dyspnea?`| Parents)**	
`Has bronchitis?`	`Tuberculosis or Cancer?`	`yes`	`no`
`yes`	`yes`	0.9	0.1
`yes`	`no`	0.8	0.2
`no`	`yes`	0.7	0.3
`no`	`yes`	0.1	0.9

a commitment to Bayesian statistics. Indeed, it is common to use frequentist approaches to estimate their parameters. Rather, they are so called because they use Bayes' theorem for probabilistic inference. Inference in Bayesian networks means computing any desired posterior conditional probability of some variables given any combination of evidence (observations) on other variables[2]. For example, given the evidence that the patient is a *smoker* and he *has visited Asia*, we can compute the posterior probability that the patient *has lung cancer*.

Let us now more formally define a Bayesian Network. Let $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ be a set of random variables for a domain under study.

Definition 28. A **Bayesian Network** over $\mathbf{X}$ is a tuple $BN = (S, \mathbf{\Theta}_S)$ where the first component, the *network structure* $S = (\mathbf{X}, \mathbf{A})$ is a *directed acyclic graph* (`DAG`) whose *nodes*

[2]For an in-depth study of inference in Bayesian networks the reader is referred to Pearl's book [108].

represent the random variables and whose *arcs* represent direct dependencies between variables; and the second component $\mathbf{\Theta}_S = \{\mathbf{\Theta}_1, \mathbf{\Theta}_2, \ldots, \mathbf{\Theta}_n\}$ is the set of *conditional probability functions* where each $\mathbf{\Theta}_i = P(X_i|\mathbf{Pa}_i) \in \mathbf{\Theta}_S$ represents a *conditional probability function* over the values of X_i given the values of its parents $\mathbf{Pa}_i$. 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:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa}_i) \tag{2.1}$$

The factorization 2.1 of the joint probability distribution is fundamental because it allows us to specify the joint distribution more compactly, thus reducing the number of parameters needed to specify the conditional probability functions. In the most general case, the variables can be *continuous* or *discrete*, and the *conditional probability functions* can be represented in a variety of ways. In this thesis we are restricting to the case when $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ represents a set of *discrete* random variables where each variable X_i may take on values from its finite domain Ω_{X_i}. Assuming discrete variables, each $P(X_i|\mathbf{Pa}_i) \in \mathbf{\Theta}_S$ represents a *conditional probability table* (CPT). Each CPT associated to the variable X_i is composed of q_i rows, one for each possible parent configuration $pa_j \in \Omega_{\mathbf{Pa}_i}$. The entries in each row associated to the configuration pa_j represents the probabilities $P(X_i = x_k|\mathbf{Pa}_i = pa_j)$ for each possible value $x_k \in \Omega_{X_i}$.

2.3 The Problem of Learning Bayesian Networks

While the compact and comprehensible representation considerable facilitates knowledge acquisition, eliciting Bayesian networks from experts can be a very expensive and time consuming task mainly due to the need of specify a great number of probabilities. When no expert knowledge is available, techniques for automatically building Bayesian networks from data would be desirable.

The problem of learning Bayesian networks from data can be formally stated as follows. Given a training dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ of *i.i.d.* examples of $\mathbf{X}$ and some prior information ξ (background knowledge), find the Bayesian network $BN = (S, \mathbf{\Theta}_S)$ that best matches $\mathcal{D}$. We can distinguish a variety of learning tasks, depending on whether the structure

is *known* or *unknown*, the data is *complete* or *incomplete*, and there are *hidden variables* or not. The case of known structure and complete data is the easiest. In this case, we only need to learn the CPT's entries. Each case can be placed into the CPT entries corresponding to the values of the parent variables at each node. Moreover, sequential updating of parameters is fairly straighforward: we only need to update the sufficient statistics[3]. In the context of this work we consider the case when the structure is *unknown*, there are *no hidden variables* and the data is *complete*. Under all these assumptions the learning problem comprises two tasks: *i*) learn the network structure S; *ii*) learn the set of parameters $\mathbf{\Theta}_S$. Approaches to learn the network structure can be further classified into two types:

- **constraint-based approaches** (dependency analysis & search) where some kind of conditional independence (CI) test, such as χ^2 test or *mutual information* test are used to locally measure the dependency relationships between the variables. Then, a search algorithm is used to find a model that is consistent with the observed dependencies and independencies [22, 23, 24, 28, 123].

- **score-based approaches** (scoring & search) where some model selection criterion (scoring function) is used to measure the fitness of each possible structure to the observed data. Then, a search algorithm is used to find one (or more) model that optimizes the score in the space of feasible hypotheses [15, 16, 27, 29, 45, 59, 82, 119].

These two approaches have their advantages and disadvantages. Constraint-based approaches are usually asymptotically correct when the probability distribution of data satisfies certain assumptions, but they have several disadvantages. First, the time complexity is very high since CI tests with large condition-sets are computational expensive. Second, they relay on an arbitrary significance level to test for independence. Score-based approaches, on the contrary, are computationally less expensive (they have less time complexity in the worst case, i.e., when the underlying `DAG` is densely connected), but they may not be able to find the optimal solution due to their heuristic nature [24]. We further focus on *score-based* approaches to learn Bayesian Network structures.

[3]Standard techniques to accomplish sequential updating of parameters using a Bayesian approach can be found in [33, 104, 130].

2.4 Learning Bayesian Networks as a Search Problem

Score-based approaches to learn Bayesian networks typically consist of identifying one or more DAG structures that fit a set of observed data well according to some scoring criterion. Once the structure is identified the parameter are estimated from data. The task of selecting the structure (a probabilistic model) is a model selection problem.

Suppose we have a set $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of Bayesian network structures. The model selection problem in the Bayesian network framework can be formally posed as follows. Given a training dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ of N *i.i.d.* examples of $\mathbf{X}$ sampled from the unknown joint distribution $P(\mathbf{X})$ and some prior information ξ (background knowledge), find the structure $S \in \mathcal{S}$ containing the Bayesian network $BN = (S, \boldsymbol{\Theta}_S)$ that best matches $\mathcal{D}$ according to a given scoring function $Score(S, \mathcal{D})$. As stated in Section 1.4, model selection can be exposed as a *discrete optimization problem* where the scoring function is optimized in the space of possible network structures.

Discrete optimization problems [1] are defined by a finite set of solutions, the *solution space* $\Omega = \left\{\omega_1, \omega_2, \ldots, \omega_{|\Omega|}\right\}$, together with an *objective function* $f : \Omega \rightarrow \Re$. The objective function is a quantitative measure of the quality of each solution that assigns a real value to each element in the solution space, that is, $f(\omega) \in \Re, \forall \omega \in \Omega$. The goal when addressing a discrete optimization problem is to find solutions that *minimize/maximize* the objective function. A solution, $\omega^* \in \Omega$, that *minimizes/maximizes* the objective function is a *globally* optimal solution. A procedure to solve discrete optimization problems is essentially a *search algorithm* that explores the space of possible solutions while optimizing the objective function.

Chickering et al. in [26] proved that the optimization problem of finding an optimal Bayesian network structure is NP-hard. The number of possible `DAGs` grows *super-exponentially* with the number of variables, a fact that can be deduced using the recursive Robinson's formula presented in [118]. One way to handle NP-hard discrete optimization problems is to develop *heuristic search algorithms* with the goal of identifying good or near-optimal solutions. Therefore, score-based approaches to learn the structure of a Bayesian Network 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 the next sections we briefly present the main results related to the three major factors that affect

the performance of score-based approaches: the *parameter estimator* to learn the parameters; and the *scoring function* and the *search method* to learn the structure.

2.5 Parameter Estimators

Learning the parameters of a Bayesian Network with known structure, discrete variables and complete data is fairly straightforward. The structure reduces the dimensionality of the parameter space to the point where it is feasible to estimate parameters from data. The main issue is that we can solve separately each estimation problem for each local multinomial model associated to each variable and each possible parent configuration. We further consider the same notational convention presented in [58].

Let consider a domain $\mathbf{X}$ of discrete random variables $\{X_1, X_2, \ldots, X_n\}$ where each variable X_i may take on values from its finite domain $\Omega_{X_i} = \left\{x_i^1, x_i^2, \ldots, x_i^{r_i}\right\}$ where r_i represents the number of possible values of X_i. Define q_i to be the number of possible parent configurations of X_i, $X_i = k$ the event that $X_i = x_i^k$ and $\mathbf{Pa_i} = j$ the event that $\mathbf{Pa}_i = pa_i^j$. Let S^h denote the hypothesis that the joint distribution of $\mathbf{X}$ can be factored according to the structure S, that is, we define S^h to be true if there exists $\boldsymbol{\Theta}_S \in \Omega_{\boldsymbol{\Theta}_S}$ such that $\boldsymbol{\Theta}_S = \{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \ldots, \boldsymbol{\Theta}_n\}$ where

$$P(\mathbf{X} \mid S^h, \boldsymbol{\Theta}_S) = \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa_i}, S^h, \boldsymbol{\Theta}_i) \tag{2.2}$$

Given the hypothesis S^h, the Bayesian Network $BN = (S, \boldsymbol{\Theta}_S)$ defines the factorization 2.2 of the joint probability distribution over $\mathbf{X}$ as a product of *local distribution functions* $P(X_i \mid \mathbf{Pa_i}, S^h, \boldsymbol{\Theta}_i)$ parameterized by the set of parameters $\boldsymbol{\Theta}_i$. Each $\boldsymbol{\Theta}_i \in \boldsymbol{\Theta}_S$ represents the parameters of the CPT associated to the variable X_i which is composed of q_i rows, one row for each possible parent configuration $\mathbf{Pa}_i = j$. Let further define the set of parameters $\boldsymbol{\Theta}_i$ as $\boldsymbol{\Theta}_i = \{\boldsymbol{\Theta}_{i1}, \boldsymbol{\Theta}_{i2}, \ldots, \boldsymbol{\Theta}_{iq_i}\}$, where each $\boldsymbol{\Theta}_{ij}$ represents the parameters of the CPT's row associated to the parent configuration $\mathbf{Pa}_i = j$.

Assumption 1. Multinomial local models: Each local distribution $P(X_i \mid \mathbf{Pa_i}, S^h, \boldsymbol{\Theta}_i)$ associated to the variable X_i belongs to the multinomial family and can be further decompose as a product of local multinomial models $M_{ij} = P(X_i \mid \mathbf{Pa}_i = j, S^h, \boldsymbol{\Theta}_{ij})$, one for each possible parent configuration $\mathbf{Pa}_i = j$. Each $\boldsymbol{\Theta}_{ij} = \{\theta_{ij1}, \theta_{ij2}, \ldots, \theta_{ijr_i}\}$ represents the set of

parameters of M_{ij} where each θ_{ijk} is the probability $P(X_i = k \mid \mathbf{Pa}_i = j)$ for each possible value x_i^k in Ω_{X_i}.

Assumption 2. Complete Data: The training dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ of N *i.i.d.* examples of $\mathbf{X}$ sampled from the unknown joint distribution $P(\mathbf{X})$ is *complete*, that is, there is no missing values.

Given the hypothesis S^h and under Assumption 2 we can obtain the sufficient statistics of a multinomial Bayesian network if we just count the number of times each variable value and each possible parent configuration is observed in $\mathcal{D}$. The set of sufficient statistics of a multinomial Bayesian network is therefore the set of frequency counters $\mathbf{T}(\mathcal{D} \mid S) \equiv \{N_{ijk} \mid i = 1 \ldots n, j = 1 \ldots q_i, k = 1 \ldots r_i\}$, where each N_{ijk} is the number of cases in $\mathcal{D}$ such that $X_i = k$ and $\mathbf{Pa_i} = j$.

2.5.1 ML Estimate

As stated in section 1.3.1 maximum likelihood estimation is to maximize the likelihood function of the overall parameter $\mathbf{\Theta}_S$ given data $\mathcal{D}$. Assuming example independence, the likelihood can be written as a product of N terms. Then using the factorization of the joint distribution given in Equation 2.2 we obtain

$$\begin{aligned}
\mathcal{L}(\mathbf{\Theta}_S : \mathcal{D}, S^h) &= \prod_{l=1}^{N} \prod_{i=1}^{n} P(\mathbf{x}^{(l)} \mid \mathbf{\Theta}_i, S_i) \\
&= \prod_{i=1}^{n} \prod_{l=1}^{N} P(\mathbf{x}^{(l)} \mid \mathbf{\Theta}_i, S_i) \\
&= \prod_{i=1}^{n} \mathcal{L}(\mathbf{\Theta}_i : \mathcal{D}, S_i) \qquad (2.3)
\end{aligned}$$

where S_i represents the local structure defined by the node X_i and its parents $\mathbf{Pa_i}$. Under Assumption 1 of multinomial local models we get further decomposition of the local likelihood:

$$\begin{aligned}
\mathcal{L}(\mathbf{\Theta}_i : \mathcal{D}, S_i) &= \prod_{l=1}^{N} \prod_{j=1}^{q_i} P(\mathbf{x}^{(l)} \mid \mathbf{\Theta}_{ij}, M_{ij}) \\
&= \prod_{j=1}^{q_i} \prod_{l=1}^{N} P(\mathbf{x}^{(l)} \mid \mathbf{\Theta}_{ij}, M_{ij}) \\
&= \prod_{j=1}^{q_i} \mathcal{L}(\mathbf{\Theta}_{ij} : \mathcal{D}, M_{ij}) \qquad (2.4)
\end{aligned}$$

Thus, for each variable X_i and each possible parent configuration pa_i^j we get an independent estimation problem for each local multinomial model M_{ij}. From the properties of the multinomial distribution we further obtain:

$$\mathcal{L}(\mathbf{\Theta}_{ij} : \mathcal{D}, S_i) = \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}} \tag{2.5}$$

Frequency counting is the method used to determine the ML estimator for a multinomial distribution given complete data [41]. Hence the ML estimate of each multinomial parameter $\theta_{ijk} \in \mathbf{\Theta}_{ij}$ is obtained from the observed relative frequencies. More precisely,

$$\hat{\Theta}_{ijk} = \frac{N_{ijk}}{Nij} \tag{2.6}$$

where $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$, $i = 1 \ldots n, j = 1 \ldots q_i, k = 1 \ldots r_i$.

Substituting Equation 2.5 into Equations 2.4 and 2.3 we obtain the likelihood of the Bayesian Network $BN = (S, \Theta_S)$. We have that

$$\mathcal{L}(BN : \mathcal{D}) \equiv P(\mathcal{D} \mid \mathbf{\Theta}_S, S^h) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}} \tag{2.7}$$

2.5.2 Bayesian Estimates

In the Bayesian approach to parameter estimation we express our uncertainty on the parameters by regarding $\mathbf{\Theta}_S$ as a random vector over the parameter space $\Omega_{\mathbf{\Theta}_S}$ and by specifying a prior distribution $P(\mathbf{\Theta}_S \mid S^h)$. After observing the data $\mathcal{D}$, the problem of learning the parameters for the given structure S is that of computing the posterior distribution $P(\mathbf{\Theta}_S \mid \mathcal{D}, S^h)$. As proved, for instance in [29, 58, 59], under some nice assumptions the posterior distribution $P(\mathbf{\Theta}_S \mid \mathcal{D}, S^h)$ can be efficiently computed in a closed-form solution.

Assumption 3. Global Parameter Independence: The global parameters $\mathbf{\Theta}_i$ are mutually independent.

Assumption 4. Local Parameter Independence: The local parameters $\mathbf{\Theta}_{ij}$ are mutually independent.

Under Assumptions 3 and 4 of parameter independence we have

$$P(\mathbf{\Theta}_S \mid S^h) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} P(\mathbf{\Theta}_{ij} \mid S^h) \tag{2.8}$$

Under Assumption 2 of complete data and assumptions of parameter independence, the parameters remain independent given $\mathcal{D}$. Hence the posterior of $\mathbf{\Theta}_S$ is

$$P(\mathbf{\Theta}_S \mid \mathcal{D}, S^h) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} P(\mathbf{\Theta}_{ij} \mid \mathcal{D}, S^h) \tag{2.9}$$

This decomposition is crucial because we can update each parameter $\mathbf{\Theta}_{ij}$ of each local multinomial model independently. Therefore, in the Bayesian approach to parameter estimation we also get an independent estimation problem for each local multinomial model M_{ij}. By treating each $\mathbf{\Theta}_{ij}$ as a random vector, we further assume that $\mathbf{\Theta}_{ij}$ has a prior Dirichlet distribution, which is conjugate to the multinomial distribution.

Definition 29. The **Dirichlet distribution** of each parameter vector $\mathbf{\Theta}_{ij}$ with parameters $\{\alpha_{ij1}, \ldots, \alpha_{ijr_i}\}$ denoted by $Dir(\mathbf{\Theta}_{ij} \mid \alpha_{ij1}, \ldots, \alpha_{ijr_i})$ is defined by

$$P(\mathbf{\Theta}_{ij}) \equiv \frac{\Gamma(\alpha_{ij})}{\prod_{k=1}^{r_i} \Gamma(\alpha_{ijk})} \prod_{k=1}^{r_i} \theta_{ijk}^{\alpha_{ijk}-1}$$

where $\alpha_{ijk} \in \mathrm{Z}, \alpha_{ijk} > 0$, $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$ and $\Gamma(.)$ is the gamma-function, which satisfies $\Gamma(x+1) = x\Gamma(x), \Gamma(1) = 1$ (related to the factorial by $\Gamma(n+1) = n!$).

The conjugate Dirichlet prior perfectly captures the results of past data and allows to express our prior beliefs in terms of some "*imaginary*" data. Usually the parameters α_{ijk} of the conjugate Dirichlet are called *hyper-parameters* in order to differentiate them from the parameters θ_{ijk} of the multinomial distribution. Thus, the hyper-parameters α_{ijk} can be though of as "*imaginary*" counts from our past experience.

Assumption 5. Dirichlet Priors: Each vector $\mathbf{\Theta}_{ij} \in \mathbf{\Theta}_S$ has Dirichlet prior $P(\mathbf{\Theta}_{ij} \mid S^h)$ with hyper-parameters $\alpha_{ijk} > 0$, that is, $P(\mathbf{\Theta}_{ij} \mid S^h) = Dir\left(\mathbf{\Theta}_{ij} \mid \alpha_{ij1}, \ldots, \alpha_{ijr_i}\right)$.

Since Dirichlet distribution is the conjugate prior to the multinomial distribution, the posterior distribution remains in the Dirichlet family and it can be efficiently computed in a closed-form:

$$P(\mathbf{\Theta}_{ij} \mid \mathcal{D}, S^h) = Dir(\mathbf{\Theta}_{ij} \mid \alpha_{ij1} + N_{ij1}, \ldots, \alpha_{ijr_i} + N_{ijr_i}) \tag{2.10}$$

where N_{ijk}, as defined, is the number of cases in $\mathcal{D}$ such that $X_i = k$ and $\mathbf{Pa}_i = j$.

From the obtained posterior distribution in Equation 2.10 and the properties of the Dirichlet distribution we can derive both the Bayesian and the MAP estimate of each multinomial

parameters θ_{ijk} for every value of i, j and k in a closed-form solution. The full derivation of the Bayesian estimates under all the above assumptions, is given, for instance, in [29, 58, 59, 103]. Here we only depict the main results. As defined in Section 1.3.2 the Bayesian estimate can be obtained by averaging (integrating) the probabilities over the parameter space weighting their results by the respective posterior probabilities. This yields to the computation of the posterior expectation as follows:

$$\hat{\theta}_{ijk} = E_{P(\mathbf{\Theta}_{ij}|\mathcal{D},S^h)}(\theta_{ijk}) = \int \theta_{ijk} P(\mathbf{\Theta}_{ij} \mid \mathcal{D}, S^h) d\mathbf{\Theta}_{ij} \tag{2.11}$$

Then we can use the definition of the Dirichlet distribution to solve these expectations. As a result, we obtain the Bayesian estimate of each multinomial parameter $\theta_{ijk} \in \mathbf{\Theta}_{ij}$ for $i = 1 \ldots n, j = 1 \ldots q_i, k = 1 \ldots r_i$:

$$\hat{\theta}_{ijk} = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}} \tag{2.12}$$

where $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$.

Other alternative is to use the MAP estimate of $\mathbf{\Theta}_{ij}$, that is, the value that maximizes the posterior distribution

$$\hat{\mathbf{\Theta}}_{ij} = \arg\max_{\mathbf{\Theta}_{ij}} P(\mathbf{\Theta}_{ij} \mid \mathcal{D}, S^h) \tag{2.13}$$

By solving this optimization problem, we can obtain the MAP estimate[4] of each multinomial parameter $\theta_{ijk} \in \mathbf{\Theta}_{ij}$ for $i = 1 \ldots n, j = 1 \ldots q_i, k = 1 \ldots r_i$ as follows:

$$\hat{\theta}_{ijk} = \frac{\alpha_{ijk} + N_{ijk} - 1}{\alpha_{ij} + N_{ij} - r_i} \tag{2.14}$$

where $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$.

As stated in Section 1.3.2, the MAP estimate is the Bayesian counterpart to the ML estimate and they become equivalent if we assume uniform priors. Note that when in Equation 2.14 all the hyper-parameters α_{ijk} are set to 1 we get the ML estimate defined in equation 2.6. Moreover, as argued in [74] if the data was actually generated from the given network structure, then both ML and Bayesian estimates converge asymptotically to the correct parameter setting. If not, then they converge to the distribution with the given structure which is "closest" to the distribution from which the data was generated. A crucial advantage of

[4]The derivation of the MAP estimate of a binomial parameter, is given, for instance, in [114]. Generalizing these results we can obtain the MAP estimates for the multinomial distribution.

both estimation methods is that both can be implemented on-line by accumulating sufficient statistics.

Although the ML estimator has been the most commonly used estimator for Bayesian networks, Bayesian estimates tend to be more robust and, furthermore, generally, less sensitive to the presence of zeroes in frequency counts [31]. Note that in many domains and for real datasets some frequency counts can remain zero even though the underlying parameters are not. Hence, it is desirable to bias the estimates away from zero. A commonly applied technique is to smooth the estimates a little thus avoiding 0 values. Usually, frequency counters are initialized to some small value c. When $c = 1$ the ML estimates are equivalent to the MAP estimates with the *add-one prior*, which in turns is equivalent to the Bayesian estimates with uniform priors [128]. More details about these issues with parameter estimates for Bayesian networks are given in [4, 31, 128].

2.6 Scoring Functions

Given a training dataset $\mathcal{D}$ and a set $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of possible Bayesian network structures, the model selection problem consists of selecting the structure $S \in \mathcal{S}$ containing the Bayesian network $BN = (S, \mathbf{\Theta}_S)$ that "*best fits*" $\mathcal{D}$ according to a given scoring function $Score(S, \mathcal{D})$. We further derive the formulae for the scores that we used in our experiments. From a practical point of view, not philosophical, we can classify them into four categories:

1. Log-likelihood-based scores: MLC.

2. Penalized log-likelihood scores: MDL/BIC[5], AIC.

3. Bayesian scores: BD, BDeu.

4. Predictive scores: k-Fold-CV, Preq.

[5]Note that MDL leads to the same criterion as BIC differing only by a minus sign, so here we only derive the BIC score, although in our experiments we will use the MDL score.

2.6.1 The Maximum Likelihood Criterion MLC

As defined in section 1.5.1, the MLC is the maximized *log-likelihood* of a Bayesian network given the training data. From the likelihood of a Bayesian Network given in Equation 2.7 we can obtain the *log-likelihood* as follows:

Proposition 2.6.1. *The* **log-likelihood** *of a Bayesian Network $BN = (S, \Theta_S)$ is given by*

$$l(BN : \mathcal{D}) \equiv l(\Theta_S : \mathcal{D}, S) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \theta_{ijk} \tag{2.15}$$

Substituting the parameters θ_{ijk} to the ML estimates $\hat{\theta}_{ijk}$ given in Equation 2.6 into Equation 2.15 we obtain the MLC for Bayesian networks:

$$Score_{\mathsf{MLC}}(S, \mathcal{D}) \equiv l(\hat{\Theta}_{ML} : \mathcal{D}, S) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} \tag{2.16}$$

2.6.2 Penalized Likelihood Scores: BIC/MDL and AIC

We can derive AIC and BIC using the general formula 1.13 of penalized log-likelihood scores defined in Section 1.5.1. The formulas for BIC and AIC are given by:

$$Score_{\mathsf{AIC}}(S, \mathcal{D}) \equiv Score_{\mathsf{MLC}}(S, \mathcal{D}) - \parallel S \parallel \tag{2.17}$$

$$Score_{\mathsf{BIC}}(S, \mathcal{D}) \equiv Score_{\mathsf{MLC}}(S, \mathcal{D}) - \frac{1}{2}\, logN \parallel S \parallel \tag{2.18}$$

where $\parallel S \parallel$ is the dimension of the network structure defined as the number of its parameters:

$$\parallel S \parallel \equiv \sum_{i=1}^{n} q_i (r_i - 1) \tag{2.19}$$

2.6.3 Bayesian Scores: BD and BDeu

To derive the Bayesian score for Bayesian networks we follow the same steps for the derivation of the Bayesian score described in Section 1.5.2. We consider a finite set $\mathcal{S}$ of possible structure hypotheses. Each $S^h \in \mathcal{S}$ denotes the hypothesis that the joint distribution of $\mathbf{X}$

can be factored according to the structure S. We express our uncertainty on the structure by defining a random discrete variable $\mathcal{S}$ whose states correspond to the possible hypotheses S^h. The next step is to place a prior distribution $P(S^h)$ for each candidate hypothesis. After observing the data $\mathcal{D}$, the prior distribution is combined with the *marginal likelihood* $P(\mathcal{D} \mid S^h)$ by means of the Bayes' theorem in order to obtain the posterior probability of each structure hypothesis S^h. As a result, the Bayesian score is based on the *log of the relative posterior probability* of the structure S^h given the dataset $\mathcal{D}$. The Bayesian score for Bayesian networks is usually called Bayesian Dirichlet, BD for short:

$$Score_{\mathsf{BD}}(S^h, \mathcal{D}) \equiv log\ P(S^h) + log\ P(\mathcal{D} \mid S^h) \tag{2.20}$$

To obtain the Bayesian score we need to asses the prior distribution $P(S^h)$ for each candidate structure and to compute the marginal likelihood $P(\mathcal{D} \mid S^h)$. Cooper et al. in [29] and Heckerman et al. in [59] proved that under some nice assumptions the marginal likelihood $P(\mathcal{D} \mid S^h)$ can be derived in a closed-form solution and it decomposes into a product of terms, one for each local multinomial model. The basic results are summarized in Theorem 2.6.2. Before enunciating this theorem, let us introduce a last assumption:

Assumption 6. Parameter Modularity: If a node X_i has the same parents in two different network structures S_1 and S_2 then the local multinomial distributions associated with this node are identical, i.e.,

$$M_{ij}^1 \equiv P(X_i \mid \mathbf{Pa}_i = j, S_1, \mathbf{\Theta}_{ij}^1) = M_{ij}^2 \equiv P(X_i \mid \mathbf{Pa}_i = j, S_2, \mathbf{\Theta}_{ij}^2)$$

Theorem 2.6.2. *Let* $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ *be a set of* n discrete random variables*, where each variable* X_i *has* r_i *possible values. Let* $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$ *be a dataset of* N *i.i.d. examples of* $\mathbf{X}$*. Under assumption 1 of multinomial local distributions; assumption 2 of complete data, assumptions 3 and 4 of parameter independence; assumption 5 of Dirichlet prior for each local parameter* $\mathbf{\Theta}_{ij}$ *and assumption 6 of parameter modularity it follows that the marginal likelihood of a Bayesian Network given data,* $P(\mathcal{D} \mid S^h)$*, can be derived in a closed-form and it decomposes into a product of terms as follows:*

$$P(\mathcal{D} \mid S^h) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ijk})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})} \tag{2.21}$$

where $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$, $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ *and* $\Gamma(.)$ *is the gamma-function.*

It is evident from Equations 2.20 and 2.21 that in order to compute the BD score of a network structure S we need to asses:

1. The structure prior $P(S^h)$.

2. The parameter priors $P(\boldsymbol{\Theta}_S \mid S^h)$, that is, all the hyper-parameters α_{ijk} for every value of i, j, and k.

When the number of possible structure is large, the assessment of the prior for each structure is intractable. The simplest and therefore most common solution is to ignore the prior component in 2.20 assuming all the candidate structures to be equally probably a priori. This leads to the *log marginal likelihood*, one of the most commonly used scores to learn Bayesian networks. On the other hand, when the structure is complex, in the sense that the number of parameters is large, the assessment of all the hyper-parameters α_{ijk} is also intractable. For avoiding prior assessments some special cases of the BD score were derived by using *non-informative* priors for the parameters.

For different prior assessments we can derive different cases of the BD score. One special case is the K2 score derived by Cooper and Herskovits in [29]. K2 uses the *log marginal likelihood* with the simple *non-informative* assignment $\alpha_{ijk} = 1$. Other special case is the BDeu score. BDeu uses the assignment $\alpha_{ijk} = \frac{1}{(r_i q_i)}$ suggested by Buntine in [15]. The name BDeu for the Buntine's assignment was later established by Heckerman et al. in [59] as a special case of the BDe score, which corresponds to BD with the additional assumption of *likelihood equivalence*. This assumption says that for any dataset $\mathcal{D}$, the likelihood of two structure hypotheses corresponding to any two *equivalent* network structures is the same. As proved in [59] the Buntine's assignment satisfies the property of *likelihood equivalence*. Moreover, BDeu also satisfies the property of *uniform joint distribution*, which means that every instance of the joint space is equally probable given S^h. Therefore in the BDeu score, "**e**" means *likelihood equivalence* and "**u**" *uniform joint distribution*.

The two most commonly used scores to learn Bayesian networks are the Bayesian score (usually the marginal likelihood) [15, 16, 27, 29, 59] and the MDL score [43, 44, 81, 133, 134]. These scoring functions are asymptotically equivalent as the sample size increases. Moreover,

they are both asymptotically correct, that is, with probability equal to one the learned distribution converges to the underlying distribution as sample size increases [10, 58].

2.6.4 Predictive Scores: k-Fold-CV and Prequential

Given a set $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of candidate Bayesian network structures the model selection task is to choose a structure $S \in \mathcal{S}$ so that the predictive joint distribution yields the most accurate predictions. Predictive model selection criteria, such as *cross-validation* or *prequential* scores, can be used to learn a Bayesian Network for predictive purposes. As stated in Section 1.5.6 both approaches require a *loss function* for measuring the predictive accuracy. The *log-loss* given in Equation 1.36 is the loss function most commonly used in learning Bayesian networks. However, when a Bayesian Network is used in classification tasks, the *zero-one loss* is usually employed.

Algorithm 1 The algorithm for computing the k-fold cross-validation score for Bayesian networks

Require: A Bayesian network structure S, a dataset $\mathcal{D}$ of *i.i.d.* examples of $\mathbf{X}$, a loss function lossF(BN,$\mathcal{D}$), the number of folds k

Ensure: The k-Fold-CV score for the structure S given the data $\mathcal{D}$

1: Split the dataset $\mathcal{D}$ in k folds
2: $\Theta_S \Leftarrow$ initialize-CPTs(S)
3: BN $\Leftarrow (S, \Theta_S)$
4: **for** each fold in $\mathcal{D}$ **do**
5: $\quad \mathcal{D}_{\texttt{training}} \Leftarrow \mathcal{D} \backslash$ fold {first: training}
6: $\quad$ learnParameters($\Theta_S, \mathcal{D}_{\texttt{training}}$)
7: $\quad \mathcal{D}_{\texttt{test}} \Leftarrow$ fold {second: testing}
8: $\quad$ loss[fold] $\Leftarrow$ lossF(BN, $\mathcal{D}_{\texttt{test}}$)
9: **end for**
10: **return** Average(loss[fold]) {the k-Fold-CV score for S given data $\mathcal{D}$}

The *cross-validation* score for a candidate structure $S \in \mathcal{S}$ is computed by splitting the given dataset $\mathcal{D}$ into k subsets. Each subset is used in turn as a validation set, while the union of the remaining k-1 sets are used as training set for parameter estimation. The k-Fold-CV score is then the average over the k loss values. Algorithm 1 is the algorithm for the computation of the k-Fold-CV score for Bayesian networks. The extreme case of the algorithm is the *leave-one-out cross-validation* when $k = N$ and each subset containing a single example. We call this score LOO-CV.

The *prequential score* for a candidate structure $S \in \mathcal{S}$ is based on the Dawid's prequential approach [35]: for each example the current model is used to do prediction and a individual loss is returned. Then this example is used to update the parameters. The prequential score is the resulting *cumulative loss*. Algorithm 2 is the base algorithm for computing the prequential score for Bayesian networks.

Algorithm 2 The algorithm for computing the prequential score for Bayesian networks

Require: A Bayesian network structure S, a dataset $\mathcal{D}$ of *i.i.d.* examples of $\mathbf{X}$, a loss function lossF(BN,$\mathcal{D}$))

Ensure: The *prequential* score Preq for the structure S given the data $\mathcal{D}$

1: $\Theta_S \Leftarrow$ initialize-CPTs(S)
2: BN $\Leftarrow (S, \Theta_S)$
3: **for** each example $\mathbf{x}$ in $\mathcal{D}$ **do**
4: `cumLoss`+ =lossF(BN, $\mathbf{x}$) {first: predict}
5: update(Θ_S, $\mathbf{x}$) {second: update the parameters with new example}
6: **end for**
7: **return** `cumLoss` {the prequential score for S given data $\mathcal{D}$}

Whereas the results with cross-validation depend on the way the data is partitioned into the k folds, the main problem with the prequential score is the fact that the criterion is sensitive to the ordering the data is processed with. In [79] various methods for avoiding the effect of examples' ordering in model selection have been addressed. The results suggest that averaging over random ordering may be a more sensible strategy for solving the ordering problem than trying to find the ordering the optimizes the prequential score. Both k-Fold-CV and Preq scores are easy to implement, however they are computationally more expensive than those scores computed by means of a closed formula such as MLC, MDL/BIC, AIC, BD and BDeu. Preq score, particularly, requires learning the parameters from increasingly large parts of the data and, as usual, learning the final parameters from all the data.

2.7 Heuristic Search Algorithms

A search algorithm, broadly speaking, is an algorithm that takes a problem as input and returns a solution to the problem, usually after evaluating a number of possible solutions [144]. The set of all possible solutions to the problem is called the *solution space*. The search space consists of a *set of states* that represent the set of possible solutions and a *set of operators* used by the search algorithm to transform one state to another. *Brute-force* or

blind search algorithms use an exhaustive search through the search space to reach a *goal state*. Blind search is *uninformed* search because it does not use domain knowledge to move from the current state to the goal [120].

In principle, *blind search* algorithms can be applied to any problem. However, many real-world problems have very strong constrains in *computing time* and *memory space*, and hence, a brute-force approach becomes impractical. We can improve the search process if we use some problem-specific knowledge to reduce search costs. Unlike brute-force methods, *heuristic*[6] search incorporates domain knowledge to reduce the amount of time spent searching. Heuristic search algorithms are designed with the goal of traversing the solution space in searching for optimal/near optimal solutions. The rationale is that of exploring the state in the search space that is most likely to be nearest to the goal state at each search step. Usually the problem-specific knowledge to make this exploration is provided by the *objective function* which measures the quality of each solution.

In sum, to implement a heuristic search algorithm we need to define the following elements: *i*) the *search space* composed by the *solution space* and the set of *operators*; *ii*) the *initial solution* in the solution space; *iii*) the *search strategy*; *iv*) the *objective function*; *v*) the *goal state*. Usually the goal state is given by means of a *stopping criterion* (e.g. stop when the new solution cannot improve the current solution).

The *search strategy* defines how to organize the search in the search space. Search strategies can be categorized into two types [7]: *deterministics* and *non-deterministics*. Among deterministic strategies there are *hill-climbing* often called *greedy search*, *repeated hill climbing*, *tabu search*, *branch-and-bound*, etc. All these algorithms are deterministic in the sense that all the runs always obtain the same solution. They, as a rule, tend to get stuck in local maximums. In order to make an effort for escaping from local maximum, *non-deterministic* heuristics use *randomness* so that different solutions can be obtained from different runs. Examples of non-deterministic search algorithms are *generalized hill-climbing* (GHC) algorithms [137] such as *simulated annealing*, *threshold accepting*, *Monte Carlo search*, and so on.

[6]The word *heuristic* is derived from the Greek verb *heuriskein*, meaning "to find" or "to search". In the area of search algorithms, it refers to a function that provides an estimate of the solution [120].

2.7.1 Heuristic Search in Learning Bayesian Networks

As stated, the *score-based* approach to learn a Bayesian network structure can be posed as a discrete optimization problem where some scoring function is maximized in the space of possible structures. To solve this problem using a heuristic search algorithm we first need to decide what goes into the composition of the *search space*, that is, the set of its *states* and its *operators*. In the simplest formulation of the search space, the states can be defined to be *individual* `DAGs` and the operators to be *local modifications* to those `DAGs`. Usually these operators are defined as follows: for any pair of nodes X and Y, if X and Y are non-adjacent we can add an arc in any direction. Otherwise the arc connecting them can be either *deleted* or *reversed*. Moreover, all operators are subject to the constraint that a cycle cannot be formed. We call these operators `addArc`, `deleteArc` and `reverseArc`, respectively. Chickering in [27] called the search space defined in this way the `B-space`. Figure 2.2 shows an example of the three operators of the `B-space`.

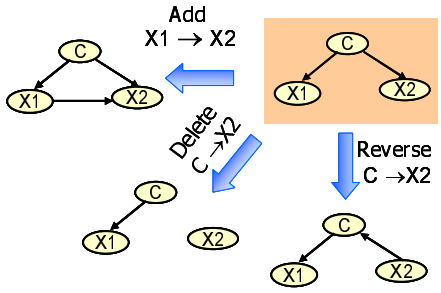

Figure 2.2: Operators `addArc`, `deleteArc` and `reverseArc` in the `B-space`

We can choose the initial solution in the `B-space` to be a `DAG` with no edges and iteratively add arcs that most increase the score subject to never introducing a cycle. On the contrary, we can start with a complete `DAG`, and then iteratively delete the arcs that most increase the score. We can also select an initial solution somewhere in the middle of the search space.

A more sophisticated representation of the search space was proposed by Chickering in [27]. Instead of using individual `DAGs` he defined the states to be *equivalence classes* of Bayesian network structures. Two Bayesian-network structures are *equivalent* if the set of distributions that they represent are identical. The scores mostly used in learning Bayesian networks do not distinguish among equivalent networks. It makes sense therefore to search

among equivalence classes of network structures as opposed to the simplest approach of searching among individual DAGs.

Heuristic search methods can be more efficient if the scoring function is *decomposable*.

Definition 30. A score for a $BN = (S, \mathbf{\Theta}_S)$ is **decomposable** if it can be written as a sum of local contributions, each of which is a function only of one node and its parents

$$Score(S, \mathcal{D}) = \sum_{i=1}^{n} Score_{local}(X_i \mid Pa(X_i), N_{X_i|Pa(X_i)}) \tag{2.22}$$

where for each node X_i, $Score_{local}$ is a local function that only depends on the family of X_i (the node itself and its parents) and $N_{X_i|\mathbf{Pai}}$ denotes the subset of sufficient statistics corresponding to the family of X_i. [27].

Following the distinction made by Bouckaert in [11], we can further distinguish two types of approaches depending of whether the score is *decomposable* or not: *local* score-based approaches and *global* score-based approaches. Local score-based approaches explore the *decomposability* property of scores and decompose the global optimization problem into local optimization problems. This decomposition allows the implementation of local search methods. This means that the change in score that results from the application of an operator can be computed locally. Only those terms in the sum in Equation 2.22 that correspond to nodes whose parents have changed need to be re-computed, and hence, we only need to re-examine the sufficient statistics of the families in consideration. It is evident that MLC, AIC, BIC/MDL, BD and BDeu scores are all decomposable, since they are based on the *log likelihood* (Equation. 2.15) or *log marginal likelihood* (Equation 2.15) and both decompose into a sum of terms, one for each node. Global score-based approaches, on the contrary, use scores which cannot be decomposed into local scores for individual nodes. So, the whole network needs to be considered in order to determine the score. Search-based algorithms using predictive scores such as k-Fold-CV or Preq are global score-based approaches. Several local and global score-based algorithms to learn Bayesian network classifiers using different search algorithms, scoring functions and parameter estimators are implemented in Weka [145, 11].

2.7.2 The Hill-climbing Algorithm for Learning Bayesian Networks

Due to its obvious simplicity for computational implementation, *hill-climbing*[7] is one of the most used search algorithms in learning Bayesian networks [15, 16, 27, 29, 45, 59, 82, 119]. The search starts from an initial solution. Then the solution is constructed *iteratively* by making local changes in the current solution by means of the defined operators. The algorithm always moves to a neighbor solution in the direction of increasing quality. At each search step the local changes that gives a maximum improvement of the objective function are selected. The algorithm ends when there is no more improvement of the score or when there is no possible to build a new solution.

To implement a hill-climbing search algorithm for Bayesian networks we must define the following elements:

- **the search space**: we consider the B-space=$(\mathcal{S}, \mathcal{O})$ where $\mathcal{S}$ is the space of possible DAGs, and $\mathcal{O} \equiv \{\texttt{addArc}, \texttt{deleteArc}, \texttt{reverseArc}\}$ is the set of local operators.
- **the initial solution**: a Bayesian network structure $S_0 \in \mathcal{S}$.
- **the objective function**: one of the scoring functions $Score(S, \mathcal{D})$ defined in Section 2.6 that measures the quality of a given structure S.
- **the stopping criterion**: the algorithm stops when there is no more improvement of the score or when there is no possible to apply a new operator.

Algorithm 3 is the hill-climbing algorithm to learn the structure of Bayesian networks. The algorithm takes as input the search space B-space=$(\mathcal{S}, \mathcal{O})$, an initial structure S, a given dataset $\mathcal{D}$ of *i.i.d.* examples of a set $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ of random variables for a domain under study and a scoring function $Score(S, \mathcal{D})$. At each search step, it applies the operator that results in the maximal gain in the score. This process will continue until the stopping criterion is reached. As a result, a Bayesian network structure of high quality is returned.

Cooper and Herskovits [29] were the first in implementing a hill-climbing algorithm (they called K2) for learning an *unrestricted* structure of a Bayesian Network using the K2 score.

[7] The alternative of hill-climbing is the *gradient descent*, if we view the evaluation function as a cost rather than a quality.

Given a variable ordering, the algorithm begins with an empty network and then it iteratively adds arcs that result in the maximal improvements in the K2 score until there is no more improvement for that score or until it is no possible to add a new arc. A number of improvements of their approach have since been proposed that also rely in a hill-climbing procedure. For instance, Buntine in [15] proposed a hill-climbing algorithm that does not require variable ordering. Later, Singh and Voltara in [129] proposed an extension to the K2 algorithm which they called CB. The CB algorithm uses conditional independence tests to generate a "good" variable ordering from the data. Then, it uses the generated variable ordering, but unlike the k2, the CB algorithm begins with a complete DAG and then uses arc deletions in the process. Heckerman et al. in [59] provide a discussion and evaluation of the hill-climbing approach for learning the structure of Bayesian networks. A recent in-depth study of the hill-climbing algorithm for learning the structure of Bayesian networks along with a novel approach to adapt it for incremental learning is presented in [119].

Algorithm 3 The hill-climbing search algorithm for learning the structure of Bayesian networks

Require: A B-space=$(\mathcal{S}, \mathcal{O})$, where $\mathcal{S}$ is the space of possible DAGs, and $\mathcal{O} = \{\texttt{addArc}, \texttt{deleteArc}, \texttt{reverseArc}\}$ is the set of possible operators, an initial structure $S \in \mathcal{S}$, a dataset $\mathcal{D}$ of *i.i.d.* examples of a set $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ of random variables for a domain under study, a scoring function $\texttt{Score}(S, \mathcal{D})$

Ensure: A Bayesian network structure $S \in \mathcal{S}$ with high value of the score

```
continue ⇐ True
while continue do
    Compute Score(S, D)
    Find best operator op such that op = arg max_{op∈O} Score(op(S), D)
    if op exists ∧ Score(op(S), D) > Score(S, D) then
        S ⇐ op(S) {Apply the operator to the current structure}
    else
        continue ⇐ False
end while
return S {a structure with a high score}
```

The hill-climbing algorithm is cheaper in terms of memory and computing time when compared to other more sophisticated search algorithms. We can reduce its computational cost if we restrict the search space by limiting the number of parents for each variable or by providing a variable ordering. We can also limit the number of visited neighbor's structures at each search step by limiting the use of their basic operators. But as pointed out in [120], this simple search algorithm has three main drawbacks: *i*) *local maxima*: once on a local

maximum, hill-climbing will halt, even though there is a better solution; *ii*) *plateau*: hill-climbing will do a random walk in an area of the state space where the evaluation function is nearly flat; *iii*) *ridges*: a ridge can have steeply sloping sides, so that the search reaches the top with ease, but the top may slope only very gently toward a peak. Thus, the search may oscillate from side to side making little or no progress.

2.8 Concluding Remarks

In this chapter we have introduced the Bayesian networks along with the score-based approach to learn a Bayesian network structure from data. We have posed the learning problem as a search problem and showed the existing connection between this learning problem and the more general problem of model selection exposed in Chapter 1. By using all the formulae presented in the model selection problem, we could derive all the parameter estimators and scores for Bayesian networks. In the following chapter we discuss the most relevant issues related to the learning problem in the particular case when Bayesian networks are induced for classification. We can, in principle, learn Bayesian network classifiers from data using the same algorithms used for learning Bayesian networks. Thus, understanding all the issues related to this more general problem is crucial for a good understanding and further implementation of the learning algorithms for the more specific class of Bayesian network classifiers.

Chapter 3

Bayesian Network Classifiers

3.1 Introduction

Classification plays an important role in the field of *machine learning* [97, 98], *pattern recognition* [138] and *data mining* [55, 57]. A *classifier* is a function that assigns a class label to objects described by a set of attributes. *Supervised learning* is the task of building classifiers from data. The word "*supervised*" refers to the fact that the objects described in the data have known class memberships which were determined by a *supervisor* or *teacher*. The study of supervised learning is of growing interest and importance since many real-world problems can be modeled as classification problems. In a typical scenario [57], we have a categorical outcome measurement, that is, a class label (e.g. *heart attack/no heart attack*) that we wish to predict based on a set of attributes (e.g. *diet* and *clinical measurements*). We have a training dataset from which we observe the class label and the attribute values (measurements) for a set of objects (e.g. *people*). Using this dataset we build a *predictive model* (a *classifier*), which will enable us to predict the class label for new unseen objects. A good classifier is one that accurately predicts such a class label.

The *Naïve Bayes* classifier [39, 83] is one of the most used classifiers in real-world applications. Naïve Bayes significantly simplifies learning by assuming that attributes are independent given class. As this strong independence assumption is "*unrealistic*", Naïve Bayes has a *high bias*. However, in spite of it all, its performance is surprisingly good in practice when compared to other more sophisticated classifiers, a fact that was widely demonstrated and

argued in many works [38, 42, 56, 83, 86, 98, 115]. One of the reasons why we might expect Naïve Bayes to perform well is because it requires fewer parameters to be estimated than alternative classifiers. Hence, Naïve Bayes has a *low variance*. As argued in [38], a classifier as Naïve Bayes, with *high bias* and *low variance*, will tend to produce lower *zero-one loss* than one with *low bias* and *high variance*, a behaviour that is specially visible with smaller datasets. However, there has been plenty of effort for improving the predictive performance of Naïve Bayes by reducing the bias resulting from the assumptions of attribute independence.

In the next sections we introduce the problem of supervised learning along with the Naïve Bayes classifier and summarize several reasons why we might expect this surprisingly good performance of Naïve Bayes in practice. We further provide an overview of previous works aimed at improving the performance of Naïve Bayes and place a greater emphasis on those approaches that have attempted to relax the independence assumption by adding dependencies among the attributes. The works [43, 44] done by Friedman et al. are particularly relevant in this context as the first attempt in establishing a sound connection between the Bayesian classifiers and the theory of Bayesian networks. Since Bayesian networks provide a sound theoretical framework to represent and manipulate dependencies, from that moment until now BNCs have been the natural extension of the Naïve Bayes for improving its predictive performance. We provide a description of the main classes of Bayesian network classifiers found in the literature. Finally, we conclude with a discussion concerning how the choice of the *scoring function* and the *class-model* can affect the performance of Bayesian network classifiers learned from data.

3.2 Supervised learning

Let $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ be a vector of observed random variables, called *attributes*, where each attribute X_i takes values from its domain Ω_{X_i}. Each instantiation $\mathbf{x}$ of $\mathbf{X}$ is called an *example*. The space of all possible examples, $\Omega_{\mathbf{X}} = \Omega_{X_1} \times \ldots \times \Omega_{X_n} \subset \Re^n$, is called the *input space*. Let C be an unobserved random variable with values in a finite set $\Omega_C = \{c_1, c_2, \ldots, c_m\}$. C is called the *class variable* and the values of C are *classes* or *class labels*. The space of all possible classes, Ω_C, is called the *output space*.

Definition 31. A function $f : \Omega_{\mathbf{X}} \rightarrow \Omega_C$ that maps from the input space $\Omega_{\mathbf{X}}$ to the output

space Ω_C is called the **target function**.

In general, $f(\mathbf{x})$ is a random function. In the absence of noise, $f(\mathbf{x})$ is deterministic, which means that $f(\mathbf{x})$ always assigns the same class to a given example. In the particular case of *Boolean* outputs, that is, when each example $\mathbf{x} \in \Omega_{\mathbf{X}}$ is mapped onto exactly one of two possible classes (e.g. $\Omega_C = \{0, 1\}$), the target function is called the **target concept**.

Let further consider $f : \Omega_{\mathbf{X}} \rightarrow \Omega_C$ the target function to be learned.

Definition 32. A **labeled example** is a tuple $< \mathbf{x}, c >$ where $\mathbf{x} \in \Omega_{\mathbf{X}}$ is the example itself and $c = f(\mathbf{x}) \in \Omega_C$ is the class assigned by the target function f.

The problem of *supervised learning* can be stated as it follows:

> Given a training dataset $\mathcal{D} = \left\{< \mathbf{x}^{(1)}, c^{(1)} >, < \mathbf{x}^{(2)}, c^{(2)} >, \ldots, < \mathbf{x}^{(N)}, c^{(N)} >\right\}$ of *i.i.d.* labeled examples of $< \mathbf{X}, C >$ to induce a *classifier*, a hypothesis $h_{\mathcal{C}} : \Omega_{\mathbf{X}} \rightarrow \Omega_C$, that approximates f as closely as possible.

The induced classifier $h_{\mathcal{C}}$ can then be used to predict the class label of future examples. A classifier is, therefore, defined by a deterministic function, the hypothesis $h_{\mathcal{C}} : \Omega_{\mathbf{X}} \rightarrow \Omega_C$ that assigns a class label to any given example.

There are a range of supervised learning algorithms now available. In general, existing classifiers have been developed under four main approaches, namely: *i*) *symbolic learning*; *ii*) *instance-based learning*; *iii*) *neural networks*; and *iv*)*probabilistic classifiers*. A comprehensive review of all these supervised learning algorithms and classifiers as well as a comparative study of their performance on large real-world problems is given in [97]. This comparative study was supported by the well-known StatLog project [67]. The results showed that determining which of the algorithms performs best in practice depend critically on the dataset used. While it is well known that no algorithm can outperform all others in all the cases [71], in practice some supervised learning algorithms can be more successful than others.

Probabilistic classifiers are among the most popular classifiers used in the machine learning community. These classifiers are generally generated by an explicit underlying probabilistic model, which provides a probability of being in each class rather than a simple classification. Examples of probabilistic classifiers are *bayesian classifiers*, *Naïve Bayes*, *linear and*

quadratic discriminant, *logistic regression* and *Bayesian network classifiers*. In the probabilistic framework the goal of the learning task is to produce the classification predictive distribution $P(C \mid \mathbf{X})$. We further focus on bayesian classification, particularly, on the Naïve Bayes classifier and the Bayesian network classifiers, BNCs from now on.

3.2.1 Evaluating the Performance

In supervised learning is very important that induced classifiers are *accurate*. Hence, the most natural measure of the performance in classification problems is the *predictive accuracy*, or alternatively, the *error rate*. The classification for each test example can be either *correct*, if the classification agrees with the actual value, or *incorrect*, if it does not. The *accuracy* is represented by the proportion of correct classifications. The *error rate*, quite the opposite, is represented by the proportion of misclassified examples.

To assess the accuracy of a classifier $h_{\mathcal{C}}$ we need to define a *loss function*. Given a target function $f(\mathbf{x})$ a *loss function* for supervised learning is some function which, for any example $\mathbf{x}$, takes a prediction $h_{\mathcal{C}}(\mathbf{x})$ and the true class $f(\mathbf{x})$, and determines how much loss is incurred due to predicting $h_{\mathcal{C}}(\mathbf{x})$ when input is $\mathbf{x}$ and true output is $f(\mathbf{x})$. One of the loss functions commonly used in supervised learning is the *zero-one loss*. This function simple assigns a loss of *one* if the classification is *incorrect*, or *zero* otherwise.

Definition 33. The **zero-one loss function** of a classifier $h_{\mathcal{C}}$ with respect to a target function $f(\mathbf{x})$ and a example $\mathbf{x}$, denoted by $\delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x}))$, is defined as follows:

$$\delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x})) = \begin{cases} 1 & when\ f(\mathbf{x}) \neq h_{\mathcal{C}}(\mathbf{x}) \\ 0 & otherwise \end{cases} \tag{3.1}$$

Having the notion of the zero-one loss function $\delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x}))$ we can refine the goal of supervised learning. The goal is, therefore, to induce a classifier that minimizes the zero-one loss. We can now more formally define the error rate as follows:

Definition 34. The **error rate** of a classifier $h_{\mathcal{C}}$ with respect to a target $f(\mathbf{x})$ and a dataset $\mathcal{D}$ with N *i.i.d.* labeled examples is the proportion of *misclassified* examples by $h_{\mathcal{C}}$. That is,

$$Err(D, h_{\mathcal{C}}) \equiv error(D, f, h_{\mathcal{C}}) = \frac{1}{N} \sum_{\mathbf{x} \in D} \delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x})) \tag{3.2}$$

Alternatively, we can use another measure for assessing probabilistic classifiers based on the conditional predictive distribution $P(C \mid \mathbf{X})$. This measure is the *conditional log-loss* defined as the *log-score* of the conditional distribution $P(C \mid \mathbf{X})$. We consider that for each example $\mathbf{x} \in \mathcal{D}$ the learner suffers an individual loss equal to $-log\ P(c^{(t)} \mid \mathbf{x}^{(t)}, h_{\mathcal{C}})$ where $c^{(t)} = f(\mathbf{x}^{(t)})$ is the true output. We can interpret each individual loss as the logarithmic penalty that the classifier $h_{\mathcal{C}}$ would obtain for its probability prediction of the true class value for the example $\mathbf{x}$.

Definition 35. The **conditional log-loss** of a classifier $h_{\mathcal{C}}$ with respect to a target function $f(\mathbf{x})$ and a dataset $\mathcal{D}$ with N *i.i.d.* labeled examples is the sum of all the individual loss. That is,

$$ClogLoss(M^h, D) \equiv -\sum_{\mathbf{x} \in D} log\ P(c^{(t)} \mid \mathbf{x}^{(t)}, h_{\mathcal{C}}) \tag{3.3}$$

The *conditional log-loss* is just the negative of the *conditional log-likelihood* of a classifier $h_{\mathcal{C}}$ given data $\mathcal{D}$. Thus minimizing the *conditional log-loss* is equivalent to maximizing the *conditional log-likelihood*. Obviously, the *conditional log-loss* is the alternative to the *log-loss* defined in Section 1.5.6, but for supervised learning.

In this thesis we use the *zero-one loss* function for evaluating the performance of the induced classifiers. The error rate based on the *zero-one loss* tends to be an *over-optimistic* estimate of the performance if this is estimated from the same data used to build the classifier [97]. Learning algorithms, instead, should be evaluated and compared on the basis on how well they can generalize to examples that not are among those used to build the classifiers. As described in Section 1.5.6 *hold-out testing* and *cross-validation* are two of the most frequently used methods for estimating the predictive performance. *Hold-out testing* is more suitable for large sample sizes (more than 1000 examples) while *cross-validation* is more appropriate for intermediate sample sizes (about 1000 examples). *Cross-validation* is used mainly to reduce the bias of the error estimates. However, for very small datasets, a more appropriate method is *bootstrapping* [40]. Cross-validation and bootstrapping are both "resampling" methods. However, whereas *cross-validation* repeatedly analyzes subsets of the data, bootstrapping, in its simplest form, repeatedly analyzes subsamples of the data randomly sampled with replacement from the full dataset.

3.2.2 The Prequential Framework

It is also common to evaluate learning algorithms on training sets of different sizes and then generate learning curves that chart the predictive performance with increasing set size. One common technique is to learn classifiers from training sets of increasing sizes while maintain the same *test set* for evaluation. Instead, we are interested in evaluating induced classifiers in the Dawid's prequential framework where the predictive performance is computed *predictively* and *sequentially* through a *sequential updating* of the classifier.

Without loss of generality we further assume that at each learning step data arrives in batches B and that these batches are of equal size, each containing m examples. The goal of the *prequential scenario* is to sequentially predict the labels of each incoming batch of examples. Thus, at the t^{th} time point the incoming batch $B^{(t)}$ is first evaluated using the hypothesis $h_{\mathcal{C}}^{(t-1)}$ induced from the first $t-1$ batches. Then all the examples from $B^{(t)}$ along with its correct classes are used to update the current hypothesis. The learner aims to minimize the *zero-one loss*, that is, minimize the number of the *incorrected* classified examples over the total of examples classified so far. Algorithm 8 is the base algorithm for learning and evaluating supervised learning algorithms in the *prequential* framework.

Algorithm 4 The algorithm for learning and evaluating supervised learning algorithms in the *prequential* framework

Require: A classifier class-model $\mathcal{M}$, a dataset $\mathcal{D}$ of *i.i.d.* labelled examples $< \mathbf{x}, c >$ divided in batches B of m examples

Ensure: A classifier $h_{\mathcal{C}} \in \mathcal{M}$ updated at each time point, the error rate `errRate`

1: Initialize $h_{\mathcal{C}}$ with one of the hypothesis from M
2: **for** each batch B in $\mathcal{D}$ **do**
3: **for** each example $\mathbf{x}$ in B **do**
4: $h_{\mathcal{C}}(\mathbf{x}) \Leftarrow$ predict($\mathbf{x}$, $h_{\mathcal{C}}$) {first: predictions}
5: $f(\mathbf{x}) \Leftarrow$ getActualClass($\mathbf{x}$)
6: `cumIncorrected`$+= \delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x}))$ {the zero-one loss is used}
7: `totalEvaluated`$+= m$
8: `errRate` = `cumIncorrected/totalEvaluated`
9: update($h_{\mathcal{C}}$, B) {second: update the classifier with the examples from B}
10: **end for**
11: **return** $h_{\mathcal{C}}$ and `errRate`

3.2.3 Bias-Variance Decomposition of the Error Rate

The *bias-variance* decomposition of the error can help in understanding the relative behavior of learning algorithms. The origins of the *bias-variance* decomposition is related to the *quadratic loss function* in the context of *regression* [50]. Given a fixed target function and a dataset size, the conventional formulation of the decomposition breaks the expected error into the sum of three non-negative quantities [73]:

- **Intrinsic target noise**: this quantity is a lower bound on the expected error of any learning algorithm, that is, the expected error of the *Bayes-optimal* classifier[1].
- **Squared bias**: this quantity measures how well the average prediction of the learning algorithm over all possible datasets of the given size matches the target function.
- **Variance**: this quantity measures how much the prediction of the learning algorithm "*bounces around*" for different datasets of the given size.

Thus, consider a distribution over all the possible datasets of a fixed size for a specified domain. *Bias* measures the central tendency for the predictions of the classifiers induced by the same learning algorithm from the different datasets. *Variance* measures the degree to which the predictions differ from this central tendency from dataset to dataset.

The Bias-Variance Decomposition for the Zero-One Loss

Several *bias-variance decomposition* of the *zero-one loss* has been proposed (e.g. [42, 73]). We further describe the decomposition for the *zero-one loss* proposed by Kohavi and Wolpert in [73], one of the most widely employed of those available.

Let the target function $f : \Omega_{\mathbf{X}} \rightarrow \Omega_C = \{c_1, c_2, \ldots, c_m\}$ be a conditional probability distribution $P(C_f = c_f \mid \mathbf{X})$, where $C_f \in \Omega_C$. Let the classifier $h_{\mathcal{C}}$ generated by the learning algorithm be a similar distribution $P(C_h = c_h \mid \mathbf{X})$, where $C_h \in \Omega_C$. In order to derive the error decomposition Kohavi and Wolpert consider a definition of the *zero-one loss* that does not depend on the input variable $\mathbf{X}$. In their definition the *zero-one loss*, denoted by $l(c_f, c_h)$, assigns a penalty to a pair of values $c_f, c_h \in \Omega_C$ and is defined as $1 - \delta(c_f, c_h)$, where

[1] The *Bayes-optimal* classification for a given example is obtained by using the full-bayesian approach, that is by averaging the predictions of all the hypotheses, weighting by their posterior probabilities [98].

$\delta(c_f, c_h) = 1$ if $c_f = c_h$ and 0 otherwise. Their derivation is based on the *expected zero-one loss* $E(l)$, which is is usually referred as the *expected misclassification rate*. $E(l)$ is derived as follows:

$$\begin{aligned} E(l) &= \sum_{c_f,\ c_h} l(c_f, c_h) P(c_f, c_h) \\ &= \sum_{c_f,\ c_h} [1 - \delta(c_f, c_h)] P(c_f, c_h) \\ &= 1 - \sum_{c \in\ \Omega_C} P(c_f = c_h = c) \end{aligned}$$

The *bias-variance* decomposition of the **expected misclassification rate** $E(l)$ is equivalent to the decomposition of the *zero-one loss* and is given by:

$$E(l) = \sum_{\mathbf{x} \in \Omega_{\mathbf{X}}} P(\mathbf{x})(\sigma^2_{\mathbf{x}} + \texttt{bias}^2_{\mathbf{x}} + \texttt{variance}_{\mathbf{x}}) \tag{3.4}$$

where

$$\texttt{bias}^2_{\mathbf{x}} \equiv \frac{1}{2} \sum_{c \in \Omega_C} [P(C_f = c \mid \mathbf{x}) - P(C_h = c \mid \mathbf{x})]^2 \tag{3.5}$$

$$\texttt{variance}_{\mathbf{x}} \equiv \frac{1}{2} \left(1 - \sum_{c \in \Omega_C} P(C_h = c \mid \mathbf{x})^2 \right) \tag{3.6}$$

$$\sigma^2_{\mathbf{x}} \equiv \frac{1}{2} \left(1 - \sum_{c \in \Omega_C} P(C_f = c \mid \mathbf{x})^2 \right) \tag{3.7}$$

Informally speaking, $P(C_f = c \mid \mathbf{x})$ is the probability that the fixed target f takes on the value c at point $\mathbf{x}$. $P(C_h = c \mid \mathbf{x})$, on the other hand, can be interpreted as the average (over datasets generated by f) the class c is guessed by $h_{\mathcal{C}}$ at point $\mathbf{x}$. Therefore, the $\texttt{bias}^2$ *term* measures the squared difference between the target's averaged output and the classifier's average output. Hence this term measures the persistent error of the learning algorithm. The `variance` *term*, instead, measures the variability of $P(C_h \mid \mathbf{x})$, that is, the error produced by the fluctuation when generating a single classiffier. As algorithm becomes more sensitive to changes in the datasets, the variance increases. Therefore, given a distribution over datasets, the variance only measures the sensitivity of the learning algorithms to changes in the data and it is independent of the underlying target function. The noise σ^2 measures the variance of the target. Thus, the definitions of *variance* and *noise* are identical only differing in the random variable, C_h or C_f, accordingly.

A Methodology for Computing the Bias-Variance Components

Kohavi and Wolpert proposed the following methodology for estimating the $\texttt{bias}^2$ and the `variance` *terms* in the decomposition of the *zero-one loss* given in Equations 3.5 and 3.6:

1. Randomly split the dataset $\mathcal{D}$ into two parts, $\mathcal{D}_{\texttt{train}}$ and $\mathcal{D}_{\texttt{test}}$. $\mathcal{D}_{\texttt{train}}$ is used to sample the training sets, while $\mathcal{D}_{\texttt{test}}$ is used to evaluate the terms in the decomposition.

2. Get training sets of size N, chose $\mathcal{D}_{\texttt{train}}$ to be of size $2N$. This guarantees does not get many duplicates training sets, even for small values of N. From $\mathcal{D}_{\texttt{train}}$ generate k training sets by using uniform random sampling without replacement.

3. Run the learning algorithm on each of the training set. Then, estimate the terms $\textbf{bias}^2$ and **variance** using the generated classiffier for evaluating each example $\mathbf{x}$ of the test set $\mathcal{D}_{test}$. Estimate all the needed probabilities using frequency counters.

3.3 Naïve Bayes Classifier

A common approach to supervised learning is to associate each class c_j with a discriminant function $f_j(\mathbf{x})$ and then assign the example to the class whose discriminant function is maximum. Bayesian classifiers use the class posterior probabilities $P(C = c_j \mid \mathbf{X} = \mathbf{x})$ as discriminant functions. For short, let $P(c_j \mid \mathbf{x})$ denote the probability that the example $\mathbf{X} = \mathbf{x}$ belong to the class c_j. As proved in [39], if we have $P(c_j \mid \mathbf{x})$ for each class $c_j \in \Omega_C$, the *zero-one loss* is minimized *if, and only if*, the example $\mathbf{x}$ is assigned to the class c^* for which $P(c^* \mid \mathbf{x})$ is maximum. That is,

$$c^* = h_{\mathcal{C}}(\mathbf{x}) = \arg \max_{j=1...m} P(c_j \mid \mathbf{x}) \tag{3.8}$$

We can then apply the Bayes' theorem to derive the posterior probability of each class c_j given an example $\mathbf{x}$

$$P(c_j \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c_j)P(c_j)}{P(\mathbf{x})}$$

$P(\mathbf{x})$ can be ignored since it is the same for all the classes (as usually, a normalization constant). This yields Bayes discriminant functions:

$$f_j(\mathbf{x}) = P(\mathbf{x} \mid c_j)P(c_j) \tag{3.9}$$

Therefore, Bayes classifier finds the MAP hypothesis given the example $\mathbf{x}$. That is,

$$h_{\mathcal{MAP}}(\mathbf{x}) = \arg \max_{j=1...m} P(\mathbf{x} \mid c_j)P(c_j) \tag{3.10}$$

To compute $h_{\mathcal{MAP}}$ for a given example $\mathbf{x}$ we need to estimate the prior probability $P(c_j)$ and the conditional probability distribution $P(\mathbf{x} \mid c_j)$ for each class $c_j \in \Omega_C$. Direct estimation of $P(\mathbf{x} \mid c_j)$ is hard when the input space is *high-dimensional*, unless we introduce some assumptions on the model that allow to decompose this probability into a product of conditional probabilities, one for each attribute. For instance, under the very naïve assumption that the attributes are independent given the class, $P(\mathbf{x} \mid c_j)$ can be decomposed into a product of n terms, one for each attribute. Hence we have:

$$P(\mathbf{x} \mid c_j) = \prod_{i=1}^{n} P(X_i = x_i \mid c_j) \tag{3.11}$$

Applying 3.11 into 3.10 we obtain the Naïve Bayes classifier where the class c^* attached to the example $\mathbf{x}$ is given by the expression:

$$c^* = h_{\mathcal{NB}}(\mathbf{x}) = \arg \max_{j=1...m} \prod_{i=1}^{n} P(X_i = x_i \mid c_j)P(c_j) \tag{3.12}$$

3.3.1 Learning Naïve Bayes from Data

Our aim is to predict from a training dataset $\mathcal{D}$ of N labeled examples the class of an unseen example $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ where x_i is the value of the i^{th} attribute. To this end, we seek estimates $\hat{P}(c_j \mid \mathbf{x})$ of each class conditional probability $P(c_j \mid \mathbf{x})$ for all $c_j \in \Omega_C$. From Equation 3.12 we have that the class attached to the example $\mathbf{x}$ is the class $c^* \in \Omega_C$ such that $c^* = \arg\max_j \hat{P}(c_j \mid \mathbf{x})$ where

$$\hat{P}(c_j \mid \mathbf{x}) = \prod_{i=1}^{n} \hat{P}(X_i = x_i \mid c_j)\hat{P}(c_j) \tag{3.13}$$

Assuming the training dataset $\mathcal{D}$ is a representative sample of the joint distribution from which it is drawn, we can use $\mathcal{D}$ to compute the estimates for each term in Equation 3.13. We further consider a domain $\mathbf{X}$ of discrete attributes $\{X_1, X_2, \ldots, X_n\}$ where each attribute X_i may take on values from its finite domain $\Omega_{X_i} = \left\{x_i^1, x_i^2, \ldots, x_i^{r_i}\right\}$ where r_i represents the number of possible values of X_i. For simplicity, define $X_i = k$ the event that $X_i = x_i^k$. At training time Naïve Bayes needs only obtain the sufficient statistics in order to compute the

estimates $\hat{P}(c_j)$ for each class $c_j \in \Omega_C$ and the estimates $\hat{P}(X_i = k \mid c_j)$ for each attribute value $x_i^k \in \Omega_{X_i}$ and each class c_j. Thus, the set of sufficient statistics $\mathbf{T}(\mathcal{D} \mid NB)$ of Naïve Bayes is given by:

1. A table $\mathtt{T_c}$ containing m counters N_j, one counter for each class $c_j \in \Omega_C$.

2. For each attribute X_i:

   - A $m \times r_i$ contingency table $\mathtt{CT}_i$ containing the counters N_{ijk}. Each N_{ijk} is the number of cases in $\mathcal{D}$ such that $X_i = k$ and $C = c_j$.

Assuming complete data[2], the computation of all the required estimates requires a simple scan through the data, an operation of time complexity $\mathcal{O}(Nn)$, where N is the number of training examples. At classification time, to classify a single example has time complexity $\mathcal{O}(mn)$ using the tables formed at training time with space complexity $\mathcal{O}(mnr_{avg})$ where r_{avg} is the average number of values per attribute [141].

We can further adopt a *frequentist* or a *bayesian* approach to estimate the parameters of the Naïve Bayes. Usually, the frequentist ML estimator with the Laplace correction for avoiding zero counters as proposed in [38] is used. Nevertheless, as argued in [4, 31, 128] Bayesian estimates tend to avoid overfitting, to be more robust and in general, less sensitive to the presence of zeroes in frequency counts.

3.3.2 Analysis of the Naïve Bayes Performance

It has been widely observed in many applications that Naïve Bayes: *i*) *is simple*; *ii*) *learns quickly*: it only requires a single pass through the data; *iii*) *predicts quickly*: it needs low computations to make predictions; *iv*) *is easily interpretable*: its results as probabilities are easy to understand and apply; *v*) *is naturally incremental*: it needs only to accumulate sufficient statistics. Nevertheless, the fact that the independence assumption is clearly almost always violated in practice has led often to underestimate the classification power of Naïve Bayes in favor of more sophisticated classifiers (*neural networks*, *decision trees*, etc.) on the grounds that the latter can provide more accurate classifications. However, empirical

[2]Missing values can be handled either by simply ignoring them or by introducing "*missing*" as an extra attribute value.

comparisons have shown that Naïve Bayes performs surprisingly well so often when compared with other more complex classifiers.

For instance, Hand and Yu in [56] provided an overview of the empirical studies comparing Naïve Bayes with other more sophisticated classifiers and summarized some reasons why Naïve Bayes performs so well. One well-known reason is that Naïve Bayes requires fewer parameters to be estimated, and hence, lower variance for all the parameter estimates. This reduction in variance can compensate the effect of the high bias resulting from the strong independence assumption. As proved by Friedman in [42] "*certain types of (very high) bias can be canceled by low variance to produce accurate classification*", as it happens with Naïve Bayes. By providing a *bias-variance* decomposition of the classification error, Friedman also argued that focusing on improved probability estimation when the goal is to produce accurate classification may be mistaken. Good probability estimates are not necessary for good classification as long as an optimal classifier is rather obtained when both, the target and learned distributions agree on the most-probable class. Estimates of the class conditional probability $P(c_j \mid \mathbf{x})$ from Equation 3.13 clearly has two source of bias:

1. The bias resulting from the *modeling error*, that is, from the assumptions of attribute independence.

2. The bias resulting from the *estimation error*, that is, from the use of the parameter estimates.

Since the independence assumption is *unrealistic*, the bias resulting from the *modeling error* can be quite large, especially in high dimensional settings involving many attributes. Introduced estimates for parameters can introduce further bias and also variance. This high degree of bias associated with Naïve Bayes makes it generally inappropriate to approximate the target predictive distribution $P(C \mid X)$. However, Naïve Bayes performs surprisingly well because of the relatively low variance of its estimates of $P(c_j \mid \mathbf{x})$. Although these estimates present a high bias, this may not matter in classification tasks, because all that need to be preserved is the *rank order*: $\hat{P}(c_j \mid \mathbf{x}) > \hat{P}(c_k \mid \mathbf{x})$ whenever $P(c_j \mid \mathbf{x}) > P(c_k \mid \mathbf{x})$ [56].

All these argumentations related to the good performance of Naïve Bayes were supported by the experimental and theoretical results obtained by Domingos and Pazzani in [38] and also by Rish et al. in [115]. Domingos and Pazzani compared the performance of several

classifiers on several UCI's benchmark problems [102]. They found that Naïve Bayes often outperforms other classifiers, even when there is substantial attribute dependence. Moreover, they proved Naïve Bayes optimality for some problems with high degree of attribute dependence (*disjunctive* and *conjunctive concepts*). For measuring the degree of attribute dependence they use the *conditional mutual information* (see Definition 20). In addition, they conducted an empirical study comparing two extensions to Naïve Bayes [75, 106] that relax the independence assumption by joining two attributes into a new compound attribute but using two different joining criteria: *i*) the *conditional mutual information*; and *ii*) the LOO-CV score under *zero-one loss*. They reached two crucial conclusions:

1. There is a low correlation between the degree of attribute dependence and the performance of the Naïve Bayes.

2. *Cross-validation accuracy* is a better predictor of the effect of an attribute join than the *conditional mutual information*.

For the first conclusion it could follow that searching for attribute dependencies is not necessarily the best approach for improving the performance of Naïve Bayes. For the second, that it would be more appropriate for attribute joining the use of a score optimized for classification. In the next sections we will provide a more in-depth analysis about these two issues.

3.4 Improving Naïve Bayes. Related Work

There has been plenty of work attempted to improve the predictive performance of the Naïve Bayes mainly following three approaches: *i*) *attribute subset selection*; *ii*) *improving the probability estimates*; and *iii*) *relaxing the independence assumptions*. Some authors [75, 83] have argued that Naïve Bayes is very robust to noise and irrelevant attributes. However, approaches based on attribute (feature) subset selection may help in improving the Naïve Bayes performance, specially when the attribute space is highly dimensional. Feature subset selection (FSS) involves identifying the relevant attributes in a dataset and giving only that subset to the learning algorithms. By reducing the number of attributes we reduce the number of parameters to be estimated, and hence, the variance of the test error.

There are two main approaches to FSS: *i*) the *wrapper* approach; and *ii*) the *filter* approach. Similarly to *model selection*, the wrapper approach to FSS (scoring & search) can be exposed as a *discrete optimization problem* where a selection criterion is optimized in the space of possible subset of attributes. The selection criterion is usually a predictive score based on the estimates of the predictive accuracy. Thus, the FSS task is to find a subset of attributes so that the resulting predictive distribution yields the most accurate classifications for future data. *Filter approaches* to FSS, instead, are constraint-based approaches. They search for a good subset of attributes using only the intrinsic characteristics of the data. The most common way is to rank the attributes in terms of the value of some scoring function (e.g. probabilistic or distance metrics, information-theoretic measures, etc.) and then choose the attributes with the highest scores. A comparison of wrapper and filter approaches to the FSS problem for learning BNCs in the domain of biomedical informatics is given in [8, 63]. A wrapper approach to FSS for improving the performance of Naïve Bayes was proposed in [84]. This combination of FSS with Naïve Bayes is known as "*selective Naïve Bayes*". For further reading in the FSS problem the reader is referred to the relevant work [72], where a complete overview of the general problem of FSS in supervised learning focusing on wrapper approaches is provided.

In the next subsections we provide a compact overview of the work aimed at improving Naïve Bayes more related to this thesis, giving a special emphasis to the Iterative Bayes, which improves the parameter estimates; and several classes of BNCs, which relax the independence assumptions.

3.4.1 Improving the Probability Estimates

Several researches have examined ways of achieving better performance than Naïve Bayes by improving its probability estimates $\hat{P}(X_i = x_i \mid c_j)$ and/or $\hat{P}(c_j)$, thus reducing their bias and/or variance. Interested readers can follow the reference [86] for an overview of these approaches. For instance, Webb and Pazzani in [139] proposed a method that adjusts only the estimates $\hat{P}(c_j)$ of the prior probability for each class c_j. The adjustment for each class c_j is done by means of a numeric weight, which is inferred using a hill-climbing search procedure. During classification, the class's probability $P(c_j)$ is multiplied by the resulting weight to obtain the adjusted value. Webb and Pazzani showed that the use of this adjusted

value as the estimate $\hat{P}(c_j)$ allows to significantly improve the accuracy of the Naïve Bayes.

The Iterative Bayes proposed by Gama in [46], instead, improves the predictive distribution $P(C \mid \mathbf{x})$ associated with each example $\mathbf{x}$ by adjusting the estimates $\hat{P}(X_i = x_i \mid c_j)$ of the conditional probabilities. The algorithm begins with the CPTs built by the Naïve Bayes followed by an optimization process which consists of an iterative updating of the sufficient statistics. In each iteration and for each example the corresponding conditional probabilities are updated so as to increase the probability on the correct class. The iterative procedure finishes when a stopping criterion is reached. Experimental evaluation of Iterative Bayes have shown consistent reductions of the error rate. Using the bias-variance decomposition of the error described in Section 3.2.3, Gama empirically demonstrated that the reduction of the error is mainly due to a reduction on the bias component. Hence, Iterative Bayes can reduce the bias of the Naïve Bayes resulting from the estimation error.

A further computational advantage of Iterative Bayes is that it lends itself directly to incremental learning. In [47] we introduced Adaptive Bayes, an incremental version of Iterative Bayes, that can also work in an on-line learning framework. The rationale is that after seeing a new example, we first increment the corresponding counters as usually do with Naïve Bayes. Then we can run Iterative Bayes using only this example so as to increase the probability on its actual class. Experimental results showed that the use of Adaptive Bayes in both, an incremental and on-line learning framework, has significant advantages in comparison against the non-adaptive Naïve Bayes. In a further work [19] we evaluated Adaptive Bayes in the context of a user modeling task. The results from conducted experiments using simulated data showed that Adaptive Bayes seems to be a good and simple alternative to the Naïve Bayes in user modeling tasks where concept drift can take place.

Iterative Bayes is an adaptive algorithm that we have widely used in our investigation. Hence, in Section 3.6 we will provide more details about its updating procedure and present the Iterative Bayes algorithm for the more general class of BNCs.

3.4.2 Relaxing the Attribute Independence Assumption

Many attempts have been made to extend Naïve Bayes trying to maintain its simplicity and efficiency while relaxing the attribute independence assumption. One of the pioneer exten-

sions to the Naïve Bayes is the *semi-Naïve Bayes classifier*, first implemented by Kononenko in [75] and later by Pazzani in [106]. The semi-Naïve Bayes classifier joins two or more attributes in a new compound attribute - a cartesian product of a subset of attributes. While Kononenko used conditional independence tests as the joining criterion, Pazzani achieved better results by using the LOO-CV score.

The works [43, 44] of Friedman et al. are particularly relevant as a first attempt in establishing a sound connection between the Bayesian classifiers and the theory of Bayesian networks. They proposed to augment the Naïve Bayes structure with arcs among attributes thus relaxing the strong independence assumption. Adding the best set of augmenting arcs is computationally expensive because we need to search among DAGs containing the Naïve Bayes structure. To overcome computational limitations they proposed to learn a more restricted structure called *Tree Augmented Naive Bayes* (TAN). TAN extends Naïve Bayes by allowing the attributes to form a tree, that is, each attribute has only one attribute as a parent. Friedman et al. used *conditional mutual information* as the criterion for parent selection and a variation of Chow and Liu's algorithm [28] for learning tree-like structures from complete data. TAN outperformed Naïve Bayes while it is also computational simple because searches over structures are not involved. In addition, Friedman et al. proposed to generalize TAN by using *Bayesian multinets* [49] as classifiers. They partitioned the training dataset by classes and for each class learned a Bayesian network using a generalization of Chow and Liu's algorithm that learns *multinets* consisting of tree structures. Results from conducted experiments showed that TAN *multinets* perform as well as TANs do.

In a later work [66], Keogh and Pazzani proposed two improvements for finding the set of augmented parents of a TAN structure: *i*) a *hill climbing search procedure* using only arc additions and the LOO-CV score; *ii*) the *Super Parent* method (SP-TAN) - a more efficient search heuristic aimed at reducing the computational cost involved with the use of the LOO-CV score. During the search process, they also considered the deletion of irrelevant attributes. Moreover, unlike TAN, SP-TAN does not necessarily add every arc in the tree of the TAN structure. On the contrary, SP-TAN stops adding arcs when there are no more improvements on the score. As a result, SP-TAN builds a more simple and effective classifier than the original TAN. Results from similar Friedman's experiments showed that SP-TAN outperforms TAN and Naïve Bayes. However this improvement is obtained at a considerable

computational cost due to the use of the LOO-CV score.

In [43, 44] Friedman et al. also implemented the hill-climbing search algorithm with the MDL score for learning other kinds of classifiers: *i*) *unrestricted augmented Naïve Bayesian networks*; and *ii*) *unrestricted Bayesian network classifiers*. The former are known as Bayesian networks Augmented Naïve Bayes (BANs) and the latter as General Bayesian Networks (GBNs) [22]. BANs extend Naïve Bayes by allowing the attributes to form an arbitrary graph, rather than just a tree. GBNs, on the contrary, treat the class node as an ordinary node, which means that the class node does not need to be the parent of all the attribute nodes. Friedman et al. showed that the MDL score does not necessarily optimize the performance of the induced Bayesian networks when they are used for classification. Since the MDL score heavily biases for simpler networks, the hill-climber procedure is not able to find enough dependencies among the attributes. Specially for smaller datasets and when there are many classes, the Naïve Bayes structure itself requires many parameters, and the addition of an augmenting arc involves adding at least as many parameters as the number of classes. Because the number of parameters increases, the penalty complexity term in the MDL score grows largely so that the addition of any single arc does not improve the score. In Section 3.7.1 we will provide an analysis about these issues related to the choice of an appropriate score for learning BNCs.

Sahami in [121] introduced the k-Dependence Bayesian Classifiers (k-DBCs) - an unified framework for all the bayesian classifiers containing the structure of Naïve Bayes (e.g. Naïve Bayes itself, TAN, BAN, etc.). A k-DBC allows each attribute to have a maximum of k attribute nodes as parents. The learning algorithm proposed by Sahami generalizes the algorithm for learning TANs [43] and, like it, also uses the mutual information as a measure of the degree of dependence. Sahami compared his original algorithm with another variant that introduces a threshold for the conditional mutual information. This threshold avoids the inclusion of parents whose dependencies are not significant. From the results of experiments comparing the performance of several k-DBCs for different k values ($k = 0, 1, 2, 3$) on several datasets, Sahami concluded that modeling attribute dependencies can improve the classification results.

A comparison between filter and wrapper approaches to attribute selection (FSS) using k-DBCs is given in [8, 86]. The filter approach is a variation of Sahami's algorithm. This

uses the mutual information as the selection criterion, but with the restriction that not all the attributes can be added to the Naïve Bayes structure. The wrapper approach implements a hill-climbing search algorithm with a *cross-validation* score. These algorithms for learning k-DBCs using FSS were recently evaluated in the domain of medical bioinformatics in order to distinguish between two subgroups of cirrhotic patients [8].

In a more recent paper [141], Webb et al. have proposed a new approach for improving the accuracy of Naïve Bayes. Their approach averages all classifiers from the class of 1-DBCs, so they called it AODE (short form of "*aggregating one-dependence estimators*"). AODE attempts in retaining the simplicity of Naïve Bayes while relaxing the attribute independence assumption but without incurring at a great computational cost, as for instance, SP-TAN does. Results from conducted experiments suggest that AODE is more accurate than Naïve Bayes. This presents substantially lower bias at the cost of a very small increase of the variance. Results also show that AODE outperforms TAN and is very competitive with SP-TAN. Although the classification time increases a little, AODE has lower variance but higher bias while considerably reducing the learning time compared to SP-TAN. To reduce the time for classifying a given example, Webb et al. propose not to include in the averaged predictions those models for which the probability estimates are inaccurate (unbiased) taking into account the number of cases in the training datasets from which these estimates were computed.

Comparative Studies of BNCs

An overview of several classes of BNCs as well as a comparative study of their performance was given by Cheng and Greiner in [22, 23]. They particularly focused on the study of BANs, GBNs and unrestricted Bayesian multinets using constraint-based approaches and compared their performances against Naïve Bayes and TAN. Cheng and Greiner used the CBL_1 algorithm (a precursor of the TPDA algorithm [24]) to learn the structure of Bayesian networks. The search procedure builds a Bayesian network through three phases: *i*) *drafting*, which builds an initial tree-like structure using Chow-Liu's algorithm; *ii*) *thickening*, which adds arcs to the draft; *iii*) *thinning*, which verifies the necessity of each arc. The original algorithm requires a threshold for determining how much mutual information between two nodes is considered as significant. To overcome the use of a threshold, a *wrapper algorithm*

that searches for the optimal threshold was implemented. The CBL_1 algorithm is based on the conditional mutual information. Experimental results showed that this algorithm for learning unrestricted BNCs does not suffer from the limitations pointed out by Friedman et al. in [43, 44] using the hill-climber search with the MDL score. The induced classifiers were competitive with the best reported results.

A study of the performance of BNCs induced with different scores was presented by Madden in [93]. Madden compared the performance of Naïve Bayes, TANs and GBNs classifiers induced with different approaches. He learned TANs using the original algorithm from [43] but with the K2 score and the GBNs using a modified version of the K2 learning algorithm [29]. His results were compared against the results reported by Friedman et al. in [43] and by Cheng and Greiner in [23]. As observed, the results were significantly different although it was proved that the MDL score is asymptotically equivalent to the K2 score (a Bayesian score) [10, 58] and that the structure search based on maximizing a score is equivalent to the structure search based on conditional independence tests [31]. Madden performed a more in-depth analysis of the implementation of each learning algorithm with the aim of finding the sources of disparities. He found that the differences in the performance are caused by the differences in the parameter estimators and heuristic search algorithms used rather than by differences in the scoring functions. Madden also claimed that the results obtained with TANs were very similar to each other, which provided experimental support to the theoretical results presented in [31], since for TANs search is not required.

3.5 Bayesian Networks as Classifiers

As stated, in classification problems the domain variables are partitioned into two separate sets: the attribute set $\mathbf{X} = \{X_1, X_2, \ldots, X_n\} \in \Omega_{\mathbf{X}}$ and the set consisting of a single class variable $C \in \Omega_C = \{c_1, c_2, \ldots, c_m\}$. The aim is to correctly predict the value c of the class variable C given an example $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$. If the performance measure is the *predictive accuracy*, the optimal prediction for $\mathbf{x}$ is the class c^* that maximizes the posterior probability distribution $P(C \mid \mathbf{x})$ [39].

A Bayesian network can be used for classification in a quite straightforward way. One of the variables is selected as the class variable, and the remaining variables as attribute

variables. Next, inference methods can be used to calculate the marginal distribution of the class variable given the value of the attributes. Thus, a Bayesian network can be used as a classifier that gives the posterior distribution $P(C \mid \mathbf{x})$ of the class node $C \in \Omega_C$ given an example $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$. We can compute the posterior probability $P(c_j \mid \mathbf{x}, S)$ for each class $c_j \in \Omega_C$ by marginalizing the joint probability distribution $P(c_j, \mathbf{x} \mid S)$ as follows:

$$P(c_j \mid \mathbf{x}, S) = \frac{P(c_j, \mathbf{x} \mid S)}{P(\mathbf{x} \mid S)} \propto P(c_j, \mathbf{x} \mid S) \tag{3.14}$$

and then by returning the class c^* that maximizes the joint probability distribution:

$$c^* = h_{\mathcal{BNC}}(\mathbf{x}) = \arg \max_{j=1\ldots m} P(c_j, \mathbf{x} \mid S) \tag{3.15}$$

3.5.1 Classes of Bayesian Network Classifiers

In this section we formally define four classes of BNCs introduced previously in Section 3.4.2[3]. These classes are: Naïve Bayes, TAN, BAN and GBN. Naïve Bayes represents the most restrictive class of BNCs because it strictly allows no dependencies between attributes given the class value. On the other extreme, GBNs represent the less restrictive class of BNCs because no restrictions are imposed to the dependencies among the variables. Moreover, whereas a common feature of Naïve Bayes, TAN and BAN is that the class node is treated as a special node, GBNs treat the class node as an ordinary node, that is, the class node is not necessarily a parent of all the attributes.

Definition 36. A **N**aïve **B**ayes (NB) classifier can be viewed as a Bayesian network with a simple structure that has the class node as the parent node of all other attribute nodes. Figure 3.1 shows an example of the Naïve Bayes structure.

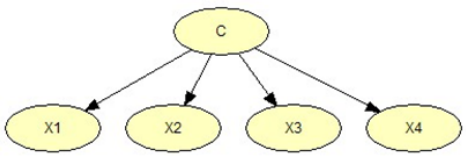

Figure 3.1: A Naïve Bayesian network classifier structure

[3]TANs have been widely studied in [8, 22, 23, 43, 44, 66, 86, 93, 141], BANs in [22, 23, 43, 44] and GBNs in [22, 23, 32, 43, 44, 52, 86, 93].

Definition 37. A **T**ree **A**ugmented **N**aïve Bayes (TAN) classifier is a Bayesian network which contains the structure of the Naïve Bayes and allows each attribute to have only one attribute node as parent. Figure 3.2 shows an example of a TAN classifier structure.

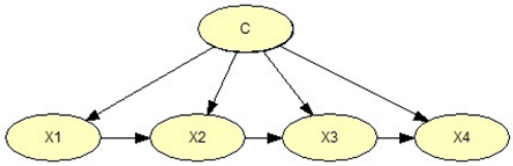

Figure 3.2: A TAN classifier structure

Definition 38. A **B**ayesian network **A**ugmented **N**aïve Bayes (BAN) classifier is a Bayesian network which contains the structure of the Naïve Bayes and allows the attribute to form an arbitrary DAG. Figure 3.3 shows an example of the structure of a BAN classifier.

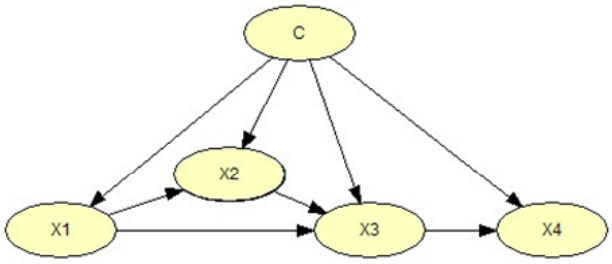

Figure 3.3: A BAN classifier structure

Definition 39. A **G**eneral **B**ayesian **N**etwork (GBN) classifier is an unrestricted Bayesian network that is used for classification tasks. Figure 3.4 shows an example of a GBN structure.

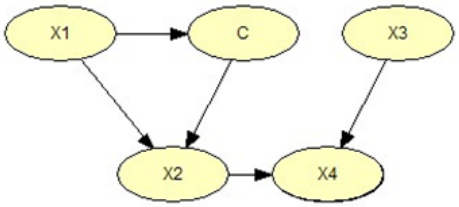

Figure 3.4: A GBN classifier structure

In a Bayesian network any variable is influenced only by its *Markov Blanket* composed by its parent variables, its children variables and the parent variables of its children variables [108]. Therefore, we should use only the nodes that belong to the *Markov blanket* of the class node for computing the predictive distribution of a GBN classifier.

3.5.2 k-Dependence Bayesian Classifiers

k-Dependence Bayesian Classifiers [121] represent an unified framework for all those classes of BNCs that contain the structure of the Naïve Bayes, such as Naïve Bayes itself, TAN, BAN, etc. More formally,

Definition 40. A k-DBC is a Bayesian network which contains the structure of the Naïve Bayes and allows each attribute to have a maximum of k attribute nodes as parents.

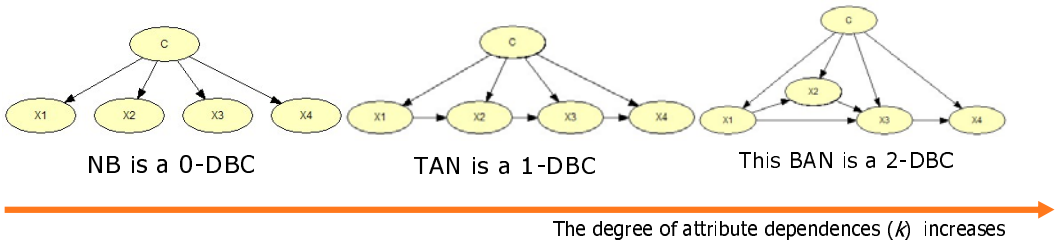

Figure 3.5: Examples of k-Dependence Bayesian Classifiers

As illustrated in Figure 3.5 we can vary the value of k, starting from $k = 0$ until $k = n-1$ and obtain classifiers that smoothly move along the spectrum of attribute dependencies. Obviously, Naïve Bayes is a 0-DBC that lies at the most restrictive end because it strictly allows no dependencies between attributes given the class value. TANs are 1-DBCs because they allow only one dependence among the attributes. The BAN shown in Figure 3.5 is a 2-DBC because it has a maximum of two dependencies among the attributes. At the most general extreme lies the full augmented Naïve Bayes classifier, a $(n-1)$-DBC, with no independence among the attributes.

Sahami's Learning Algorithm

Sahami in [121] presented a learning algorithm for inducing k-DBCs from data, which remains much of the computational efficiency of the Naïve Bayes algorithm since searches are not involved. The algorithm can be viewed as a generalization of the algorithm for learning TAN structures proposed by Friedman et al. in [43, 44].

The rationale is as follows. Starting with a k-DBC's structure S with a single class node C, the algorithm iteratively add $m = min(|S|, k)$ parents to each new attribute added to S with largest dependence with the class C. The m parents for each new attribute are selected among those with higher degree of dependence given the class. As measure of the degree of attribute dependence Sahami used the mutual information. The process finishes when all the attributes have been added to the structure S. Algorithm 5 depicts the pseudo-code of Sahami's algorithm for learning k-DBCs.

Algorithm 5 Sahami's algorithm for learning k-DBCS

Require: A dataset $\mathcal{D}$ of N labeled examples of $< \mathbf{X}, C >$, the k value for the maximum allowable degree of attribute dependence

Ensure: A k-DBC

1: $V \Leftarrow C$ {the set of nodes for the k-DBC}
2: $A \Leftarrow \emptyset$ {the set of arcs for the k-DBC}
3: `Temp` $\Leftarrow \emptyset$ {the used attribute list}
4: **for all** attributes X_i and pair of attributes (X_i,X_j) such that $X_i \neq X_j$ **do**
5: Compute $I(X_i, C)$ from data $\mathcal{D}$
6: Compute $I(X_i, X_j \mid C)$ from data $\mathcal{D}$
7: **repeat**
8: Select X_{max} such that $X_{max} = \arg \max_{X_i \notin \texttt{Temp}} I(X_i, C)$
9: Add the node X_{max} to V
10: Add the arc (C, X_{max}) to A
11: Add $m = min(|\texttt{Temp}|, k)$ arcs to A from m distinct attributes $X_j \in$ `Temp` with the highest value of $I(X_i, X_j \mid C)$
12: Add the attribute X_{max} to `Temp`
13: **until** `Temp` includes all the attributes $X_i \in \mathbf{X}$
14: Compose S such that $S = (V, A)$ {the k-DBC structure}
15: Estimate the parameters Θ_S given S from data $\mathcal{D}$
16: **return** k-DBC= (S, Θ_S)

3.6 Iterative Bayes for Bayesian Network Classifiers

Consider a set $\mathbf{X} \in \Omega_{\mathbf{X}}$ of discrete attribute variables $\{X_1, X_2, \ldots, X_n\}$ where each attribute X_i may take on values from its finite domain $\Omega_{X_i} = \{x_i^1, x_i^2, \ldots, x_i^{r_i}\}$ and r_i is the number of its possible values. Let $C \in \Omega_C = \{c_1, c_2, \ldots, c_m\}$ be the class variable. Suppose we observe a dataset $\mathcal{D}$ of *i.i.d.* labelled examples $< \mathbf{x}, c = f(\mathbf{x}) >$ of $< \mathbf{X}, C >$, where $f(\mathbf{x})$ is the target function to be learned. The aim is to learn a Naïve Bayes classifier $h_{\mathcal{C}}$ from the dataset $\mathcal{D}$.

The Iterative Bayes proposed by Gama in [46] iteratively updates the contingency tables of the Naïve Bayes in order to improve the estimates of the conditional probabilities associated with each training example. For each example of $\mathcal{D}$ the aim is to iteratively update the corresponding sufficient statistics so as to increase the probability of the correct class. Suppose that during a iteration we process an example $\mathbf{x}^{(l)} = (x_1^{(l)}, x_2^{(l)}, \ldots, x_n^{(l)})$ which belongs to the class $c_{\texttt{obs}} \equiv f(\mathbf{x}^{(l)})$ and that it is classified by the current classifier $h_{\mathcal{C}}$ as belongs to the class $c_{\texttt{pred}}$, that is, $c_{\texttt{pred}} \equiv h_{\mathcal{C}}(\mathbf{x}^{(l)})$. The rationale of the updating procedure is as follows:

1. An increment, `delta`, proportional to the difference $1 - P(c_{\texttt{pred}} \mid \mathbf{x})$ is computed. If the example is incorrectly classified, that is, $c_{\texttt{pred}} \neq c_{\texttt{obs}}$, then `delta` is multiplied by -1 in order to be a negative quantity.

2. For each contingency table $\texttt{CT}_i$ associated to the attribute X_i only the counters in the column where $X_i = x_i^{(l)}$ are updated as follows:

   (a) the increment `delta` is added to the entry where $C = c_{\texttt{pred}}$.

   (b) the increment `delta` is proportionally subtracted to the remaining entries.

   (c) for avoiding zero or negative counters the counters never goes below 1.

Figure 3.6 illustrates how Iterative Bayes updates the probability estimates of the Naïve Bayes in the *balance* domain[4]. Consider the example of the class `"Right"` depicted in this figure. Before running Iterative Bayes, the classification for this example is correct, but the *confidence* on this prediction is low (about 59%). Once obtained the predictive probability we run Iterative Bayes. First, we compute `delta` (a positive increment because the classification

[4]The balance domain is a benchmark problem from the UCI's repository [102]. We provide a complete description of this domain in Section 4.2.3.

is correct). Next the increment `delta` is used to update the contingency tables. For instance, for the contingency table corresponding to the attribute `left_W` we only need to update the entries in the column where `left_W=1`, that is, the entries of the first column labeled `I1`. The increment `delta` is added to the entry where the class variable takes the value `Right` and proportionally subtracted to the remaining entries. After iteratively cycling for all the examples we obtain a higher confidence level (about 79%) for the probability $P(\texttt{Right}|\mathbf{x})$. As a result, Iterative Bayes guarantees that after an example $\mathbf{x}^{(l)}$ is seen, the probability $P(c^{(l)}|\mathbf{x}^{(l)})$ is increased. Hence the Iterative Bayes attempts in minimizing the *conditional log-loss* defined in Equation 3.3 instead of the *zero-one loss* function.

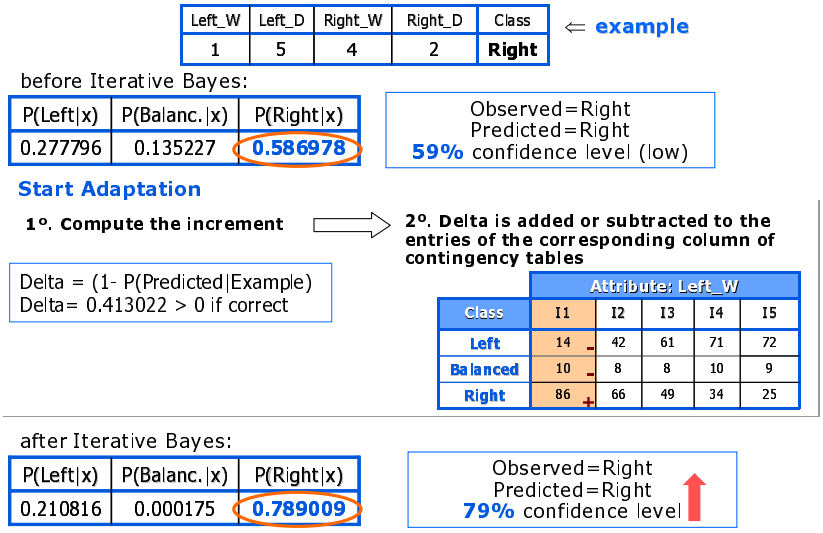

Figure 3.6: An example of the updating procedure of the Iterative Bayes

Iterative Bayes can be easily extended to be used with BNCs. We only need to use the same updating procedure for each table of sufficient statistics associated to each attribute X_i separately. Thus, we further consider a BNC, that is, a classifier $h_{\mathcal{C}} = (S, \Theta_S)$ over $< \mathbf{X}, C >$. Let $\texttt{CT}_i$ denote the subset of counters N_{ijk} from the set of sufficient statistics $\mathbf{T}(\mathcal{D} \mid S)$ associated to the attribute X_i. As before, let q_i denote the number of possible parent's configurations for X_i. Each $\texttt{CT}_i$ can be viewed as a *contingency table* composed of q_i rows and r_i columns, one row for each possible parent configuration $\mathbf{Pa}_i = j$ and one column for each possible attribute value x_i^k. Algorithm 6 presents the pseudo-code of the Iterative Bayes for improving the parameter estimates of BNCs. Once the increment `delta`

is computed, it is added to the corresponding entries of each contingency table $\mathtt{CT}_i$ where the class variable C takes on value $\mathtt{c}_{\mathtt{pred}}$ in the parent configurations pa_i^j from $\mathbf{Pa}_i$.

Algorithm 6 The Iterative Bayes algorithm for Bayesian Network Classifiers

Require: A current BNC, that is, a classifier $h_{\mathcal{C}} = (S, \mathbf{\Theta}_S)$ over $< \mathbf{X}, C >$, a dataset $\mathcal{D}$ of *i.i.d.* labeled examples $< \mathbf{x}, c = f(\mathbf{x}) >$ of $< \mathbf{X}, C >$, where $f(\mathbf{x})$ is the target function to be learned, a set of contingency tables $\mathtt{CT}_i$, one table for each attribute variable X_i.

Ensure: An increase of the confidence level of the correct class $c^{(l)}$ for each example $\mathbf{x}^{(\mathbf{l})}$ of $\mathcal{D}$

repeat
  for each example $\mathbf{x}^{(l)} = (x_1^{(l)}, x_2^{(l)}, \ldots, x_n^{(l)}) \in \mathcal{D}$ **do**
    $\mathtt{c}_{\mathtt{obs}} \Leftarrow f(\mathbf{x})$ {observe the correct class}
    $\mathtt{c}_{\mathtt{pred}} \Leftarrow h_{\mathcal{C}}(\mathbf{x})$ {predict the class using the current set of parameters Θ_S}
    $\mathtt{delta} \Leftarrow 1 - P(\mathtt{c}_{\mathtt{pred}} \mid \mathbf{x})$ {compute the increment}
    if $c_{\mathtt{pred}} \neq \mathtt{c}_{\mathtt{obs}}$ **then**
      $\mathtt{delta} \Leftarrow \mathtt{delta} \times -1$
    for each contingency table $\mathtt{CT}_i$ associated to the attribute X_i **do**
      $k \Leftarrow$ GetColumnNumber($\mathtt{CT}_i$) where $X_i = x_i^{(l)}$
      for each parent configuration $pa_i^j \in \mathbf{Pa}_i$ **do**
        if $(C = \mathtt{c}_{\mathtt{pred}}) \in pa_i^j$ **then**
          $N_{ijk} += \mathtt{delta}$ {the entry in the j^{th}row is increased if $(C = \mathtt{c}_{\mathtt{pred}})$ is in pa_i^j}
        else
          $N_{ijk} -= \mathtt{delta}/|\Omega_C|$
        if $N_{ijk} < 1$ **then**
          $N_{ijk} \Leftarrow 1$
until a maximum of a user-defined number of iterations is reached
return $h_{\mathcal{C}} = (S, \Theta_S)$ with the updated set of parameters

3.7 Learning Bayesian Network Classifiers

The learning problem for BNCs is typically solved by first choosing a suitable class of BNCs (e.g. Naïve Bayes, TAN, BAN, etc.) almost certainly based on our knowledge and intuitions about the given problem. This class of BNCs defines a *class-model* in our terminology. If the chosen class-model is a restricted BNC (e.g. TAN, BAN, k-DBCs for a given k value), then the space of the feasible Bayesian network structures $\mathcal{S}$ is defined according to the imposed restrictions. Given a training dataset $\mathcal{D}$ of *i.i.d.* labeled examples of $< \mathbf{X}, C >$ the learning problem consists of selecting the BNC, that is, the hypothesis $h_{\mathcal{C}} = (S, \mathbf{\Theta}_S)$, such that $S \in \mathcal{S}$, that yields the most accurate classifications for future data. In principle, we can solve this learning problem using the same algorithms that we use when Bayesian networks

are induced for general purposes. We can induce a Bayesian network that encodes a joint probability distribution and uses it for classification as shown in Equation 3.15. However, when a Bayesian network is induced for classification, the main goal is to build an accurate classifier. As pointed out by Kontkanen in [78]:

> *"Bayesian networks producing the most accurate predictive distribution in the joint probability estimation sense does not necessarily perform well in classification tasks, unless the joint probability distribution represents the true domain probability distribution exactly".*

We further focus on *score-based* approaches to learn BNCs. This learning problem - a *model selection problem* - can be approached as a discrete optimization problem where a score that measures the quality of each candidate structure is optimized in the space $\mathcal{S}$ of feasible structures. The quality of a BNC is defined in terms of its *predictive accuracy*. There are two aspects that are crucial in order to induce an accurate BNC from data using a score-based approach:

1. The choice of an appropriate scoring function.
2. The choice of an appropriate class-model (in the sense of its complexity) according to the amount of training data available.

In the next subsections we provide a discussion about how these two aspects can affect the performance of BNCs induced from data.

3.7.1 Choosing a Scoring Function

The choice of an appropriate score according to the learning goal is crucial for score-based approaches to model selection. In the previous chapters we have stated that there are many alternative approaches for constructing theoretically valid scoring functions. However, when Bayesian networks are used for classification, we are interested that the resulting predictive distribution yields accurate classifications. The learning goal, therefore, is to choose a model which is expected to give the most accurate classifications for future data. A model selection criterion of *goodness of fit* based on the joint predictive distribution will not necessarily be

optimal for classification purposes. Under this perspective, we can distinguish two types of model selection criteria [77]: *i*) *unsupervised* scores; and *ii*) *supervised* scores.

Examples of unsupervised scores are all the scores optimized for the *log-likelihood* or related function (e.g. MLC, BIC, AIC, MDL) or the *log-marginal likelihood* (e.g. BD, K2, BDeu, etc.). All these scores are optimized for the *log-loss* on the joint distribution defined in Equation 1.36. Unsupervised scores are not specialized for classification tasks, which can result in suboptimal choices during the search process, and hence, one of the reasons why some BNCs induced with one of these scores cannot always improve the performance of the Naïve Bayes.

In supervised model selection, on the contrary, the goal is to choose the model which yields the most accurate classifications with respect to a given loss function. Hence supervised scores are specialized for classification tasks. One *frequentist* approach is, for instance, the use of scores based on the *conditional log-likelihood*. However, the main drawback of using the *conditional log likelihood* is that unlike the likelihood (see Equation 2.15), the conditional likelihood of a Bayesian network does not decompose over its structure [43]. This means that we cannot decompose the parameter estimation problem into local estimation problems as we previously did in Section 2.5. Thus, the computational cost of finding the parameters that maximizes the conditional likelihood is prohibitive. One practical solution to overcome this problem was proposed by Grossman and Domingos in a recent paper [52]. They proposed to choose the structure that maximizes the conditional likelihood while using the ML estimates as parameter estimators, that is, the set of parameters that maximizes the log-likelihood. As shown the ML estimates of a Bayesian network under some nice assumptions can be efficiently computed in a closed-form solution. Moreover, it was proved [43] that ML estimates are asymptotically equivalent to the maximum conditional likelihood estimates.

If we opt for a Bayesian approach to model selection we can instead use the *conditional marginal likelihood* as a supervised score. Like conditional likelihood the computation of the conditional marginal likelihood is generally not computationally feasible, even in the cases when the marginal likelihood can be computed in a closed-form solution. In [77] Konkatnen obtained an approximation of the *conditional marginal likelihood* by replacing it by a single conditional likelihood and using the MAP estimates for the parameters. Alternatively, we can use predictive scores such as *cross-validation* and *prequential* scores as described in Section

2.6.4. Predictive scores can be optimized for classification tasks since they measure the predictive performance of the models and, in addition, they can be easily modified for different loss functions. The most commonly used loss functions with cross-validation and prequential scores for learning BNCs are the *zero-one loss* and the *conditional log loss*.

In recent years the issue of *unsupervised* versus *supervised* learning of BNCs has received a lot of attention [32, 38, 43, 52, 78]. Unsupervised score-based learning approaches may result in more poor BNCs than those ones induced with supervised scores. Although asymptotically unsupervised criteria theoretically can find optimal BNCs, in practice they may not be the best guide in the search process. Several researchers have empirically shown examples for which BNCs induced with unsupervised scores perform badly. In [43] Friedman et al., for instance, investigated the use of the MDL score for learning BANs and unrestricted BNCs. They empirically showed that MDL does not optimize the classification accuracy of the induced BNCs. They suggested the use of supervised scores based on the conditional likelihood. However, they recognized that the computation of the conditional likelihood of Bayesian networks is computationally intractable. Hence, they suggested the exploration of good approximations of the conditional likelihood, as it was done later, for instance, by Grossman and Domingos [52]. In a similar setting, Cowell [32] performed a simulation study using the set of all DAGs with 4 or 5 nodes. Cowell also concluded that finding good classifiers using a score-based learning approach with the BD score (the log marginal likelihood) might produce bad classifiers. Kontkanen et al. in [78] performed experiments using several datasets from the UCI's repository [102] aimed at comparing the unsupervised score BD against supervised ones (k-Fold-CV, Preq and the *conditional marginal likelihood*). The results showed that supervised criteria clearly outperform unsupervised ones. The best results were obtained with the prequential approach, but similarly the k-Fold-CV score also showed a good performance. The success for cross-validation was explained through the existing connection between the *cross-validation* score and the *supervised marginal likelihood*.

Whereas there is a considerable previous work on comparing *unsupervised* versus *supervised* learning of BNCs, we are more interested in investigating other aspects of different model selection criteria that can affect the performance of BNCs. Namely, how different scores are capable of handling the *bias-variance* trade-off in learning BNCs according to the chosen *class-model* and the available data. As stated in Section 1.4 all model selection cri-

teria that are used in practice perform a trade-off between *goodness of fit* and *complexity* of the models involved [53]. Indeed, model selection makes a *bias-variance* trade-off in order to select a model with the appropriate complexity [25, 57], which is automatically regularized by the model selection criterion.

Van Allen and Greiner in [3, 4] and later Chung-Chieh in [128] conducted empirical comparisons of various scores in order to identify how each one handles *bias-variance* in learning Bayesian network structures for small datasets. From their empirical studies the authors concluded that it is difficult to identify which model selection criterion is better than another because they were quite sensitive to the amount of data provided. They also suggested "*caution*" in applying penalized likelihood criteria such as MDL and AIC (specially MDL), as they may lead to *underfitting* the data. MDL and AIC can be dominated by their complexity penalty portion, thus displayed a bias for simplicity. Specially for the MDL score, this situation becomes more critical for smaller datasets and complex domains with many classes and attribute values. In this case, the addition of a new arc involves the addition of many parameters [43], and hence, the penalty complexity term has a greater influence in the computation of the score.

On the other hand, the MLC while very simple is not appropriate for learning Bayesian network structures. MLC tends to favor complete DAGs, that is, structures in which every variable is connected to every other variable. The same happens with the BDeu score. As pointed out in [44] such complete networks are not useful, since they do not provide any useful representation of the independences in the learned distributions. A complete network can overfit the data, specially for smaller datasets, since it has a great number of parameters, and hence, an extremely high variance. In general, small datasets for very complex domains result in a more challenged *bias-variance* trade-off. In this case it would be more appropriate the use of scores that not biased to much, neither, for simpler nor for complex networks, as for instance, the BD score does. As argued in Section 1.5.2 the marginal likelihood decreases as models become more complex. Thus, the BD score tends to select models less complex than MLC does, thus performing a more optimal trade-off between *complexity* and *fitness to data*.

Therefore, to obtain a good performance of BNCs induced from data it is also crucial choosing a scoring function according to how this one is able to handle the *bias-variance*

trade-off for selecting a model with the appropriate complexity for the available data in each particular domain.

3.7.2 Choosing the Class-Model

Other aspect that can also influence the performance of learning algorithms for BNCs is the selection of the *class-model*. There is a direct trade-off between the complexity of a classifier and its predictive performance. Finding the appropriate balance between *complexity* and *performance* is a matter of handling the *bias-variance trade-off*. Increasing model complexity decreases bias but increases the variance in the parameter values. In practice, complex classifiers cannot perform well when tested on unseen data, they can *overfit* the training data. Otherwise, simpler classifiers cannot capture the true structure in the data, they can *underfit* the data. Both, *overfitting* and *underfitting* lead to a deterioration of the performance. *Underfitting* increases the bias component of the test error while *overfitting* increases the variance component. As shown in Figure 3.7, there is an optimal model complexity that gives the best performance on unseen data.

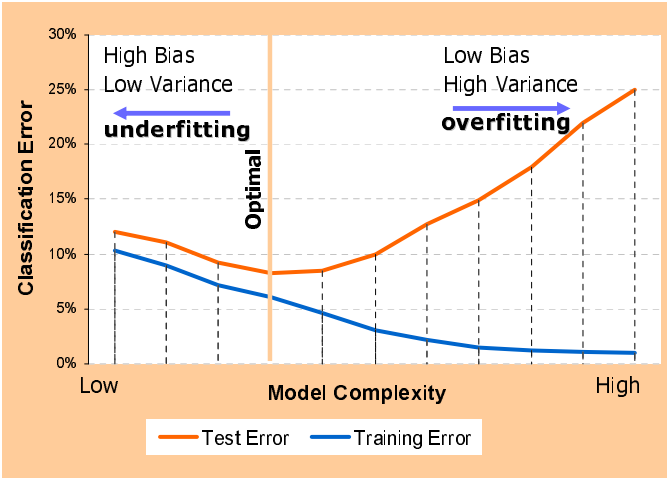

Figure 3.7: Behavior of the test error and the training error varying the model complexity

A simple classifier as Naïve Bayes, with *high bias* and *low variance*, will tend to produce lower *zero-one loss* that more complex classifiers with *low bias* and *high variance*. Since variance decreases with increasing sample size, this behaviour should be specially visible with smaller sample sizes. Thus, given a limited sample it would be more appropriate to use

the simplest Naïve Bayes classifier that no fits the data so good but with a more optimal *bias-variance* trade-off. On the other hand, Naïve Bayes can also perform well for large datasets, because its relatively large bias may not has influence on its performance [38]. However, Kohavi in [71] showed that for some larger datasets the accuracy of Naïve Bayes does not scale up as well as, for example, using decision trees. The high bias produced from the independence assumption yields that achievable accuracy may asymptote early and will not improve much as data size increases. For discrete data, because only few parameters need to be estimated, the estimates tend to stabilize quickly and more data does not change the underlying model much. This fact is specially supported by the results of an empirical study performed by Brain and Webb [13] in order to investigate how the data set size may affect the *bias-variance* decomposition of the test error. Their results using Naïve Bayes showed that as training data increases variance will decrease and this will become a less significant part of the error. On the contrary, the bias component becomes a large portion of error. For this reason, they suggested that variance management may not be very relevant as training set size increases. Instead, we must place more focus on *bias management*.

Different classes of BNCs presented in Section 3.5.1 attempt to reduce the bias of the Naïve Bayes by relaxing the attribute independence assumption. One of the main differences among them is the way each of them restrict the number of parents for each node. We can arrange them in order of their increasing complexity as follows: TAN, BAN, and so on. We should adjust the complexity of BNCs to suit the amount of training data. A simple way to control the model's complexity is therefore by selecting an appropriate class-model according to the available training data. This regularization of the complexity must lead to the selection of simpler class-models when we have few data and of more complex ones as training data increases, thus avoiding the problems caused by either too much bias (*underfitting*) or too much variance (*overfitting*).

3.8 Concluding Remarks

The main goal of this chapter was to introduce the Naïve Bayes classifier as well as different classes of BNCs through a revision of previous work. We have shown that Bayesian networks is a widely established framework as classification tool aimed at improving the performance

of the Naïve Bayes classifier by relaxing its "*unrealistic*" attribute independence assumption. Moreover, we have presented Iterative Bayes, a parameter refinement procedure that we have successfully implemented in our experimental studies. At the end of this chapter we discussed the implications of choosing a particular score and a particular *class-model* for the performance of BNCs induced by score-based learning algorithms. We argue that to obtain the desirable performance of induced BNCs it is crucial not only the choice of an appropriate model selection criterion but also the choice of an appropriate class-model according to the available data. Moreover, the selection of the appropriate score depends not only on the learning goal (that is, whether it is specialized for the classification task or not) but also on how this score makes the *bias-variance* trade-off for selecting a model with the appropriate complexity for the available training data. Most of the time, however, we have no clear idea of how to use our domain knowledge and intuitions for selecting an appropriate model selection criterion and class-model. This selection is still more challenging in a *prequential learning framework* since the amount of training data varies over time. In the next chapter we provide the results along with an in-depth analysis of a experimental study using the class of k-DBCs that aimed at exploring these issues.

Chapter 4

A Study of k-Dependence Bayesian Classifiers

4.1 Introduction

The class of k-Dependence Bayesian Classifiers (k-DBCs) introduced by Sahami in [121] provides an unified framework for characterizing all the classes of BNCs that contain the structure of Naïve Bayes such as, NB itself, TAN, BAN, etc. This class is very suitable for scaling up the complexity of BNCs according to the current amount of training data. By varying the maximum allowable degree of attribute dependence k we can obtain classifiers that smoothly move along all the spectrum of attribute dependencies, thus providing a flexible control over the model's complexity. The simplest way to regulate the complexity of k-DBCs is, therefore, by choosing an appropriate k-value according to the data available.

We carried out an experimental study using the class of k-DBCs to investigate how:

1. the choice of the score,
2. the choice of the class-model (i.e. the k-value); and
3. the size of the training data

can affect the performance of k-DBCs induced with the same underlying learning algorithm in a prequential learning framework.

Our study is different from existing related studies in three main aspects:

1. Whereas previous studies evaluated several scores in learning BNCs [32, 38, 43, 78, 93] in a batch learning scenario, that is, when all the training examples are given at the same time to the learning algorithm, our goal was to investigate how the choice of the score affects the performance of BNCs in a prequential learning framework.

2. Whereas previous studies focused on the comparison of *unsupervised* versus *supervised* scores [32, 38, 43, 52, 78] our goal was to evaluate other aspects related with the choosing of the score, namely how different scores are capable of handling the *bias-variance*, *complexity-performance* trade-offs according to the chosen class-model and the amount of training data provided.

3. Whereas previous work [3, 4, 128] compared several scores in order to identify how each one handles the bias-variance trade-off for small samples and in the general context of learning Bayesian Networks, we have investigated the *bias-variance* trade-off for large datasets and for the particular case when Bayesian Networks are induced for classification.

Finally, we would like to stress that this study was basically motivated by the fact that our main interest is in the development of adaptive algorithms for learning BNCs in a prequential learning scenario, where the amount of training data varies over time. Therefore, one of our main goals with these experiments was to test whether it makes sense to gradually increase the k value in order to adjust the complexity of the class-model, and thus, the complexity of the induced k-DBCs to the current amount of training data.

4.2 Experimental Setting

We evaluated the performance of several k-DBCs for different scores in a prequential learning framework using three large datasets from the UCI repository [102].

4.2.1 Underlying Learning Algorithms

We used two underlying learning algorithms to induce k-DBCs:

- for $k = 0$: the algorithm for learning the parameters of the NB (see Section 3.3.1).

- for $k > 0$: a hill-climbing learning algorithm further described.

The Hill-Climbing Algorithm for Learning k-DBCs

Instead of using the learning algorithm proposed by Sahami in [121] based on the computation of the conditional mutual information, we relied on a hill-climbing learning algorithm as proposed, for instance, in [8, 86]. We chose a hill-climber mainly due to its obvious simplicity for computational implementation.

Algorithm 7 The hill-climbing algorithm for learning k-DBCs

Require: A dataset $\mathcal{D}$ of N labeled examples of $< \mathbf{X}, C >$, the k value for the maximum allowable degree of attribute dependence, a scoring function $\texttt{Score}(S, \mathcal{D})$, the space $\mathcal{S}$ of possible `DAGs` restricted by k

Ensure: A k-DBC with high value of $\texttt{Score}(S, \mathcal{D})$

Initialize S to the NB structure
continue $\Leftarrow$ True
while continue **do**
  Compute $\texttt{Score}(S, \mathcal{D})$
  Find arc $(X', X'') = \arg\max \texttt{Score}(S \bigcup (X_i, X_j), \mathcal{D})$, where $|\mathbf{pa}(X_i) \setminus C| < k \ \wedge \ |\mathbf{pa}(X_j) \setminus C| < k$
  if arc (X', X'') exists $\wedge$ $\texttt{Score}(S \bigcup (X', X''), \mathcal{D}) > \texttt{Score}(S, \mathcal{D})$ **then**
    Add the arc (X', X'') to S
  else
    continue $\Leftarrow$ False
end while
Estimate the parameters Θ_S given S from data $\mathcal{D}$
return k-DBC$=(S, \Theta_S)$

Algorithm 7 is the hill-climbing algorithm for learning k-DBCs. We provide the learning algorithm with a *dataset* of labelled examples, the k value for the maximum allowable degree of attribute dependence and a *scoring function*. The rationale is as follows. 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 no possible to add a new arc. Once a network structure is chosen, the parameters are estimated using the selected *estimator*. As a result, the algorithm outputs a k-DBC with a high score. All the learning algorithms were implemented in Java using Weka's classes for BNCs [11, 145].

Scores

In our experiments we compared the following scores derived in Section 2.6:

- five *unsupervised* scores: MLC, MDL, AIC, BD and BDeu
- two *supervised scores*: k-Fold-CV (a.k.a k-Fold or CV) and Preq

Since we used different scores for the same learning algorithm, this helped us in ensuring that any differences in performance are due to the differences in the scores, and not to differences in the underlying learning algorithm.

Parameter Estimators

We used the ML estimates for parameters. For avoiding zero counters we smoothed the estimates a little and initialized all the frequency counters to 0.5, which is equivalent to use the Bayesian estimates with the *non-informative* assignment $\alpha_{ijk} = 0.5$. As shown in [4], the use of Bayesian estimators with Bayesian networks allows avoiding *overfitting*.

4.2.2 The Prequential Scenario for Learning and Evaluating k-DBCs

In the current study with k-DBCs we implemented a such-called *revolutionary approach* for updating the current k-DBC with new data. At each time point Algorithm 7 was invoked in order to rebuild the current hypothesis from all the available training data. Therefore, after new data is available, the current classifier is dropped and a new classifier is built from scratch using all the examples seen so far. This approach was called *naïve* by Friedman and Goldszmit in the context of the sequential updating of Bayesian networks [45]. As they argued, since learning from scratch use all the data provided so far, the *naïve* or *revolutionary* approach for updating the classifier is essentially optimal in terms of the quality of the models it can induce. Thus, this helped us in ensuring a more objective study of the performance for different class-models and scores. However, building a new classifier at each learning step requires a great amount of memory to store all the data and a lot of computer time.

To compare and evaluate the performance of different k-DBCs for different scores using the *prequential* learning framework we split each dataset into N batches of 100 examples.

At the t^{th} learning step we used the first t batches as *training data* and the examples of the next $(t+1)^{th}$ batch as *test data*. The performance was measured as the *cumulative accuracy*. Algorithm 8 is the base algorithm for learning and evaluating k-DBCs in a prequential learning framework using the revolutionary updating approach.

Algorithm 8 The algorithm for learning and evaluating k-DBCs in a prequential learning scenario using a revolutionary updating approach

Require: A dataset $\mathcal{D}$ of labeled examples of $< \mathbf{X}, C >$ divided in batches of m examples, the k value for the maximum allowable degree of attribute dependence, a scoring function $\texttt{Score}(S, \mathcal{D})$

Ensure: A k-DBC with a high score, the cumulative accuracy `cumAcc`

1: `trainingData` $\Leftarrow$ the first batch of m examples of $\mathcal{D}$
2: k-DBC $\Leftarrow$ learn-k-DBC(`trainingData`, k, $\texttt{Score}(S, \mathcal{D})$) {using the Algorithm 7}
3: `cumCorrected` $\Leftarrow 0$
4: `totalEvaluated` $\Leftarrow 0$
5: **for** each next batch B of m examples of $\mathcal{D}$ **do**
6: `testData` $\Leftarrow B$ {first: predict}
7: `cumCorrected` $+= \texttt{Evaluate}(\texttt{testData}, k\text{-DBC})$
8: `totalEvaluated` $+= m$
9: `cumAcc` $\Leftarrow$ `cumCorrected`/`totalEvaluated`
10: `trainingData` $\Leftarrow$ `trainingData` $\bigcup B$ {second: update the classifier}
11: k-DBC $\Leftarrow$ learn-k-DBC(`trainingData`, k, $\texttt{Score}(S, \mathcal{D})$, ...) {rebuild from scratch using the Algorithm 7}
12: **end for**
13: **return** k-DBC, `cumAcc`

4.2.3 Datasets

We evaluated the performance of k-DBCs on three large datasets from the UCI repository [102]:

1. **balance**: This dataset was generated to model psychological experimental results. This is a *three-class* problem, with *four continuous attributes*. The attributes are the left-weight, the left-distance, the right-weight, and the right-distance. Each example is classified as having the balance scale tip to the right, to the left, or to be balanced. The correct way to find the class is the greater of left-distance $\times$ left-weight and right-distance $\times$ right-weight. If they are equal, this is balanced.

2. **nursery**: This dataset was derived from ranking applications for nursery schools. It was used during several years in Ljubljana, Slovenia, where there was excessive enrollment to

these schools and the rejected applications frequently needed an objective explanation. The final decision depended on three sub-problems: occupation of parents and child's nursery; family structure and financial standing; and social and health picture of the family. This is a *five-class* problem, with *eight nominal attributes*.

3. **adult**: This dataset was extracted from the US Census Bureau database. This is a *two-class* problem, with *six continuous attributes* and *eight nominal attributes*. The prediction task is to determine whether a person makes over 50K a year. This is a hard domain to learn.

We used the *nursery* dataset available on the UCI repository [102] and a discretized version of the adult dataset available on-line in [76]. Since we needed datasets with large number of examples to better explore the behaviour of incremental algorithms, we randomly generated samples of 10000 examples for the *balance* domain using its underlying rules. Moreover, we removed instances with missing values from all the datasets. As a result we used 12800 instances for nursery (128 learning steps) and 16000 instances for adult (160 learning steps), respectively. Thus, we consider all the datasets to be discrete and complete. While comparing learning algorithms in the *prequential* learning framework, it is important does not judge their performances on a single sequence of data. For avoiding the effect of example ordering we generated 10 samples for each dataset. At each learning step, the performance was measured as the average of the cumulative accuracy over the 10 samples. The main characteristics of the three chosen datasets are summarized in Table 4.1. This table also shows the classification accuracy of batch learning algorithms for different classes of BNCs. Superscripts denote references from which these results were taken.

Table 4.1: Datasets used in the empirical study with k-DBCs

	#	#	#	Learning	Reported Results			
Dataset	Attrib.	Classes	Inst.	Steps	NB	TAN	BAN	GBN
balance	4	3	10000	100	91.43[46]	n/a	n/a	n/a
nursery	8	5	12800	128	90.48[93]	94.16[93]	93.08[22]	92.63[93]
adult	14	2	16000	160	84.18[22]	86.01[22]	85.82[22]	86.11[22]

4.3 Results and Analysis

For each evaluated dataset the experiments lead us to the cross-analysis of tree factors: the k-DBC *class-model* (that is, the k value), the *score* and the *training dataset size*. We can pick a *score* and then compare the performance of different k-DBCs varying the k value at different time points. This kind of analysis for each score allows us to evaluate how increasing the k value above 0 would affect the performance of k-DBCs with increasing training data. Thus, we can analyze how the selection of a class-model with a particular complexity would affect the performance of k-DBCs for different amounts of training data. Alternatively, we can pick a particular *class-model* (a k value) and then compare the performance of several k-DBCs which belong to a same class-model but induced with different scores at different time points. This kind of analysis for each class-model allows us to evaluate how the selection of a particular score would affect the performance of k-DBCs for different amounts of data.

4.3.1 Comparing the Performance of k-DBCs per Score

Tables 4.2, 4.3 and 4.4 show the gains in the accuracy of induced k-DBCs with respect to the accuracy of the NB at 10 selected time points grouped by score for the *balance*, *nursery* and *adult* domains, respectively. We evaluated whether the accuracy differences were significant at the 5% level using a one-tailed *paired t-test*. For comparison purposes the first line ($k = 0$) shows the averaged accuracy (in %) of the NB. The remaining lines ($k > 0$) show the gains in accuracy over the NB. A gain of, say, "+1.0" means that the corresponding k-DBC significantly improves NB in 1.0%. A "no" means that the corresponding k-DBC obtains neither no significant gains nor improvements over NB.

Using these comparative tables allows us to identify for each particular score if increasing the k value above 0 leads to significant improvements over NB. In each learning step and for each score the best results giving maximum improvements are reported with bold text. From the results we can make the following observations. **First**, improvement is not noticed for smaller training dataset (of ≤ 1000 examples) since for all the domains and scores there is practically no k-DBC that significantly outperforms NB. These results may corroborate the hypothesis that NB performs better than more complex k-DBCs for smaller datasets. **Second**, gradual gains in accuracy for $k > 0$ are noticed as training data increases. This

Table 4.2: Significative gains of the predictive accuracy of k-DBCs with respect to the Naïve Bayes *per score* for the *balance* dataset

		Learning Step (t), # training examples: $t \times 100$									
Score	k	5	10	30	40	50	60	70	80	90	100
	0	**85.70**	**87.80**	**90.17**	**90.44**	**90.63**	**90.95**	**91.05**	**91.17**	**91.22**	**91.30**
MLC	1	no	no	no	no	no	no	no	no	no	+0.31
	2	no	no	no	no	no	+0.55	+0.98	+1.28	+1.54	+1.77
	3	no	no	**+3.15**	**+4.52**	**+5.33**	**+5.69**	**+6.07**	**+6.30**	**+6.53**	**+6.66**
MDL	1	no	no	no	no	**+0.19**	**+0.30**	**+0.44**	**+0.55**	**+0.66**	**+0.72**
	2	no	no	no	no	+0.19	+0.30	+0.44	+0.55	+0.66	+0.72
	3	no	no	no	no	+0.19	+0.30	+0.44	+0.55	+0.66	+0.72
AIC	1	no	no	no	no	no	no	no	+0.28	+0.42	+0.50
	2	no	no	no	**+0.55**	+0.98	+1.26	+1.58	+1.81	+2.01	+2.20
	3	no	no	no	+0.55	**+1.04**	**+2.08**	**+2.98**	**+3.60**	**+4.13**	**+4.48**
BD	1	no	no	no	no	no	no	no	+0.36	+0.51	+0.58
	2	no	no	no	+0.71	+1.10	+1.34	+1.63	+1.81	+2.02	+2.19
	3	no	no	**+1.99**	**+3.64**	**+4.62**	**+5.09**	**+5.56**	**+5.86**	**+6.14**	**+6.30**
BDeu	1	no	no	no	no	no	no	no	no	no	+0.29
	2	no	no	no	no	no	+0.66	+1.06	+1.35	+1.63	+1.86
	3	no	no	**+3.14**	**+4.50**	**+5.31**	**+5.67**	**+6.06**	**+6.29**	**+6.52**	**+6.65**
k-Fold	1	no	no	no	no	no	no	no	no	+0.30	+0.40
	2	no	no	no	no	+0.78	+1.16	+1.53	+1.78	+2.02	+2.22
	3	no	no	**+3.29**	**+4.61**	**+5.40**	**+5.75**	**+6.12**	**+6.35**	**+6.57**	**+6.70**
Preq	1	no	no	no	no	no	no	no	+0.27	+0.46	+0.57
	2	no	no	no	no	+0.65	+1.01	+1.33	+1.60	+1.84	+2.03
	3	no	no	**+2.25**	**+3.82**	**+4.77**	**+5.22**	**+5.67**	**+5.95**	**+6.22**	**+6.38**

may indicate that as training data increases, more complex classifiers can reduce the bias resulting from the independence assumption, and hence, outperform NB.

However, the amount of improvement varies considerably for different k values, scores and domains as we further summarize:

- **balance domain**: since this domain contains strong dependencies between attributes, we can observe large jumps in accuracy when going from $k = 0$ to $k = 3$ for all the scores. The only exception is the MDL score, which was not capable of improving the performance of induced k-DBCs when k goes above 1. This suggests that the MDL could not find the existing dependencies in this domain, thus underfitting the data.

- **nursery domain**: we observe gains in accuracy when going from $k = 0$ to $k = 3$ and

Table 4.3: Significative gains of the predictive accuracy of k-DBCs with respect to the Naïve Bayes *per score* for the *nursery* dataset

		Learning Step (t), # training examples: $t \times 100$									
Score	k	5	10	20	30	40	50	60	80	100	128
	0	**86.96**	**87.52**	**88.62**	**89.04**	**89.22**	**89.34**	**89.52**	**89.66**	**89.97**	**89.96**
MDL	1	no	no	no	**+0.32**	**+0.51**	**+0.71**	**+0.77**	**+0.82**	**+0.84**	**+1.02**
	2	no	no	no	+0.32	+0.51	+0.71	+0.77	+0.82	+0.84	+1.02
	3	no	no	no	+0.32	+0.51	+0.71	+0.77	+0.82	+0.84	+1.02
	4	no	no	no	+0.32	+0.51	+0.71	+0.77	+0.82	+0.84	+1.02
	5	no	no	no	+0.32	+0.51	+0.71	+0.77	+0.82	+0.84	+1.02
AIC	1	no	**+0.54**	**+0.89**	**+1.46**	**+1.87**	**+2.19**	**+2.38**	**+2.70**	+2.80	+3.02
	2	no	+0.54	+0.89	+1.46	+1.87	+2.19	+2.38	+2.66	**+2.95**	**+3.23**
	3	no	+0.54	+0.89	+1.46	+1.87	+2.19	+2.38	+2.66	+2.95	+3.23
	4	no	+0.54	+0.89	+1.46	+1.87	+2.19	+2.38	+2.66	+2.95	+3.23
	5	no	+0.54	+0.89	+1.46	+1.87	+2.19	+2.38	+2.66	+2.95	+3.23
BD	1	no	no	+0.86	+1.21	+1.41	+1.68	+1.87	+2.08	+2.13	+2.42
	2	no	no	+0.80	+1.24	+1.38	+1.46	+1.53	+1.78	+1.96	+2.24
	3	no	no	+0.91	+1.31	+1.43	+1.50	+1.56	+1.67	+1.87	+2.32
	4	no	no	+0.93	+1.32	+1.44	**+1.51**	+1.57	**+1.81**	**+1.99**	**+2.32**
	5	no	no	**+0.94**	**+1.33**	+1.44	+1.51	**+1.58**	+1.67	+1.88	+2.32
BDeu	1	no	no	no	**+0.93**	**+1.12**	**+1.36**	**+1.51**	**+1.75**	+1.90	+2.18
	2	no	no	no	no	no	+0.45	+0.97	+1.64	**+2.00**	**+2.35**
	3	no	no	no	no	no	no	no	no	+0.95	+1.73
	4	no	no	no	no	no	no	no	no	no	+0.27
	5	no	no	no	no	no	no	no	no	no	no
MLC	1	no	no	**+0.87**	**+1.46**	**+1.91**	**+2.10**	**+2.25**	**+2.42**	+2.46	+2.67
	2	no	no	no	no	no	+0.87	+1.51	+2.37	**+2.86**	**+3.43**
	3	no	no	no	no	no	no	no	+0.79	+1.87	+2.95
	4	no	no	no	no	no	no	no	no	no	no
	5	no	no	no	no	no	no	no	no	no	no
CV	1	no	no	+2.11	+2.60	+2.88	+3.12	+3.30	+3.57	+3.67	+3.98
	2	no	**+1.46**	**+2.62**	+3.36	+3.77	+4.22	+4.41	+4.77	+4.94	+5.17
	3	no	+1.36	+2.58	+3.57	**+4.09**	**+4.59**	**+4.91**	**+5.32**	**+5.62**	**+6.05**
	4	no	+1.30	+2.49	**+3.59**	+4.05	+4.50	+4.79	+5.22	+5.50	+5.98
	5	no	+1.30	+2.49	+3.59	+4.04	+4.48	+4.81	+5.29	+5.59	**+6.07**
Preq	1	no	no	+1.59	+2.31	+2.72	+2.96	+3.13	+3.35	+3.46	+3.72
	2	no	no	+1.86	+2.69	+3.26	+3.75	+4.06	+4.50	+4.74	+5.05
	3	no	no	**+2.06**	**+3.14**	**+3.64**	**+4.10**	**+4.42**	**+4.99**	+5.28	+5.73
	4	no	no	+2.04	+3.09	+3.60	+4.11	+4.44	+5.03	**+5.35**	**+5.78**
	5	no	no	+2.07	+3.11	+3.61	+4.12	+4.45	+5.02	+5.33	+5.69

Table 4.4: Significative gains of the predictive accuracy of k-DBCs with respect to the Naïve Bayes *per score* for the *adult* dataset

		Learning Step (t), # training examples: $t \times 100$									
Score	k	5	10	20	40	60	80	100	120	140	160
	0	**81.00**	**81.14**	**81.91**	**81.69**	**81.89**	**82.00**	**81.99**	**82.01**	**82.04**	**82.16**
MDL	1	no	no	**+0.34**	**+1.36**	**+1.59**	**+1.66**	+1.74	+1.86	**+1.91**	**+1.87**
	2	no	no	+0.34	+1.36	+1.59	+1.66	**+1.75**	**+1.88**	+1.91	+1.87
	3	no	no	+0.34	+1.36	+1.59	+1.66	+1.75	+1.88	+1.91	+1.87
	4	no	no	+0.34	+1.36	+1.59	+1.66	+1.75	+1.88	+1.91	+1.87
AIC	1	no	**+1.06**	**+0.60**	**+1.71**	**+1.77**	**+1.78**	**+1.84**	+1.96	+1.98	+1.96
	2	no	+1.04	+0.55	+1.51	+1.65	+1.73	+1.82	**+1.99**	**+2.07**	**+2.10**
	3	no	+1.04	+0.55	+1.51	+1.65	+1.73	+1.82	+1.99	+2.07	+2.10
	4	no	+1.04	+0.55	+1.51	+1.65	+1.73	+1.82	+1.99	+2.07	+2.10
BD	1	**+1.56**	**+1.54**	**+0.83**	**+1.68**	+1.65	+1.67	+1.72	+1.84	+1.87	+1.84
	2	no	+1.18	+0.72	+1.62	+1.77	+1.84	**+1.94**	+2.11	**+2.13**	**+2.11**
	3	no	+1.20	+0.73	+1.63	**+1.78**	**+1.85**	+1.94	**+2.12**	+2.13	+2.11
	4	no	+1.20	+0.73	+1.63	+1.78	+1.85	+1.94	+2.12	+2.13	+2.11
BDeu	1	no	no	no	no	+0.36	+0.51	+0.70	+0.95	+1.05	+1.16
	2	no	no	no	no	no	no	+0.59	+0.83	+0.94	+1.03
	3	no	no	no	**+0.67**	**+0.69**	**+0.72**	**+1.05**	**+1.24**	**+1.28**	**+1.38**
	4	no	no	no	+0.67	+0.69	+0.72	+1.05	+1.24	+1.28	+1.38
MLC	1	no	no	**+0.15**	**+1.34**	**+1.46**	**+1.54**	**+1.65**	**+1.80**	**+1.86**	**+1.82**
	2	no	no	no	no	no	+0.59	+0.8	+1.08	+1.17	+1.26
	3	no	no	no	no	no	no	no	+0.48	+0.63	+0.75
	4	no	no	no	no	no	no	no	no	no	no
CV	1	no	**+0.72**	**+0.48**	+0.81	+0.98	+0.98	+1.00	+1.15	+1.18	+1.19
	2	no	no	no	+0.83	+1.10	+1.14	+1.22	+1.38	+1.43	+1.41
	3	no	+0.7	no	**+1.25**	**+1.38**	**+1.37**	+1.41	+1.55	+1.59	+1.60
	4	**+0.96**	no	no	+1.23	+1.37	+1.31	**+1.48**	**+1.71**	**+1.74**	**+1.75**
Preq	1	no	no	+0.92	+1.41	+1.35	+1.28	+1.31	+1.35	+1.37	+1.37
	2	**+1.12**	+1.44	**+1.06**	+1.67	+1.66	+1.54	+1.54	+1.67	+1.69	+1.68
	3	no	**+1.52**	+1.00	**+1.71**	+1.77	+1.67	+1.68	+1.79	+1.81	+1.78
	4	no	+1.16	+0.88	+1.71	**+1.88**	**+1.78**	**+1.79**	**+1.96**	**+1.96**	**+1.94**

at $t \geq 10$. The amount of improvement is greater for the supervised scores CV and Preq. MDL, AIC and BD, in general, were not able to obtain extra improvements for $k > 1$ (there are only two exceptions at the last two learning steps for AIC when small improvements for $k = 2$ are observed). The bad results obtained with BDeu and MLC evidence some overfitting, especially for more complex k-DBCs.

- **adult domain**: both AIC and BD show significant gains in the accuracy when going from $k = 0$ to $k = 2$. Note that both scores show the best improvements when compared with other scores. We also observe significant gains using the supervised scores CV and Preq, which increase when k goes from 1 to 4. However, in this domain supervised scores cannot outperform the unsupervised AIC and BD. On the other hand, MLC and BDeu do not show significant improvements with the increasing of the k value. Specially at earlier learning steps and for greater k values the results eventually evidence overfitting. However, note that as the learning process advances (at $t \geq 20$) BDeu starts showing significant gains in the accuracy when a 3-DBC is used. For MLC and MDL we observe no extra improvements when k goes above 1.

This comparative study in some way is related to the study of k-DBCs performed by Sahami in his original work [121]. As shown by Sahami this class of k-DBCs is very useful for experimental purposes. By knowing how the classification performance changes with the value of k we can get a notion of the degree of attribute dependence in each particular domain. In his experiments Sahami observed that for some domains starting from some k value there is no more improvements in the accuracy, which may indicate that there is lower degree of dependence in these domains. Unlike Sahami's experiments which are based on a batch learning approach, we have compared the performance of k-DBCs varying k for different scores and different amount of training data in a prequential learning framework.

4.3.2 Comparing the Performance of k-DBCs per Class-Model

Tables 4.6, 4.7 and 4.8 show another view of the results about the performance of different k-DBCs, but unlike the previous Tables 4.2, 4.3 and 4.4, we have now grouped the results by class-model according to three different k values, $k = 1, 2, 3$. For each k value we have arranged the scores in order of their bias toward simplicity: MDL, BD, AIC, Preq, CV, MLC and BDeu. A sign "+" indicates significant gains over NB using a paired t-test at the 5% level. For comparison purposes, at each time point we provide a rank for each score associated to each k value. We assigned rank 1 to the score that provides more significant gains, rank 2 to the second best score, and so on. Superscripts on each entry denote theses ranks. The last column shows the average rank of each score. To get evidence about the complexity

of induced classifiers, Tables 4.9, 4.10 and 4.11 show the number of arcs added to the NB structure (averaged over 10 runs) for different scores at different time points also grouped by class-model. Superscripts in the column "Max" for *balance* denote the learning step at which the maximum number of additions is accomplished. The six last columns in each table summarize some statistics of the number of arcs added to NB and the number of parameters of the induced k-DBCs for each k value and score. The columns "Min","Max" and "Avg" depict the *minimum*, *maximum* and *average* number of arc additions or number of parameters, respectively. The maximum number of allowable additions[1] are reported with bold text and summarized in Table 4.5.

Table 4.5: Maximum number of allowable additions for each domain

Dataset	#Attributes	1-DBC	2-DBC	3-DBC
balance	4	3	5	6
nursery	8	7	13	18
adult	14	13	25	36

By using these comparative tables now grouped by class-model (that is, for different k values) we can analyze how each score handles the *complexity-performance* trade-off in the prequential framework for learning k-DBCs. We can make the following observations for each domain:

- **balance domain**: For a 1-DBC those scores more biased toward simpler models, such as MDL, BD and AIC do, show better performance than those scores more biased toward complex models, which can lead eventually to overfitting. In general, when $k = 1$ only few k-DBCs can outperform NB. In most cases, we start observing significant gains only at $t > 50$ where there are more training data. This situation radically changes when going from $k = 1$ to $k = 3$. At $t > 10$ all the scores more biased towards complexity (BDeu, MLC, CV and Preq) show large jumps in the accuracy. In addition, results from Table 4.10 evidence that they found maximal structures, which represent the existing strong degree of attribute dependence in the balance domain. The best results were obtained with a 3-DBC and the CV score.

[1]The maximum number of allowable additions is obtained from the formula $(n-1)+(n-2)++(n-k)$ where n is the number of attributes.

Table 4.6: Gains of the predictive accuracy of k-DBCs with respect to the Naïve Bayes *per class-model* for the *balance* dataset

		Learning Step									Avg.
k	Score	1	5	10	30	50	70	80	90	100	Rank
	0	**79.40**	**85.70**	**87.80**	**90.17**	**90.63**	**91.05**	**91.17**	**91.22**	**91.30**	
1	MDL	0.00[1]	0.00[1]	0.00[1]	0.00[1]	**+0.19**[1]	**+0.44**[1]	**+0.55**[1]	**+0.66**[1]	**+0.72**[1]	**1.0[1]**
	BD	-1.80[3]	-0.86[2]	-0.81[2]	-0.34[2]	-0.04[2]	0.22[2]	+0.36[2]	+0.51[2]	+0.58[2]	**2.0[2]**
	AIC	0.00[1]	-1.46[3]	-1.46[3]	-0.71[3]	-0.24[3]	0.12[3]	+0.28[3]	+0.42[4]	+0.50[4]	**3.2[3]**
	Preq	-3.30[4]	-3.36[4]	-2.77[4]	-0.97[4]	-0.31[4]	0.09[4]	+0.27[4]	+0.46[3]	+0.57[3]	3.8[4]
	CV	-3.70[5]	-4.86[5]	-3.65[6]	-1.37[6]	-0.58[5]	-0.08[5]	0.11[5]	+0.30[5]	+0.40[5]	5.2[5]
	MLC	-9.80[7]	-5.36[6]	-3.37[5]	-1.35[5]	-0.62[6]	-0.15[6]	0.05[6]	0.21[6]	0.31[6]	5.8[6]
	BDeu	-9.40[6]	-5.74[7]	-3.69[7]	-1.37[6]	-0.63[7]	-0.20[7]	0.02[7]	0.19[7]	+0.29[7]	6.9[7]
2	MDL	0.00[1]	0.00[1]	0.00[1]	0.00[2]	+0.19[6]	+0.44[7]	+0.55[7]	+0.66[7]	+0.72[7]	5.4[6]
	BD	-1.90[3]	-0.86[2]	-0.82[2]	0.08[1]	**+1.10**[1]	**+1.63**[1]	**+1.81**[1]	**+2.02**[1]	**+2.19**[3]	**1.3[1]**
	AIC	0.00[1]	-1.46[3]	-1.46[3]	-0.06[3]	+0.98[2]	+1.58[2]	+1.81[1]	+2.01[3]	+2.20[2]	**2.2[2]**
	Preq	-3.30[4]	-3.36[4]	-2.77[4]	-0.53[4]	+0.65[4]	+1.33[4]	+1.60[4]	+1.84[4]	+2.03[4]	4.0[4]
	CV	-3.70[5]	-4.94[5]	-3.71[5]	-0.59[5]	+0.78[3]	+1.53[3]	+1.78[3]	**+2.02**[1]	**+2.22**[1]	**3.0[3]**
	MLC	-12.00[7]	-8.56[7]	-5.71[7]	-1.46[7]	0.14[7]	+0.98[6]	+1.28[6]	+1.54[6]	+1.77[6]	6.4[7]
	BDeu	-9.80[6]	-7.96[6]	-5.42[6]	-1.32[6]	0.25[5]	+1.06[5]	+1.35[5]	+1.63[5]	+1.86[5]	5.3[5]
3	MDL	0.00[1]	0.00[1]	0.00[1]	0.00[6]	+0.19[7]	+0.44[7]	+0.55[7]	+0.66[7]	+0.72[7]	6.2[7]
	BD	-1.80[3]	-3.36[4]	-0.81[2]	+1.99[5]	+4.62[5]	+5.56[5]	+5.86[5]	+6.14[5]	+6.30[5]	4.7[5]
	AIC	0.00[1]	-1.46[2]	-1.46[3]	-0.06[7]	+1.04[6]	+2.98[6]	+3.60[6]	+4.13[6]	+4.48[6]	5.8[6]
	Preq	-3.30[4]	-3.30[3]	-2.77[4]	+2.25[4]	+4.77[4]	+5.67[4]	+5.95[4]	+6.22[4]	+6.38[4]	4.0[4]
	CV	-3.70[5]	-4.94[5]	-3.44[5]	**+3.29**[1]	**+5.40**[1]	**+6.12**[1]	**+6.35**[1]	**+6.57**[1]	**+6.70**[1]	**1.4[1]**
	MLC	-10.40[6]	-6.98[6]	-3.86[6]	+3.15[2]	+5.33[2]	+6.07[2]	+6.30[2]	+6.53[2]	+6.66[2]	**2.4[2]**
	BDeu	-12.50[7]	-8.22[7]	-3.98[7]	+3.14[3]	+5.31[3]	+6.06[3]	+6.29[3]	+6.52[3]	+6.65[3]	**3.4[3]**

- **nursery domain**: CV and Preq performed a more optimal *performance-complexity* trade-off over time. As training data increases we observe gains in accuracy moving from $k = 1$ to $k = 3$. Both scores found models of intermediate complexity but more complex for more training data. A similar behavior is obtained with BD and AIC. Although both scores are more biased toward models of intermediate to simple complexity they also show a good performance, far better as data increases. The worse results were obtained with those scores that lie in more extreme situations: MDL (biased for simpler models) and BDeu and MLC (biased for more complex models). Results from Table 4.10 evidence that MDL underfits the data, as shown by the small number of arcs added to NB. On the other hand, the k-DBCs induced with MDL show only modest gains over NB and only at $t > 40$. Moreover, we do not observe changes in the performance using MDL

Table 4.7: Gains of the predictive accuracy of k-DBCs with respect with respect to the Naïve Bayes *per class-model* for the *nursery* dataset

		Learning Step									Avg.
k	Score	1	5	10	20	40	60	80	100	120	Rank
	0	**82.80**	**86.96**	**87.52**	**88.62**	**89.22**	**89.52**	**89.66**	**89.87**	**89.98**	
1	MDL	0.00^{1}	0.00^{1}	0.00^{5}	0.00^{7}	$+0.51^{7}$	$+0.77^{7}$	$+0.82^{7}$	$+0.84^{7}$	$+0.93^{7}$	5.4^{6}
	BD	-0.60^{3}	-0.80^{4}	0.22^{3}	$+0.86^{5}$	$+1.41^{5}$	$+1.87^{5}$	$+2.08^{5}$	$+2.13^{5}$	$+2.28^{5}$	4.4^{4}
	AIC	0.00^{1}	-0.44^{3}	$+0.54^{2}$	$+0.89^{3}$	$+1.87^{4}$	$+2.38^{3}$	$+2.70^{3}$	$+2.80^{3}$	$+2.90^{3}$	$\mathbf{2.9^{3}}$
	Preq	-3.40^{5}	-1.92^{5}	0.14^{4}	$+1.59^{2}$	$\mathbf{+2.72^{2}}$	$\mathbf{+3.13^{2}}$	$\mathbf{+3.35^{2}}$	$\mathbf{+3.46^{2}}$	$\mathbf{+3.60^{2}}$	$\mathbf{2.8^{2}}$
	CV	-2.80^{4}	-0.12^{2}	$\mathbf{+1.06^{1}}$	$\mathbf{+2.11^{1}}$	$\mathbf{+2.88^{1}}$	$\mathbf{+3.30^{1}}$	$\mathbf{+3.57^{1}}$	$\mathbf{+3.67^{1}}$	$\mathbf{+3.84^{1}}$	$\mathbf{1.4^{1}}$
	MLC	-10.60^{7}	-3.16^{7}	-0.70^{6}	$+0.87^{4}$	$+1.91^{3}$	$+2.25^{4}$	$+2.42^{4}$	$+2.46^{4}$	$+2.58^{4}$	4.8^{5}
	BDeu	-8.20^{6}	-2.44^{6}	-1.02^{7}	0.25^{6}	$+1.12^{6}$	$+1.51^{6}$	$+1.75^{6}$	$+1.90^{6}$	$+2.06^{6}$	6.1^{7}
2	MDL	0.00^{1}	0.00^{1}	0.00^{4}	0.00^{5}	$+0.51^{5}$	$+0.77^{7}$	$+0.82^{7}$	$+0.84^{7}$	$+0.93^{7}$	4.9^{5}
	BD	-3.00^{4}	-1.32^{4}	-0.02^{5}	$+0.80^{4}$	$+1.38^{4}$	$+1.53^{4}$	$+1.78^{5}$	$+1.96^{6}$	$+2.13^{6}$	4.7^{4}
	AIC	0.00^{1}	-0.44^{3}	$+0.54^{2}$	$+0.89^{3}$	$+1.87^{3}$	$+2.39^{3}$	$+2.66^{3}$	$+2.95^{3}$	$+3.13^{4}$	$\mathbf{2.9^{3}}$
	Preq	-3.60^{5}	-1.44^{5}	0.18^{3}	$+1.86^{2}$	$\mathbf{+3.26^{2}}$	$\mathbf{+4.06^{2}}$	$\mathbf{+4.50^{2}}$	$\mathbf{+4.74^{2}}$	$\mathbf{+4.93^{2}}$	$\mathbf{2.8^{2}}$
	CV	-2.00^{3}	-0.40^{2}	$\mathbf{+1.46^{1}}$	$\mathbf{+2.62^{1}}$	$\mathbf{+3.77^{1}}$	$\mathbf{+4.41^{1}}$	$\mathbf{+4.77^{1}}$	$\mathbf{+4.94^{1}}$	$\mathbf{+5.09^{1}}$	$\mathbf{1.3^{1}}$
	MLC	-27.40^{6}	-15.88^{6}	-8.84^{6}	-3.58^{6}	-0.09^{6}	$+1.51^{5}$	$+2.37^{4}$	$+2.86^{4}$	$+3.25^{3}$	5.1^{6}
	BDeu	-31.80^{7}	-18.60^{7}	-9.98^{7}	-4.05^{7}	-0.45^{7}	$+0.97^{6}$	$+1.64^{6}$	$+2.00^{5}$	$+2.23^{5}$	6.3^{7}
3	MDL	0.00^{1}	0.00^{1}	0.00^{5}	0.00^{5}	$+0.51^{5}$	$+0.77^{5}$	$+0.82^{5}$	$+0.84^{7}$	$+0.93^{7}$	4.6^{5}
	BD	-1.00^{3}	-0.88^{4}	0.20^{4}	$+0.91^{3}$	$+1.43^{4}$	$+1.56^{4}$	$+1.81^{4}$	$+1.98^{4}$	$+2.17^{5}$	3.9^{4}
	AIC	0.00^{1}	-0.44^{3}	$+0.54^{2}$	$+0.89^{4}$	$+1.87^{3}$	$+2.39^{3}$	$+2.66^{3}$	$+2.95^{3}$	$+3.13^{3}$	$\mathbf{2.9^{3}}$
	Preq	-2.20^{5}	-1.16^{5}	0.32^{3}	$\mathbf{+2.06^{2}}$	$\mathbf{+3.64^{2}}$	$\mathbf{+4.42^{2}}$	$\mathbf{+4.99^{2}}$	$\mathbf{+5.28^{2}}$	$\mathbf{+5.55^{2}}$	$\mathbf{2.8^{2}}$
	CV	-1.20^{4}	-0.12^{2}	$\mathbf{+1.36^{1}}$	$\mathbf{+2.58^{1}}$	$\mathbf{+4.09^{1}}$	$\mathbf{+4.91^{1}}$	$\mathbf{+5.32^{1}}$	$\mathbf{+5.62^{1}}$	$\mathbf{+5.89^{1}}$	$\mathbf{1.4^{1}}$
	MLC	-42.80^{7}	-32.60^{7}	-20.38^{7}	-11.51^{7}	-4.07^{7}	-1.28^{7}	0.14^{7}	$+0.95^{6}$	$+1.50^{6}$	6.8^{7}
	BDeu	-17.00^{6}	-20.28^{6}	-17.30^{6}	-10.26^{6}	-3.77^{6}	-0.87^{6}	0.79^{6}	$+1.87^{5}$	$+2.66^{4}$	5.7^{6}

when the search space is increased (that is, when k is increased from 1 to 3). The results obtained with BDeu and MLC, on the contrary, evidence that overfitting takes place, a situation more pronounced at earlier learning steps and for more complex class-models. However, note that for $k = 1$ & $t \geq 20$; $k = 2$ & $t \geq 80$ and $k = 3$ & $t = 120$, the MLC score shows an acceptable performance when compared with the other scores.

- **adult domain**: BD and AIC performed a more optimal *performance-complexity* trade-off over time. Both scores show similar gains in performance for all the induced k-DBCs. However, there are no more improvements in the accuracy when k goes above 2. Results from Table 4.11 indicate that both BD and AIC found models of intermediate complexity. MDL found the simplest structures while BDeu and MLC found the most

Table 4.8: Gains of the predictive accuracy of k-DBCs with respect to the Naïve Bayes *per class-model* for the *adult* dataset

k	Score	Learning Step 1	5	10	20	40	60	100	120	160	Avg. Rank
	0	**83.80**	**81.00**	**81.14**	**81.91**	**81.69**	**81.89**	**81.99**	**82.01**	**82.16**	
1	MDL	0.00^{2}	0.28^{5}	0.48^{5}	+0.34^{5}	+1.36^{4}	+1.58^{3}	+1.74^{2}	+1.86^{2}	+1.87^{2}	**3.5**^{3}
	BD	-1.00^{5}	**+1.56**^{1}	**+1.54**^{1}	+0.83^{2}	+1.68^{2}	+1.65^{2}	+1.72^{3}	+1.84^{3}	+1.84^{3}	**2.1**^{2}
	AIC	-0.60^{4}	0.92^{2}	+1.06^{2}	+0.60^{3}	**+1.71**^{1}	**+1.77**^{1}	**+1.84**^{1}	**+1.96**^{1}	**+1.96**^{1}	**1.5**^{1}
	Preq	0.40^{1}	0.44^{4}	+1.06^{2}	**+0.92**^{1}	+1.42^{3}	+1.35^{4}	+1.31^{5}	+1.35^{5}	+1.37^{5}	3.8^{4}
	CV	-1.40^{6}	0.56^{3}	+0.72^{3}	+0.48^{4}	+0.81^{6}	+0.98^{6}	+1.00^{6}	+1.15^{6}	+1.19^{6}	5.1^{6}
	MLC	-3.00^{7}	-1.08^{7}	0.18^{6}	+0.15^{6}	+1.34^{5}	+1.46^{5}	+1.65^{4}	+1.80^{4}	+1.82^{4}	5.0^{5}
	BDeu	-0.10^{3}	0.08^{6}	0.10^{7}	-0.06^{7}	0.09^{7}	+0.36^{7}	+0.70^{7}	+0.95^{7}	+1.16^{7}	6.9^{7}
2	MDL	0.00^{1}	0.28^{5}	0.48^{5}	+0.34^{4}	+1.36^{4}	+1.59^{4}	+1.75^{3}	+1.88^{3}	+1.87^{3}	3.9^{4}
	BD	-1.20^{4}	0.96^{2}	+1.18^{2}	+0.72^{2}	+1.62^{2}	+1.77^{1}	**+1.94**^{1}	**+2.11**^{1}	**+2.11**^{1}	**1.4**^{1}
	AIC	-0.60^{2}	0.92^{3}	+1.04^{3}	+0.55^{3}	+1.51^{3}	+1.65^{3}	**+1.82**^{2}	**+1.99**^{2}	**+2.10**^{2}	**2.6**^{3}
	Preq	-1.00^{3}	**+1.12**^{1}	**+1.44**^{1}	**+1.06**^{1}	**+1.67**^{1}	+1.66^{2}	+1.54^{4}	+1.67^{4}	+1.68^{4}	**2.3**^{2}
	CV	-2.00^{6}	0.60^{4}	0.52^{4}	0.31^{5}	+0.83^{5}	+1.10^{5}	+1.22^{5}	+1.38^{5}	+1.41^{5}	4.8^{5}
	MLC	-5.00^{7}	-1.08^{6}	-0.86^{7}	-1.29^{7}	0.15^{6}	0.40^{6}	+0.80^{6}	+1.08^{6}	+1.26^{6}	6.3^{6}
	BDeu	-1.32^{5}	-1.32^{7}	-0.72^{6}	-1.04^{6}	-0.42^{7}	-0.04^{7}	+0.59^{7}	+0.83^{7}	+1.03^{7}	6.8^{7}
3	MDL	0.00^{1}	0.28^{5}	0.48^{5}	+0.34^{4}	+1.36^{4}	+1.59^{4}	+1.75^{3}	+1.88^{3}	+1.87^{3}	4.0^{4}
	BD	-0.60^{3}	**+1.12**^{1}	+1.20^{2}	+0.73^{2}	+1.63^{2}	**+1.78**^{1}	**+1.94**^{1}	**+2.12**^{1}	**+2.11**^{1}	**1.4**^{1}
	AIC	-0.60^{3}	0.92^{3}	+1.04^{3}	+0.55^{3}	+1.51^{3}	+1.65^{3}	**+1.82**^{2}	**+1.99**^{2}	**+2.10**^{2}	**2.6**^{3}
	Preq	-0.40^{2}	1.00^{2}	**+1.52**^{1}	**+1.00**^{1}	**+1.71**^{1}	+1.77^{2}	+1.68^{4}	+1.79^{4}	+1.78^{4}	**2.4**^{2}
	CV	-2.40^{5}	0.56^{4}	+0.70^{4}	0.53^{4}	+1.25^{5}	+1.38^{5}	+1.41^{5}	+1.55^{5}	+1.60^{5}	4.7^{5}
	MLC	-2.60^{6}	-1.08^{7}	-1.54^{7}	-1.95^{7}	-0.54^{7}	-0.19^{7}	0.19^{7}	+0.48^{7}	+0.75^{7}	7.0^{7}
	BDeu	-3.20^{7}	-0.20^{6}	0.20^{6}	-0.28^{6}	+0.67^{6}	+0.69^{6}	+1.05^{6}	+1.24^{6}	+1.38^{6}	6.0^{6}

complex ones, thus severely overfiting. Note that CV and Preq also found models more complex than those induced with BD and AIC, and even CV overfitting a little. Preq, instead, shows an acceptable performance over time. At the first learning steps ($t \leq 40$) and for ($k = 2, 3$) Preq outperformed BD and AIC, but at $t > 40$ this situation is reverted in favor of MDL, AIC and BD. These results evidence how some amount of additional training data can affect the performance of a particular score.

All the results show that for all the domains and scores there is practically no k-DBC that significantly outperforms NB for small data. Only from 1000 training examples do we observe improvements over NB, which becomes more significant with more training data. The *balance* domain contains strong interactions between the attributes. Therefore, 3-DBCs induced with scores more biased towards complexity, such as BDeu, MLC, CV and Preq,

Table 4.9: Number of arcs added to the NB's structure and number of parameters for the *balance* dataset

		Learning Step									**# Arc Additions**			**# Parameters**		
k	Score	1	5	10	30	50	70	80	90	100	Min	Max	Avg	Min	Max	Avg
1	MDL	0	0	0	0	**3**	**3**	**3**	**3**	**3**	0	$\mathbf{3}^{(49)}$	1.7	50	194	133.7
	BD	0.4	0.1	1.3	**3**	**3**	**3**	**3**	**3**	**3**	0.1	$\mathbf{3}^{(18)}$	2.2	55	194	178.4
	AIC	0.0	1.4	**3**	**3**	**3**	**3**	**3**	**3**	**3**	0	$\mathbf{3}^{(8)}$	2.5	50	194	187.1
	Preq	0.6	2.7	**3**	**3**	**3**	**3**	**3**	**3**	**3**	0.6	$\mathbf{3}^{(6)}$	2.7	78.8	194	190.4
	CV	1.1	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	1.1	$\mathbf{3}^{(3)}$	2.8	102.8	194	192.7
	MLC	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	$\mathbf{3}^{(1)}$	**3**	194	194	194
	BDeu	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	$\mathbf{3}^{(1)}$	**3**	194	194	194
2	MDL	0	0	0	0	3	3	3	3	3.2	0	$3.2^{(100)}$	1.7	50	242	134.7
	BD	0.6	0.1	1.3	**5**	**5**	**5**	**5**	**5**	**5**	0.1	$\mathbf{5}^{(30)}$	3.6	54.8	674	551.3
	AIC	0	1.4	3	**5**	**5**	**5**	**5**	**5**	**5**	0	$\mathbf{5}^{(23)}$	3.8	50	674	578.2
	Preq	0.6	2.7	3	**5**	**5**	**5**	**5**	**5**	**5**	0.6	$\mathbf{5}^{(21)}$	4.0	78.8	674	597.9
	CV	1.1	3.1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	1.1	$\mathbf{5}^{(10)}$	4.4	102.8	674	639.7
	MLC	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	$\mathbf{5}^{(1)}$	**5**	674	674	674
	BDeu	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	$\mathbf{5}^{(1)}$	**5**	674	674	674
3	MDL	0	0	0	0	3	3	3	3	3.2	0	$3.2^{(100)}$	1.7	50	242	134.7
	BD	0.7	0.1	1.3	**6**	**6**	**6**	**6**	**6**	**6**	0.1	$\mathbf{6}^{(30)}$	4.2	54.8	1874	1503.9
	AIC	0	1.4	3	5	5.4	**6**	**6**	**6**	**6**	0	$\mathbf{6}^{(53)}$	4.3	50	1874	1173.3
	Preq	0.6	2.7	3	**6**	**6**	**6**	**6**	**6**	**6**	0.6	$\mathbf{6}^{(21)}$	4.7	78.8	1874	1609
	CV	1.1	3.1	5.6	**6**	**6**	**6**	**6**	**6**	**6**	1.1	$\mathbf{6}^{(12)}$	5.0	102.8	1874	1729.5
	MLC	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	$\mathbf{6}^{(1)}$	**6**	1874	1874	1874
	BDeu	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	$\mathbf{6}^{(1)}$	**6**	1874	1874	1874

produce more accurate k-DBCs. For the *nursery* domain 3-DBCs induced with Preq and CV perform better. Since the best jumps in accuracy were obtained going from $k = 0$ until $k = 3$, we can suspect that there are about 3-order attribute dependencies in the *nursery* domain. For the *adult domain* 2-DBCs induced with scores more biased toward models of intermediate to simple complexity, such as Preq, BD, AIC and MDL, produce the best classifiers. This may indicate that there are about 2-order attribute dependencies in the *adult* domain.

Table 4.10: Number of arcs added to the NB's structure and number of parameters for the *nursery* dataset.

		Learning Step									**# Arc Additions**			**# Parameters**		
k	Score	1	5	10	20	40	60	80	100	120	Min	Max	Avg	Min	Max	Avg
1	MDL	0.0	0.0	0.0	0.0	1.0	1.0	1.2	1.4	2.0	0.0	2.0	1.0	99	149	134
	BD	3.0	1.4	2.0	3.0	3.2	4.0	4.0	4.2	5.4	0.8	5.6	3.6	125	500	269
	AIC	0.0	0.8	1.4	2.6	5.8	6.2	6.6	**7.0**	**7.0**	0.0	**7.0**	5.3	99	589	304
	Preq	4.8	4.4	6.4	6.6	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	3.2	**7.0**	6.8	182	937	556
	CV	3.6	6.6	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	3.6	**7.0**	6.9	239	959	609
	MLC	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	340	1235	752
	BDeu	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	339	1219	672
2	MDL	0.0	0.0	0.0	0.0	1.0	1.0	1.2	1.4	2.0	0.0	2.0	1.0	99	149	134
	BD	5.2	1.6	2.0	4.0	4.2	5.4	6.0	6.2	7.8	0.8	7.8	5.0	125	500	352
	AIC	0.0	0.8	1.4	2.6	5.8	6.2	8.8	9.4	9.8	0.0	10.0	6.4	99	589	357
	Preq	7.4	5.2	8.0	10.0	11.8	12.4	12.8	12.6	**13.0**	3.6	**13.0**	11.6	227	937	789
	CV	5.2	9.8	12.4	12.6	12.8	**13.0**	**13.0**	**13.0**	**13.0**	5.2	**13.0**	12.6	295	959	891
	MLC	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	1046	1235	1143
	BDeu	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	899	1219	975
3	MDL	0.0	0.0	0.0	0.0	1.0	1.0	1.2	1.4	2.0	0.0	2.0	1.0	99	149	133
	BD	7.0	1.6	2.0	4.0	4.2	5.4	6.0	6.4	8.8	0.8	8.8	5.2	125	1208	402
	AIC	0.0	0.8	1.4	2.6	5.8	6.2	8.8	9.4	9.8	0.0	10.0	6.4	99	589	355
	Preq	8.0	5.4	8.0	11.6	13.8	14.8	15.6	17.4	17.8	3.6	17.8	14.1	227	2174	1422
	CV	5.8	10.0	14.2	15.8	16.8	**18.0**	**18.0**	**18.0**	**18.0**	5.4	**18.0**	16.7	295	2924	2193
	MLC	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	2503	3595	2989
	BDeu	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	**18.0**	2498	3459	2817

Table 4.11: Number of arcs added to the NB's structure and number of parameters for the *adult* dataset. The number of parameters is expressed in (1×10^3) units.

		Learning Step									**# Arc Additions**			**# Parameters**		
k	Score	1	5	10	20	40	60	100	120	160	Min	Max	Avg	Min	Max	Avg
1	MDL	1.0	3.2	5.8	6.0	7.6	8.0	9.6	10.4	11.0	1.0	11.0	8.6	0.3	1.8	1.2
	BD	8.8	7.2	7.6	10.2	10.8	11.0	11.0	11.0	12.0	6.4	12.0	10.8	1.2	4.6	2.6
	AIC	3.6	8.2	9.8	11.2	12.0	12.6	12.8	12.8	**13.0**	3.6	**13.0**	12.2	0.4	3.2	2.1
	Preq	11.8	12.6	**13.0**	**13.0**	12.8	12.4	12.4	**13.0**	12.4	11.8	**13.0**	12.7	1.9	18.2	8.2
	CV	12.8	12.8	12.8	12.2	12.8	**13.0**	12.6	12.8	**13.0**	11.8	**13.0**	12.8	1.8	28.1	10.7
	MLC	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	2.9	41.4	18.7
	BDeu	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	**13.0**	4.4	81.1	39.0
2	MDL	1.0	3.2	5.8	6.0	7.6	8.0	9.8	10.6	11.8	1.0	13.0	8.8	0.3	1.8	1.2
	BD	11.8	8.4	10.0	12.4	15.4	16.6	17.4	17.8	18.0	7.6	18.0	16.1	1.4	4.7	3.2
	AIC	3.6	8.2	9.8	12.0	15.4	17.8	19.0	19.4	20.4	3.6	20.4	16.8	0.4	3.2	2.3
	Preq	18.2	20.2	20.6	21.2	22.4	21.2	22.8	21.4	21.8	18.2	23.2	21.8	6.9	18.2	14.0
	CV	22.4	22.2	22.4	20.8	22.6	23.0	22.4	22.2	21.8	20.0	23.6	22.4	8.3	28.1	19.0
	MLC	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.5	41.3	34.0
	BDeu	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	**25.0**	66.4	81.1	72.5
3	MDL	1.0	3.2	5.8	6.0	7.6	8.0	9.8	10.6	11.8	1.0	13.0	8.8	0.3	1.8	1.2
	BD	12.8	8.6	10.0	12.4	15.4	16.6	17.4	17.8	18.0	7.8	18.0	16.1	1.4	4.7	3.2
	AIC	3.6	8.2	9.8	12.0	15.4	17.8	19.0	19.4	20.4	3.6	20.4	16.8	0.3	3.2	2.3
	Preq	22.2	25.6	26.4	27.6	28.0	28.0	29.6	29.4	29.2	22.2	30.6	28.0	19.7	76.5	52.1
	CV	27.4	27.4	26.2	26.4	29.2	29.0	29.2	29.6	29.2	25.0	32.0	29.0	21.2	205	94.8
	MLC	34.0	34.0	34.0	34.0	34.0	34.0	34.0	34.0	34.0	34.0	34.0	34.0	160	305	226
	BDeu	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**	765	879	792

4.3.3 Comparing the Bias-Variance Trade-off per Score

There is a bias-variance trade-off in choosing the appropriate complexity of the model. We further performed a case study with the *nursery* dataset in order to investigate how different scores handle the *bias-variance* trade-off of several k-DBCs induced in the prequential framework. We compare the performance of k-DBCs varying k from 0 to 7, the maximum number of allowable attribute dependencies in this particular domain. We chose for this study only four scores among those that should bring us more interesting behaviours for discussion. These scores arranged in order of their bias toward simplicity are: MDL, AIC, Preq and BDeu. Figures 4.1 and 4.2 allow us to identify the differences in how each score handles the bias-variance trade-off in learning k-DBCs at six time points: $t = 5$, $t = 10$, $t = 15$, $t = 50$, $t = 120$ and $t = 125$. Plots on the left show the training and test errors as a function of k-DBC class-model. Each point in a line represents the *training* or *test* error of the related k-DBC for a particular score. The first points in the lines represent the errors of NB ($k = 0$). Pictures on the right show the *bias-variance* decomposition of the test error for all the scores.

Bias-Variance Trade-off

The bias-variance decomposition of the error is a useful tool to understand the behavior of learning algorithms. We performed the bias-variance decomposition of the test error at different time points. To estimate the bias and variance we introduce some modifications to the methodology proposed in [73] and described in Section 3.2.3. For each dataset we used the ten randomly generated samples as *training samples* and then randomly generated a new, 11^{th} sample, to serve as *test sample*. We split all the training and test samples into batches of 100 examples. At the t^{th} learning step we used the first t batches of each training sample as *training sets* and the corresponding t batch of examples of the test sample as *test set*. At each time point we used the Algorithm 7 for learning the k-DBCs using each *training set*. Then we used each induced k-DBC to predict the class for each example in the *test set* and to update the corresponding frequency counters from which the bias and the variance terms are estimated.

Increasing data set size affects the bias-variance error decomposition for supervised learning algorithms. It is well known that variance decreases as training set size increases. How-

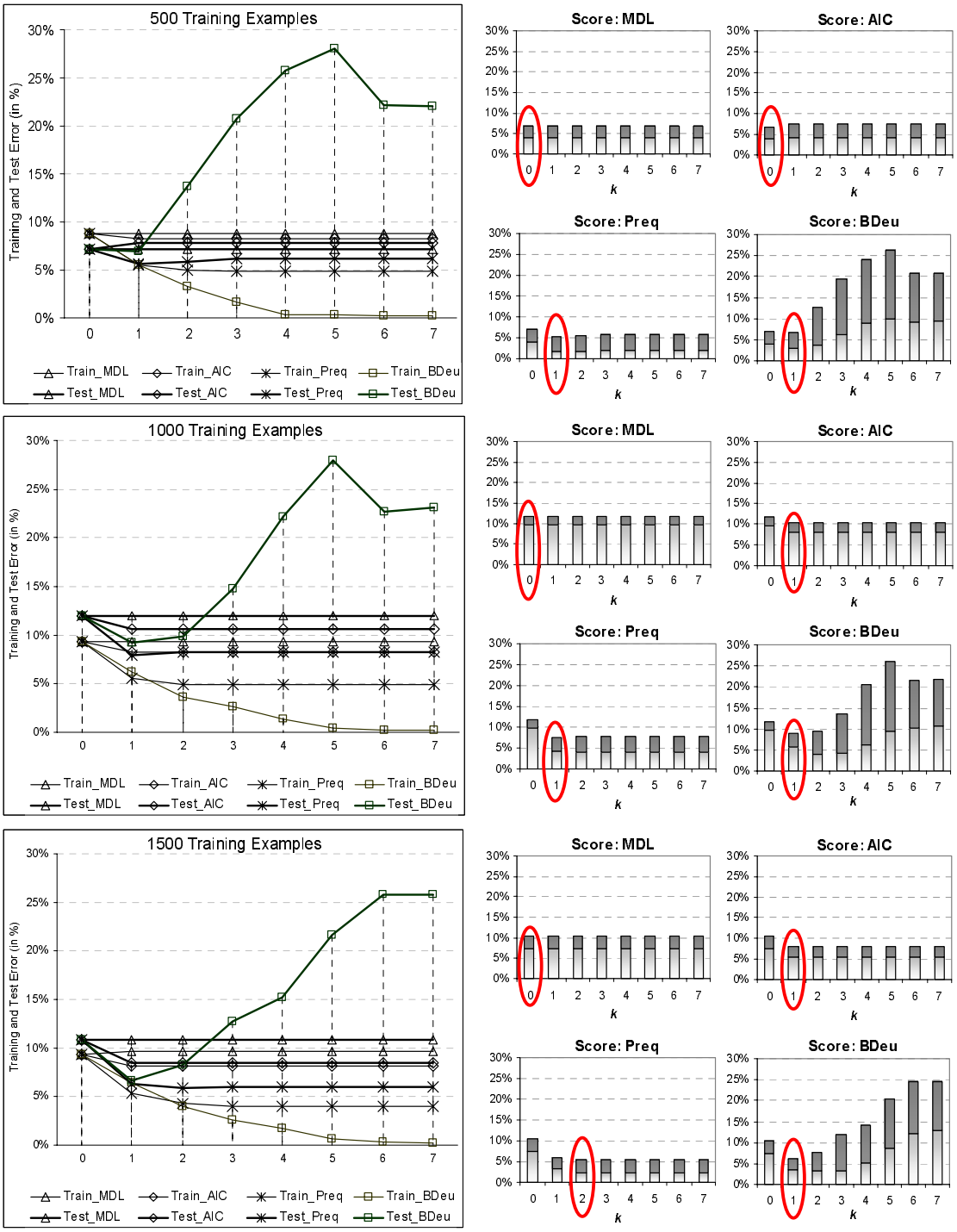

Figure 4.1: Training-test errors and bias-variance decomposition of the test error varying k at three time points: $t = 5$, $t = 10$ and $t = 15$

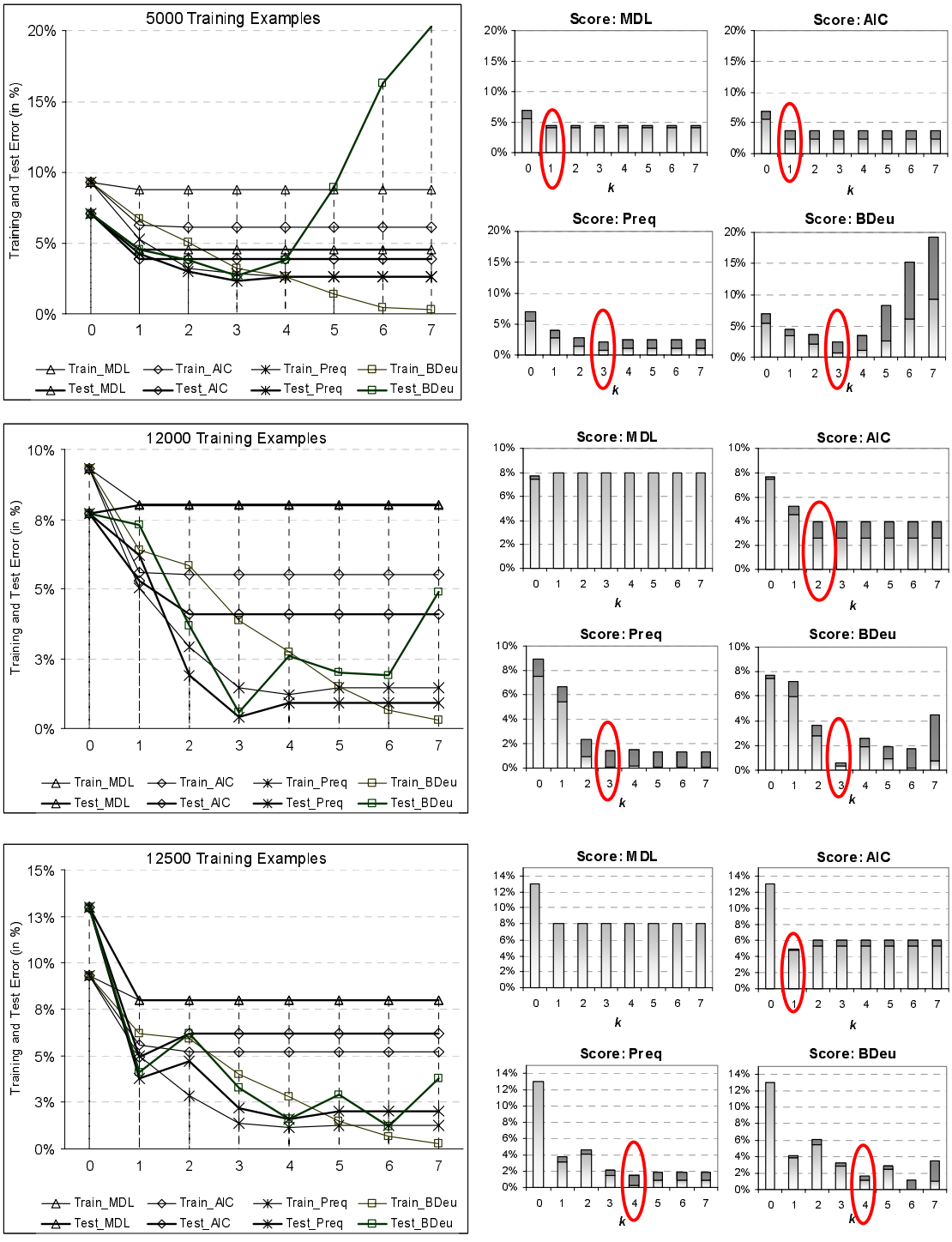

Figure 4.2: Training-test errors and bias-variance decomposition of of the test error varying k at three time points: $t = 50$, $t = 120$ and $t = 125$

ever, as shown in [13] there is no clear effect of the training set size on the bias component. By looking at Figures 4.1 and 4.2 we can analyze the differences in the *bias-variance* decomposition of the test error for different scores across the range of k-DBC class-models at different time points, that is, for different amount of training data. In addition, to get extra evidences about the differences in the complexity of k-DBCs induced with different scores Table 4.14 shows the averaged number of arcs added to the NB structure for each score and k value at different time points. An entry with (") means that the number of added arcs for the corresponding k-DBC is the same number shown in the previous line. The column "Allow." shows the maximum number of allowable arc's additions for each k-DBC class-model. Since BDeu always finds maximal models we do not present the number of added arcs for BDeu in this table. From Figures 4.1 and 4.2 and Table 4.14 we can make the following observations:

- **MDL and AIC scores**: both scores tend to underfit the data, thus biasing the model for simpler structures. This is evident from the small number of arc's additions performed. As training size increases both scores increase in bias and decrease in variance. However, since MDL has stronger bias for simplicity than AIC, the latter can reduce the bias slightly with time. As a result, AIC outperforms MDL.

- **BDeu score**: this score consistently favors the dependent structure because the addition of an arc will always increase the likelihood of a model. When the complexity grows, BDeu can lead to severe *overfitting* and consequently to a great deterioration of the performance. Note that starting from some k value, the plots of the *training/test* errors clearly evidence *overfitting*. As k approaches its maximum value 7, the *training error* becomes vanishing and the *test error* considerably increases. Since the induced models become more and more complex we can observe a considerable increase in variance. However, this tendency of overfitting diminishes with increasing data size because the variance decreases, thus reducing the *test error*.

- **Preq score**: this score does a more optimal *bias-variance* trade-off at all the time points thus producing the best results in the sense of *complexity-performance*. The k-DBCs induced with Preq are more complex than those induced with MDL and AIC, thus avoinding underfitting. On the other hand, they are less complex than the k-DBCs induced with BDeu, thus avoinding overfitting. From the *bias-variance* decomposition

of the test error we can observe that Preq performs a more artful *bias management* compared to the other scores, thus producing *lower bias* models with no so much variance. As a result, the supervised score Preq obtained the best performance in the *nursery* domain.

Optimal class-models

In each learning step given a particular score there is an *optimal class-model* that gives minimum test error. Optimal class-models perform an *optimal trade-off* between *complexity* and *performance*. The optimal class-model for each score is signalized with a vertical ellipse on the *bias-variance* decomposition of the test error in Figures 4.1 and 4.2. In addition, some statistics about the optimal class-model for all the scores and the six chosen time points are summarized in Table 4.13. The column "Optimal k" indicates the k value that corresponds to the optimal class-model. The next two columns depict the values of two complexity indicators. The column "# Added Arcs" shows the number of arcs added to the NB's stucture for the optimal class-model and the column "Dim" shows the dimension of the corresponding k-DBC expressed by the number of its parameters. The following two columns depict the values of two performance indicators. The column "Test Error" shows the test errors while the column "Gains" shows the gains in the performance obtained with the optimal class-model against NB. For comparison purposes the test errors of NB at different time points appear in Table 4.12. The last column "Rank" shows a rank for each score according to the performance on the test set.

Table 4.12: The test error of the Naïve Bayes classifier at the six chosen time points for the *nursery* dataset

# Training Examp.	500	1000	1500	5000	12000	12500
Test Error	7.20%	12.00%	10.80%	7.10%	7.70%	13.00%

Table 4.13 along with Figures 4.1 and 4.2 lead us to the following important observations:

1. For all the scores and for smaller data ($\leq$ 1000 examples) more simpler k-DBCs for $k = 0, 1$ perform better, although the models are more biased. This fact brings us more evidence that "*high bias can be compensated by low variance to produce accurate*

Table 4.13: Optimal class-model per *score* and per *time point* for the *nursery* dataset

t	# Tr.Ex.	Score	Optimal k	Complexity # Added Arcs	Complexity Dim (# Par.)	Performance Test Error	Performance Gains to NB	Rank
5	500	MDL	0	0.0	99	7.20 %	no	3
		AIC	0	0.0	99	7.20 %	no	3
		Preq	1	4.4	237	5.60 %	+1.60 %	1
		Bdeu	1	7.0	399	7.00 %	+0.20 %	2
10	1000	MDL	0	0.0	99	12.00 %	no	4
		AIC	1	1.4	143	10.60 %	+1.40 %	3
		Preq	1	6.4	298	7.90 %	+4.10 %	1
		Bdeu	1	7.0	397	9.20 %	+2.80 %	2
15	1500	MDL	0	0.0	99	10.80 %	no	4
		AIC	1	1.6	145	8.50 %	+2.30 %	3
		Preq	2	9.0	538	5.90 %	+4.90 %	1
		Bdeu	1	7.0	399	6.60 %	+4.20 %	2
50	5000	MDL	1	1.0	139	4.50 %	+2.60 %	4
		AIC	1	6.0	249	3.90 %	+3.20 %	3
		Preq	3	14.2	1213	2.30 %	+4.80 %	1
		Bdeu	3	18.0	2624	2.70 %	+4.40 %	2
120	12000	MDL	0	0.0	99	7.70 %	no	4
		AIC	2	9.8	561	4.10 %	+3.60 %	3
		Preq	3	17.8	2138	0.60 %	+7.10 %	2
		Bdeu	3	18.0	2879	0.40 %	+7.30 %	1
125	12500	MDL	1	2.0	149	8.00%	+5.00%	4
		AIC	1	7.0	309	4.90%	+8.10%	3
		Preq	4	19.6	3590	1.60%	+11.40%	1
		Bdeu	4	22.0	7967	1.60%	+11.40%	2

classification".

2. As training data increases the bias diminishes at a rate that also reduces variance and consequently the classification error. As a result the optimal class-model (optimal k) slowly moves across the spectrum of class-models, obviously slower for AIC and MDL than for BDeu and Preq. Thus, as the learning process advances k-DBCs induced with the optimal k become gradually much more complex for BDeu and Preq and much less complex for AIC and MDL.

3. At each time point if we choose a class-model *more complex* than the optimal one, the resulting k-DBC is incapable not only of improving the NB's performance, but, what is

Table 4.14: The number of arcs added to the NB's structure averaged over 10 runs for the *nursery* dataset

		# **training examples:** $t \times 100$							**Added Arcs**			
Score	k	5	10	15	50	100	120	125	Allow.	Min	Max	Avg
MDL	1	0	0	0	1	1.4	2	2	7	0	2	0.91
	2	"	"	"	"	"	"	"	13	"	"	"
	3	"	"	"	"	"	"	"	18	"	"	"
	4	"	"	"	"	"	"	"	22	"	"	"
	5	"	"	"	"	"	"	"	25	"	"	"
	6	"	"	"	"	"	"	"	27	"	"	"
	7	"	"	"	"	"	"	"	28	"	"	"
AIC	1	0.8	1.4	2.6	5.8	7.0	7.0	7.0	7	0.8	7.0	4.5
	2	"	"	"	6.0	9.4	9.8	10.0	13	"	10.0	5.7
	3	"	"	"	"	"	"	"	18	"	"	"
	4	"	"	"	"	"	"	"	22	"	"	"
	5	"	"	"	"	"	"	"	25	"	"	"
	6	"	"	"	"	"	"	"	27	"	"	"
	7	"	"	"	"	"	"	"	28	"	"	"
PREQ	1	4.4	6.4	6.6	7.0	7.0	7.0	7.0	7	4.4	7.0	6.5
	2	5.2	8.0	10.0	12.4	12.6	13.0	13.0	13	5.2	13.0	10.6
	3	5.4	"	11.6	14.2	17.4	17.8	17.8	18	5.4	17.8	13.1
	4	"	"	"	"	18.0	19.0	19.6	22	"	19.6	13.7
	5	"	"	"	"	18.4	19.2	20.4	25	"	20.4	13.9
	6	"	"	"	"	"	"	"	27	"	"	"
	7	"	"	"	"	"	"	"	28	"	"	"

worse, the performance can suffer a great deterioration due to a considerable increase in variance, thus overfitting the data. This behaviour is specially observed when we use scores more biased to complex models as BDeu does and when there is few training data. For instance, we can observe the great increase in the variance for BDeu from $k > 1$ at $t = 5$ and $t = 10$. On the contrary, the performance can also be affected if we choose a class-model *less complex* than the optimal one. This situation is still more critical for scores more biased to simpler models, such as MDL and AIC. For instance, at $t = 120$ we can observe as the bias increases for AIC when $k < 2$ and a 2-DBC is the optimal class-model. As a result underfitting takes place.

4. At the last two time points $t = 120$ and $t = 125$ for large amount of training data, the best results were obtained with the Preq and BDeu scores. In terms of bias-variance

decomposition even their behaviours are very similar, although Preq wins in terms of bias management. Thus, as training data increases, BDeu can reduce the overfitting problem and find optimal classifiers in the sense of their *complexity* and *performance*.

We would like to make a last elucidation related to the last observation. The results of the performance depicted in the previous Tables 4.2, 4.3, 4.4, 4.6, 4.7, 4.8 were based on the *cumulative accuracy*, that is, on the proportion of the number of corrected classified examples over the number of all examples seen so far. To obtain the bias-variance decomposition, we have evaluated the test error on a separate batch of unseen examples from the 11^{th} generated sample, which will not used to update the induced classifiers. So why we can observe some discrepancies in these results. For instance, if for the *nursery* dataset we observe the gains in the cumulative accuracy obtained with a 3-DBC at $t = 120$ (see Table 4.7), we can note that the cumulative gain obtained with BDeu (+2.66) is significant smaller that those obtained with Preq (+5.55). And, in general, Preq performs significantly better than BDeu over time. However, note that while there is few training data, a 3-DBC induced with BDeu severely overfits the data and this bad performance has a great weight on the final cumulative performance. However, if we use a 3-DBC with 12000 training examples and compare the performance of Preq and BDeu scores by evaluating each corresponding classifier in an independent test set of 100 examples, the results show that the performance of these scores is very similar as evidence the results from Table 4.13 and Figure 4.2.

4.4 Concluding Remarks

From the above presented results and analysis we can conclude that varying k, the *score* and the *training set size* can have different effects on the performance of induced k-DBCs for different domains. We can corroborate that it is difficult to identify which score is better than another because they were quite sensitive to the amount of data provided and the chosen k value. Even *supervised* scores (k-Fold-CV and Preq) did not always show the best results among all the scores. Nevertheless, for supervised learning it would more appropriate to choose a *supervised* score optimized for classification better matching the learning goal. But k-Fold-CV and Preq, as we know, are very expensive to compute. If for avoiding computational limitations we choose, instead, an *unsupervised* score, then we should take into account

how this score makes the *bias-variance* trade-off for selecting a model with the appropriate complexity according to the amount of training data available and the complexity of the domain. As demonstrated, some amount of additional training data can affect the performance of a particular score. If we have few data it should be not appropriate to choose scores that tend to overfit, such as MLC and BDeu do, because they select complete structures, have high variance, and hence, cannot perform well. If we, instead, have a complex domain (that is, it has a great number of classes and attribute values) we should avoid using scores that tend to underfit, such as MDL and AIC do, because the penalty term will grow faster, and hence, the amount of data required to find new attribute dependencies becomes much greater. However, because MDL penalizes complexity more severely than AIC does, the model induced with MDL tends to be simpler than the model induced with AIC. In general, it will certainly be more suitable, to choose a score that found models of intermediate complexity, such as, for instance, the Bayesian score BD or even the AIC did.

Choosing the appropriate k value is also very dependent on the amount of training data and the score provided to the learning algorithms unless we have some prior beliefs about the actual degree of attribute dependence. For example, consider an extreme situation in which we have only 500 training examples of the *nursery* domain and we unfortunately decided to use the BDeu score for learning a k-DBC using a 3-DBC class-model. Plots of the training/test errors in Figures 4.1-4.2 show that BDeu severely overfits the data. The performance of the resulting model is much worse than that obtained by using the Naïve Bayes. If we decide, instead, to use the same settings only changing the k value to 1 the resulting classifier now shows a good performance and can even outperform the Naïve Bayes. These facts raise the question about the selection of an appropriate k-DBC class-model. As stated, this selection is still more challenging in a prequential learning framework since the amount of training data varies over time. In the next chapter we will describe our adaptive algorithms under the adaptive prequential framework for supervised learning aimed at automatically solving this problem.

Chapter 5

Adaptive Learning Algorithms for Bayesian Network Classifiers

5.1 Introduction

In this chapter we consider adaptive learning algorithms for BNCs in a *prequential learning framework*. As stated, in this framework data arrives at the learning system sequentially. The actual predictive model must first make a prediction and then update the current model with new data. An efficient adaptive algorithm in a *prequential learning framework* involves an artful *trade-off* between the *gain in the quality of the updated model* and the *cost of adaptation*. Since the quality of BNCs is determined by their *predictive capability*, an efficient adaptive learning algorithm for BNCs must be able, above all, to improve its predictive accuracy over time while optimizing the cost of updating. However, in many real-world situations it may be difficult to improve and adapt to existing workflow, operational setting and changing environments. Changes in the learning environment may effect changes in the target concept that the learner is trying to approach at each learning step. This problem is known as *concept drift* in machine learning. In changing environments, learning algorithms should be provided with some control and adaptive mechanisms that effort to handle concept changes and adjust quickly to these changes. Learning systems that track a changing environment are often called *adaptive learning systems*. In this chapter we present an *adaptive, prequential framework* for supervised learning called `AdPreqFr4SL`, which tries to handle the *cost-quality* trade-off and

cope with *concept drift*.

The adaptive strategy in the `AdPreqFr4SL` for incorporating new data leads to a more artful trade-off between the *cost of updating* and the *gain in performance*. This is based upon two main policies: *bias management* and *gradual adaptation*. The basic idea is that we can improve the performance of a BNC while reducing the cost of updating if in each learning step we choose a *class-model* with the appropriate complexity for the amount of training data we have. Moreover, we use new data to primarily adapt the parameters. We adapt the structure only when there is evidence that the performance of the current model no longer improves in a desirable tempo. The `AdPreqFr4SL` integrates simple but efficient control strategies for *bias management* [18] with a method for *handling concept drift* based on *Statistical Quality Control* [21]. If during the monitoring process a concept drift is detected, some actions to adapt the learner to these changes are taken. These actions usually *forget* old examples, as they are examples of an old target function.

In Section 5.2 we will review the main issues related to adaptive learning systems. Then, we will present the main assumptions that we have adopted in the design of our adaptive algorithms. In Section 5.4 we describe the adaptive and control strategies for handling the *cost-quality* trade-off. Then, in Section 5.5 we describe our method for handling concept drift using a Schewart Control Chart. In the last section we will describe the `AdPreqFr4SL` as a whole learning framework that integrates all the presented adaptive algorithms.

5.2 Adaptive Learning Systems

In general, adaptive systems are systems whose function evolves over time, as they improve their performance through learning. In the context of this thesis we consider *adaptive learning systems* as *on-line learning systems* capable of handling concept drift. In general, we could summarize five main factors which are required for drawing up of an efficient adaptive system:

1. **Environment**: The main environmental assumption is concerned to the way in which the instances are presented to the learning system. In *off-line learning* all examples are given at the same time to the learning algorithms. In *on-line learning*, on the contrary, the examples are presented not at the same time. Thus, *on-line learning* is inherently a temporal process where at each time point the system can accept either,

only one example or a suitable-size *batch of examples*. The ability to learn from batches is an important feature that makes a learning algorithm more applicable to *real-world* applications [85]. Even if each instance is classified as it arrives, constructing batches is quite natural since, all examples arriving in the course of a day or a week can be grouped together thus performing model adaptation in jumps when there is some accumulated experience. On the other hand, an *on-line learning system* must make a usable model available for predictions at any time point. However, as pointed out in [119], there are real-world environments with very strong constrains in *computing time* and *memory space* that can affect the formulation and performance of on-line learning systems. In this sort of environments, an incremental algorithm, which continuously revises and refines a domain model by processing new data as they are available, is more suitable. Another crucial question concerns whether the environment *can change* or is assumed to be *constant* over time. In changing environments, learning algorithms should be provided with some extra mechanisms that effort to adapt the system to these changes.

2. **Model Memory**: on-line learning systems must have policies that deal with the way the target concept descriptions are stored. There are three possibilities:

   (a) **store one hypothesis**

   (b) **store several alternative hypotheses**

   (c) **store no hypothesis**

   This definition depends on the classifier employed (i.e. it is *classifier-dependent*) and how it makes use of the available resources (*memory space*, *computer time*, etc.) for learning and adaptation purposes. The most favorable would be a model that would limit the *cost of updating* and the *use of memory space*. For more sophisticated classes of classifiers (e.g. *neural networks*) it may be unreasonable to keep in memory several alternative hypotheses. For systems with *model memory*, learning can occur either in an *incremental mode* or in a *temporal batch mode* [94]. Incremental learners modify their current descriptions whenever new data arrives. On the contrary, batch learners replace current descriptions with new ones using all the examples seen so far. Another important question is concerned with model initialization (known as *cold start*), i.e., whether some background knowledge (expert or empirical) can be used to build an

initial hypothesis.

3. **Data Memory**: The formulation and performance of *on-line learning systems* also depends on the way past examples are treated. Three data memory models exist:

   (a) **long-term memory**: all the examples from the input stream are stored

   (b) **short-term memory**: only some of the past examples are stored

   (c) **no memory**: no examples are stored

   Approaches with *short-term memory* have been mostly used in on-line learning systems that deal with concept drift. This sort of approach requires the identification of policies of how to select the representative set of examples from the input stream, how to maintain them, and use them in future learning episodes. Maloof and Michalski in [94] provide a good classification and survey about how different on-line learning systems use the *model* and *data* memories for handling concept drift.

4. **Adaptive actions**: They operate over both, the *model memory* and the *data memory*. Actions for updating the model aim to incorporate new data to the current hypothesis (or alternative hypotheses) so as to yield a more effective classifier, whereas actions for updating the *data memory* aim in providing a more representative training set for learning purposes and are more oriented for handling concept drift. The incorporation of new data in *on-line learning* can be conducted using:

   (a) **an evolutionary approach**: the current hypothesis is updated based on new examples

   (b) **a revolutionary approach**: a batch learning procedure is invoked after new incoming data using all the examples seen so far, that is, at each learning step the current hypothesis is rebuilt from scratch

   (c) **a hybrid approach**: it takes elements from both the *revolutionary* and *evolutionary* approach. For instance, a new hypothesis can be learned from new data and then the old hypothesis is combined with the new hypothesis

   Thus, for systems follow an *evolutionary approach* learning occurs in an *incremental mode* whereas for those that follow a *revolutionary approach* learning occurs in a *temporal batch mode*. Since learning from scratch use all the data provided so far, the

revolutionary approach for updating the classifier is essentially optimal in terms of the quality of the hypotheses it can induce [45]. However, although theoretically it is possible to rebuild a new model from scratch after each new observation is seen, as the data grows, the cost becomes prohibitive. An alternative approach is to find a way to accommodate new observations into the current model in an incremental fashion. However, one disadvantage of the *evolutionary approach* approach is that the resulting hypothesis may be of a lower quality when compared to the hypothesis induced from all the observations seen so far. A *hybrid approach* that refines the structure of a Bayesian Network with new data was proposed in [82].

The precise way in which the *model memory* can be updated in order to include new data is also *classifier-dependent*. Some supervised learning algorithms are naturally incremental, for example *k-nearest neighbors* and *Naïve Bayes*. For other classifiers (e.g. *decision trees*, *Bayesian networks*, *support vector machines*) updating can be a more difficult task. A more in depth discussion about these issues can be found in [85]. A pioneer work comparing the *revolutionary* and *evolutionary* approaches in online learning is given in [96]. A more related work [45] compares the *revolutionary* and *evolutionary* approaches for sequential updating of Bayesian networks.

Adaptive actions can be performed *globally* or *locally*. Local adaptation should rather be viewed as model refinement. It is based on locality assessments that avoid the re-learning process of all the structure and parameters. For instance, a technique for using new data to revise a given Bayesian network in order to improve its classification accuracy is proposed in [113]. This method employs mechanisms similar to those used in *logical theory refinement*. This uses the data to focus the search for effective local modifications to the networks.

5. **Control Strategies**: Many adaptive systems employ regular model updates while new data arrives. However, when the cost of updating the model in light of new data is high, it is desirable to find a way to decide whether it is inevitable to trigger the updating process. Thus, an alternative approach is to provide the adaptive system with some controlling mechanisms that effort to select the best adaptive actions according to the current learning goal. For deciding about the best adaptive actions, at least one *characteristic value* (*indicator*) should be observed over time and compared to

previous values regularly. This process is referred as *monitoring*. An *indicator* is a qualitative or quantitative parameter that can be assessed for detecting both qualitative and quantitative problems in the current state of a learning system. Three categories of indicators can be considered [70]:

(a) **performance measures** (e.g. the *accuracy* or *error rate* of the current classifier): independent of the classifier, generally applicable.

(b) **properties of the classification model** (e.g. the *complexity* of the current hypothesis): *classifier-dependent*.

(c) **properties of the data** (e.g. *class distribution*, *attribute-value distribution*): independent of the classifier, generally applicable.

Moreover, indicators can be designed *globally* or *locally*. Global indicators are more suitable for assessing the global performance (e.g. *accuracy*, *global scores*). In contrast, local indicators (e.g. *local scores*, *complexity of a local structure*) are very useful for assessing local changes in the behavior in order to perform local refinements of the current model. For instance, Cowell and el. in [34] presented a range of global and local indicators for BNs (they called *monitors*) based on standardized scores in a *prequential learning framework*.

5.3 Main Assumptions and Settings

The main environmental assumption that drives the design of our framework 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 so that we can perform model adaptation in jumps when there is some accumulated experience. These batches are assumed to be of equal size, each containing m examples. Related to the *model memory* we propose to maintain an *unique hypothesis* $h_{\mathcal{C}}$ defined as a pair (S, Θ_S), where S is the structure and Θ_S are the parameters for that structure. We chose the class of k-DBCs for illustrating our adaptive approach. This class is very suitable because, as stated, by increasing the k value we can easily scale-up the model complexity of BNCs. We assume that all the variables are *discrete* and the data is *complete*.

The main difference between the prequential framework for supervised learning presented in Algorithm 4 and the `AdPreqFr4SL` is that the latter is adaptive. The 4 is provided with some controlling mechanisms that attempt to select the best adaptive actions according to the current learning goal. Thus, for each incoming batch 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 order to achieve the current goal.

The incorporation of new data in the `AdPreqFr4SL` is conducted using an *evolutionary approach*, that is, whenever new data arrives to the learning system, the current hypothesis is updated with new data. In principle, the current hypothesis is subject to changes as a new batch arrives by modifying both its *parameters* and its *structure*. But whenever it is possible, we will try to update only the parameters. Updating the structure of Bayesian networks is a computationally expensive task since searching is involved. The space and time complexity of search procedures increases quadratically with the number of variables (at each search step, each variable depends on the order of one change with respect to each of the other variables) [61]. Moreover, search procedures requires keeping in the main memory all the sufficient statistics needed to compute the scores for all the candidate structures, a *huge memory space*.

Nowadays, there are only a limited number of previous works concerning sequential updating of Bayesian Network structures. These works have mainly adapted a hill-climber search procedure and/or implemented more sophisticated data structures and methods for storing and computing the sufficient statistics in an incremental fashion[1]. We use a hill-climbing search procedure to learn the structure of BNCs. Whenever a structure adaptation is triggered, the hill-climber moves from the current structure to the best neighbor until it cannot improve the score anymore. During the search process we use the sufficient statistics from all the data seen so far for computing the score of each candidate structure. Only when a concept shift is detected the sufficient statistics are recomputed using the examples of the new concept. Thus, one of the limitations of our proposal is that we assume that we can keep in memory all the needed sufficient statistics. We left for future work the integration of more sophisticated data structures for storing the sufficient statistics into the `AdPreqFr4SL`

[1] We will provide a short overview of these approaches in Section 5.4.4.

and focus here on other issues, namely:

1. We use Bayesian networks for the particular task of classification in a *prequential learning scenario*.

2. We approach the *cost-performace* trade-off through *bias management* and *adaptation control*.

3. We handle *concept drift*.

In the next sections we will present the *adaptive* and *control strategies* that we have adopted in our adaptive framework for handling the *cost-quality* trade-off and *concept drift*.

5.4 Cost-Quality Management

The strategy that we follow in the `AdPreqFr4SL` for incorporating new data try to perform an artful *cost-quality* trade-off. This is based upon two main policies: *i*) *bias management*; and *ii*) *adaptation control*.

Since most of the time we have no clear idea of how to select an appropriate *class-model* of BNCs for the current learning task, we propose the use of the class of k-DBCs and start with the simplest class-model, that is, the Naïve Bayes classifier. We can improve the performance of Naïve Bayes over time if we trade-off the *bias reduction* which leads to the addition of new attribute dependencies, and, consequently, to the estimation of more parameters, with the *variance reduction* by accurately estimating the parameters. However, as argued in Section 3.7.2 and further corroborated from the results of the study with k-DBCs, as the learning process advances variance will decrease and we should place more focus on *bias management*. We can reduce the bias, if we reduce the bias resulting from the independence assumptions by gradually adding dependencies among attributes over time. To this end, we gradually increase the k value so that the search space can be also progressively extended over the space of possible structures. By choosing an appropriate k value at each learning step we can scale up the model complexity to suit the available training data thus avoiding the problems caused by either too much bias (underfitting) or too much variance (overfiting). This way we reduce bias at a rate that also reduces variance and consequently the classification error.

On the other hand, we use simple control strategies based on the performance's dynamics to decide when it makes sense to do the next move in the spectrum of attribute dependencies and to start searching for new dependencies. This way we reduce the cost of updating because we use new data to primarily adapt the parameters. We adapt the structure only at sparse time points, when there is some accumulated data and there is evidence that the use of the current structure no longer guarantees the desirable improvement in the performance. Finally, we stop the adaptation process when there is evidence that the use of more training data will not result in significantly improved performance. However, if any significant change in the behavior is further observed, then the adaptation procedures will be once again activated.

Trough all this section we will assume stability in the target concept. In Sections 5.4.1 and 5.4.2 we formalize the adaptive and control policies for handling the *cost-quality* trade-off based on *bias management* and *adaptation control*. Then, in Section 5.4.3 we will present the adaptive algorithm that integrates the proposed adaptive and control strategies. Finally, in Section 5.4.4 we will provide an overview of related work.

5.4.1 Adaptation Policy

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

- INITIAL LEVEL: a new hypothesis is built using the simplest Naïve Bayes.
- FIRST LEVEL: only the parameters are updated with new data. Optionally we can use the Iterative Bayes for improving the parameter estimates as described in Section 3.6.
- SECOND LEVEL: the structure is updated with new data.
- THIRD LEVEL: if it is still possible, the maximum number of allowable dependencies (k) is increased by one, and the current structure is once again adapted.

The rationale of the adaptation strategy is as follows. In the absence of any information about the true model the adaptive algorithm starts from its INITIAL LEVEL: k is set to 0 and a new *Naïve Bayes* (NB) classifier is built. Then, whenever new data arrives, it first tries to improve the NB by adapting only its parameters. When there is evidence indicating

that the performance of the NB stops improving in the desirable tempo, k is incremented by one and the NB structure begins to be adapted by searching for new attribute dependencies. At this time point, there is more data available which could allow the search procedure to find new dependencies. Note that only when $k = 0$, the algorithm jumps from the FIRST LEVEL to the THIRD LEVEL of adaptation. Thus, a 1-DBC structure begins to be searched using the *hill-climber search* procedure only with arc additions. Independently of which new dependencies were found or not, the algorithm is re-launched from the FIRST LEVEL, that is, it continues performing only parameter adaptation.

Suppose that at some time point t a new k-DBC structure is found where $k > 0$. Whenever a new structure is found, the algorithm is re-launched from the FIRST LEVEL of adaptation, that is, it performs only parameter adaptation until there will be again evidence that the performance using this structure stops improving. In this case, the algorithm moves to the SECOND LEVEL of adaptation and the current structure begins to be adapted by searching for new attribute dependencies. At this stage the search procedure is also allowed to delete some existing dependencies thus correcting previous errors in the search process. Only if the resulting structure remains the same, the algorithm moves to the THIRD LEVEL of adaptation: k is incremented by one and the search process continues working, now in the augmented search space. If after searching the resulting structure still continues the same then some control heuristic verifies if the current performance has already reached its *plateau*[2]. If the plateau is reached, it is assumed that adding additional data does not result in discovering new attribute dependencies and further improvements in the accuracy. In this case the adaptation process is stopped. As a result, the current hypothesis is not further updated with new data. However, the performance continues to be monitored. If any significant change in the behaviour is observed, then the adaptation procedures are once again activated. For avoiding k to increase unnecessarily, the old value of k is recovered whenever the search procedure is not able to find new dependencies, thus keeping the original search space. Only in the case when an *abrupt concept drift* is detected, the algorithm is re-launched from its INITIAL LEVEL and a new NB is built using the examples from a *short-term memory*. The whole adaptive algorithm for *cost-quality* management will be formalized later in section 5.4.3 after we finish describing the control strategies that we have adopted to this end.

[2]A plateau is a part of the learning curve where the predictive accuracy is essentially flat.

Algorithm 9 is the pseudo-code of the *parameter-updating procedure*. As stated, sequential updating of the parameters on the light of new data is straightforward for discrete variables and complete data. This requires a simple scan through all the examples in the given batch in order to increment the frequency counters corresponding to the values of each example. In addition, we provide the algorithm with the boolean variable `bIterativeBayes` to indicate whether to use the Iterative Bayes for improving the parameter estimates or not.

Algorithm 9 The algorithm for updating the parameters of k-DBCs

Require: A current classifier $h_{\mathcal{C}} = (S, \Theta_S)$ belonging to the class of k-DBCs, a batch B of m labeled examples of $< \mathbf{X}, C >$, a boolean variable `bIterativeBayes` to indicate whether the Iterative Bayes is used or not

Ensure: The set of parameters Θ_S updated with the examples of B

1: Update $\mathbf{T}(\mathcal{D} \mid S)$ with the examples from B
2: **if** `bIterativeBayes` **then**
3: IterativeBayes($h_{\mathcal{C}}$, B, ...) {improve the parameter estimates using Algorithm 6}
4: **return** $h_{\mathcal{C}}$ with the updated set of parameters Θ_S

Algorithm 10 depicts the pseudo-code of the *structure-updating* procedure. We use the hill-climbing search procedure due to its incremental nature and obvious simplicity for computational implementation[3]. We provide the learning algorithm with the current hypothesis $h_{\mathcal{C}} = (S, \Theta_S)$, the new batch B of labeled examples, a boolean variable `bUseArcDeletions` to indicate whether the operator `deleteArc` is used or not, the space $\mathcal{S}$ of possible network structures for the current k-DBC class-model and a scoring function `Score`$(S, \mathcal{D})$. Whenever a structure-adaptation process is launched, the current hypothesis is used as the initial model for the hill-climbing search procedure. If the current structure is an 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. Then it iteratively chooses the operation that results in the maximal improvements in the given score until there is no more improvement for that score or until it is no possible to perform a new operation.

5.4.2 Control Policy

Our adaptive proposal requires the definition of some control criteria and tools for deciding when start adapting the structure because the actual performance no longer improves in a

[3] As stated, 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.

Algorithm 10 The hill-climbing algorithm for updating the structure of k-DBCs

Require: A current classifier $h_{\mathcal{C}} = (S, \Theta_S)$ belonging to the class of k-DBCs, a batch B of m labeled examples of $< \mathbf{X}, C >$, a boolean variable `bUseArcDeletions` to indicate whether the operator `deleteArc` is used or not, the space $\mathcal{S}$ of possible `DAGs` restricted for the current maximum allowable degree of attribute dependence k, a scoring function $\texttt{Score}(S, \mathcal{D})$

Ensure: A k-DBC with high value of the score $\texttt{Score}(S, \mathcal{D})$

$\mathcal{O} \Leftarrow \{\texttt{addArc}\}$
if `bUseArcDeletions` **then**
    $\mathcal{O} \Leftarrow \mathcal{O} \bigcup \{\texttt{deleteArc}\}$
continue $\Leftarrow$ True
while continue **do**
    Compute $\texttt{Score}(S, \mathcal{D})$
    Find best operator $\texttt{op} \in \mathcal{O}$ such that $\texttt{op} = \arg\max_{\texttt{op} \in \mathcal{O}} \texttt{Score}(\texttt{op}(S), \mathcal{D})$
    if op exists $\wedge$ $\texttt{Score}(\texttt{op}(S), \mathcal{D}) > \texttt{Score}(S, \mathcal{D})$ **then**
        $S \Leftarrow \texttt{op}(S)$ {Apply the operator to the current structure}
    else
        continue $\Leftarrow$ False
end while
return $h_{\mathcal{C}}$ with the updated structure S and the updated parameters Θ_S

desirable tempo, and also to determine when stop the adaptation process because the use of more training data will not result in significantly improved performance. To this end, we monitor the *model error* defined as the proportion of misclassified examples in the total of the examples that were classified using the same structure.

Observation of the Model Error

During the whole learning process we obtain a sequence $\{S_0, S_1, S_2, \ldots, S_q\}$ of different structures which can belong to different class-models of k-DBCs (with increasing k values). More formally, let t be the current time point and suppose that at time t_p, a new structure S_p begins to be used ($0 \leq p \leq q$). At time point t we are interested in evaluating the model error of the classifiers $h_{\mathcal{C}}$ induced with the actual structure S_p, without considering the errors when other structures were used for classification.

Definition 41. The **model error** Err_S of the actual structure S_p with respect to a target $f(\mathbf{x})$ at the current time point t is the proportion of the *misclassified* examples by the

classifiers $h_{\mathcal{C}}$ induced with the same structure S_p. That is,

$$Err_S(t) \equiv \frac{1}{N_p} \sum_{t_i=t_p}^{t} \sum_{\mathbf{x} \in B^{(t_i)}} \delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x} \mid S_p, \Theta_{S_p}^{(t_i)})) \tag{5.1}$$

where $N_p = m(t - t_p + 1)$ is the total of classified examples and m is the number of examples in one batch.

We monitor the behaviour of the model error Err_S with time in order to asses the actual performance. To this end, 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*. We denote it by `model-LC`. The `model-LC` depicts the relationships between the time point t and the model error Err_S. Since at each learning step we process a batch of m examples, we have a direct relationships between the number of training instances ($t \times m$) and the performance over time. Observation of the `model-LC` is crucial, because it helps explain the behavior of the adaptive learning algorithm using different structures with increasing complexity.

The *slope* of a learning curve is an indicator of how much performance can be gained by increasing the number of examples. Theoretical and empirical studies have pointed out [111] that learning curves typically have three different parts: *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. A learning curve is *well-behaved* if its slope monotonically is *non-increasing* with t except for local variance [111]. Moreover, it was empirically showed that, in many cases, a learning curve that depicts the error rate starts *behaving well* when its graph becomes *monotically decreasing* and *convex* for a given number of points [14]. A learning curve *converges* when it reaches its final plateau.

We are interested in observing the behaviour of the `model-LC` for tracking two situations:

`S1`. At which time point does the performance of the current model stop improving in the desirable tempo?

`S2`. At which time point has the performance already reached a plateau?

We consider that the situation `S1` is met if at the current time point: *i*) the `model-LC` starts *behaving well*, that is, the curve is *convex* and *monotically non-increasing* for a given number of points; *ii*) its slope is *gentle*. If the situation `S1` is detected we start adapting the structure.

Thus, whenever we start using a new structure we will wait until the `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 slope of the `model-LC` to detect whether it has already reached its *plateau* (situation `S2`). If the plateau is reached, then the *stopping criteria* is met and we stop adapting the model.

The following question thus arises: *How does one verify whether the required criteria of* discrete convexity *and* non-increasing *trend are met*? From all the explored methods, we empirically found out that by using simple heuristics based on the geometrical properties of the curve we could more consistently determine *discrete convexity* and the *slope* of the `model-LC` taking into account the local variance. In addition, we use the Sen's slope estimator [127] for *trend evaluation*.

A Geometrical Method for Determining Discrete Convexity

Given the coordinates of a set of points in the plane the task is to find an efficient way to determine whether the curve that connected these points is *convex*. We use the *signed area* to test whether three points are arranged in a *convex pattern* as further explained[4].

Definition 42. The **signed area** enclosed by a general n-sided polygon P with vertices $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$ (in order around the perimeter) is given by

$$\mathcal{A}(P) = \frac{1}{2}\sum_{i=1}^{n}(x_{i+1}y_i - x_i y_{i+1}) \tag{5.2}$$

where $x_{n+1} = x_1$ and $y_{n+1} = y_1$. This is called the *signed area* because the result can be *positive* or *negative* depending on whether the path is *counterclockwise* or *clockwise*.

Given the coordinates of three *non-colinear* points $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ and $p_3 = (x_3, y_3)$ in the plane it is always possible to construct a triangle T whose vertices are precisely the points p_1, p_2 and p_3. The signed area of a triangle T enclosed by the path $p_1 \rightarrow p_2 \rightarrow p_3$ is then given by

$$\mathcal{A}(T) = \frac{1}{2}[(x_2y_1 - x_1y_2) + (x_3y_2 - x_2y_3) + (x_1y_3 - x_3y_1)] \tag{5.3}$$

[4]The subjects and results that we here exposed have been extracted from the references [87] and [105] on *Computational Geometry*.

Proposition 5.4.1. *Three points* p_1, p_2 *and* p_3 *are arranged in a* convex pattern *iff the* signed area *of the triangle* $\mathcal{A}(T)$ *with vertices* p_1, p_2, p_3 *is positive.*

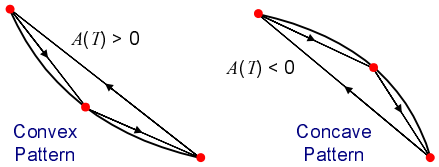

Figure 5.1: Concave and convex patterns given three points in the plane

As illustrated in Figure 5.1 the convex pattern is consistent with the existence of a *counterclockwise* path around the triangle. The concave pattern, on the contrary, is consistent with the existence of a *clockwise* path. The signed area is *positive* if the path $p_1 \rightarrow p_2 \rightarrow p_3$ is oriented *counterclockwise* and *negative* otherwise.

The Sen's Slope for Trend Evaluation

The Sen's slope estimator [127] is a *non-parametric*, linear slope estimator that can be used for trend evaluation. A *temporal trend* is the general increase or decrease in a set of observed values over time. Suppose we have observed n values $y(t_1), y(t_2), \ldots, y(t_n)$ and we are interested in determining if there is a *non-increasing trend* in these values. To this end, we estimate the Sen's slope, denoted by $\texttt{SenSlope}^{(n)}$. First, we need to compute the slopes

$$\frac{y(t_j) - y(t_i)}{(t_j - t_i)}$$

for all the pairs of observed data such that $t_j > t_i$. The Sen's slope is then the *median value* of the resulting slopes. We consider that there is a *non-increasing* trend if the Sen's slope $\texttt{SenSlope}^{(n)}$ is less than some *threshold limit* δ_s, such that δ_s is a very small positive number.

The Detection Method based on the Model Error

We empirically found that by using simple heuristics based on the graphical behaviour of the most recent q points in the `model-LC`, 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 $q = 7$.

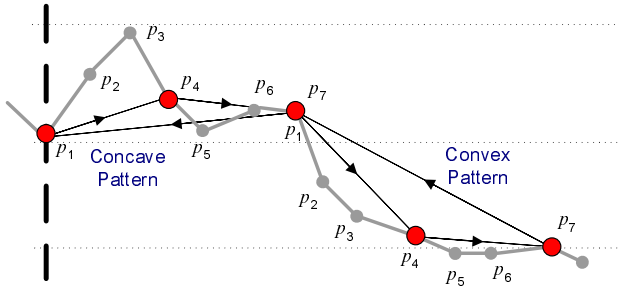

Figure 5.2: The last seven points $p_1, p_2, ..., p_7$ in the `model-LC` are analyzed to determine the existence of a *convex pattern* and a *non-increasing trend*

As illustrated in Figure 5.2, we first construct a triangle T with the points p_1, p_4 and p_7 and compute its signed area $\mathcal{A}(T)$. Taking into account the local variance we consider that p_1, p_4 and p_7 are arranged in a *convex pattern* if the resulting signed area is greater than some *threshold limit* δ_a (our tolerance for convexity), such that δ_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 *well-behaved* with respect to the three selected points. We assume this situation if the points are *almost colinear* given a tolerance δ_c, such that δ_c is a very small positive number. That is, if the sinus of each angle $\angle_l$ is less than δ_c. The sinus of an angle $\angle p_i, p_j, p_k$ is computed as follows:

$$\sin(\angle p_i, p_j, p_k) = \frac{k_2 - k_1}{1 + k_1 k_2}$$

where

$$k_1 = \frac{y(t_j) - y(t_i)}{t_j - t_i}, \; k_2 = \frac{y(t_k) - y(t_j)}{t_k - t_j}, i < j < k$$

Finally, we obtain the Sen's slope for trend evaluation. By joining all the above criteria, we consider that the points $p_1, p_2, \ldots, p_7$ are arranged in a *convex pattern* with a *non-increasing trend* and a *gentle slope* if for a given positive small number ϵ_1 (our threshold for the gentle slope), the following criterion is met:

$$\delta_a < \mathcal{A}(T) < \epsilon_1 \quad \wedge \; \sin(\angle_l) < \delta_c, \forall \angle_l, \; l = 1, 2, 3, 4 \; \wedge \texttt{SenSlope}^{(7)} < \delta_s \tag{5.4}$$

We consider that the `model-LC` has already reached its *plateau* if given a positive small

number ϵ_2 (our threshold for the plateau), such that $\epsilon_2 < \epsilon_1$, the following criterion is met:

$$|\mathcal{A}(T)| < \epsilon_2 \ \wedge \ \sin(\angle_l) < \delta_c, \forall \angle_l, \ l = 1, 2, 3, 4 \tag{5.5}$$

We further call the above criterion 5.5 the *stopping criterion*. We obtained satisfactory results by setting $\delta_a = -0.0001$ (our tolerance for *convexity*), $\delta_s = 0.01$ (our threshold for *non-increasing* trend) and $\delta_c = 0.001$ (our tolerance for *colinearity*). Both the thresholds for the gentle slope (ϵ_1) and for the plateau (ϵ_2) are parameters of the adaptive algorithm.

The Detection Method based on the Batch Error

We combine the previous method based on the observation of the `model-LC` with an alternative heuristic that, instead, is based on the observation of the *batch error* before and after the adaptation. This heuristic has been demonstrated to be efficient for an early detection of the point at which we should start adapting the structure.

Definition 43. The **batch error** Err_B of a classifier $h_{\mathcal{C}}$ with respect to a target $f(\mathbf{x})$ and a given batch B of m examples is the proportion of the *misclassified* examples by $h_{\mathcal{C}}$. That is,

$$Err_B(h_{\mathcal{C}}) \equiv \frac{1}{m} \sum_{\mathbf{x} \in B} \delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x})) \tag{5.6}$$

We proceed in the following way. Suppose that a new batch B arrives at the learning system and the examples are classified using the current hypothesis $h_{\mathcal{C}}$. Assume that feedback can be obtained and the batch error Err_B can be evaluated. First, the examples from B are used to update the parameters of $h_{\mathcal{C}}$. After updating, Err_B can be assessed once again, but now using the adapted hypothesis. Whenever we obtain a decrease of the batch error after parameter adaptation, it would be a straightforward idea to consider that the learner is still able to learn about the current target concept using the current structure. Otherwise, if for a pre-defined number of consecutive times, denoted by `maxTimes`, the batch error does not decrease, we assume that increasing the number of training examples will not result in further improvements on the parameter estimates and signal a `STOP-IMPROVING` situation in order to trigger the structure-updating procedure. More formally:

$$IF \ \ consecCounter(Err_B^{\texttt{AFTER-ADAP}} \geq Err_B^{\texttt{BEF-ADAP}}) = \texttt{maxTimes}$$

$$THEN \ \ performanceState \leftarrow \texttt{STOPIMPROVING}$$

where, $Err_B^{\texttt{BEF}-\texttt{ADAP}}$ and $Err_B^{\texttt{AFTER}-\texttt{ADAP}}$ denote the batch errors *before* and *after* parameter adaptation, respectively.

An Example to Illustrate the Control Strategies

Figure 5.3 illustrates the behaviour of the `model-LC` obtained with our `AdPreqFr4SL` using one randomly generated sample of the *adult dataset* and batches of 100 examples. To serve as baseline we also plot the *error rates* obtained with a NB classifier (the 0-DBC) and with a 3-DBC (the class-model with best performance) induced from scratch at each time point using Algorithm 7. The *error rate* (of the NB and the 3-DBC) is based on the *cumulative error* taking into account all the examples classified so far. The *model error*, on the contrary, is recomputed each time a new structure is used. Thus, the examples that were misclassified in the past, when other structures were used to classify the examples, have no any influence in the current analysis. Since the *adult* dataset is a hard domain to learn, its *learning curves* are not so *well-behaved*. However, our detection method works as expected even in this hard domain. In Figure 5.3 we indicate the points and the conditions which lead to a *structure-adaptation* action. We can see that the graphical behavior of the *model error* neatly corresponds to the pointed out conditions. Moreover, from the resulting sequence of

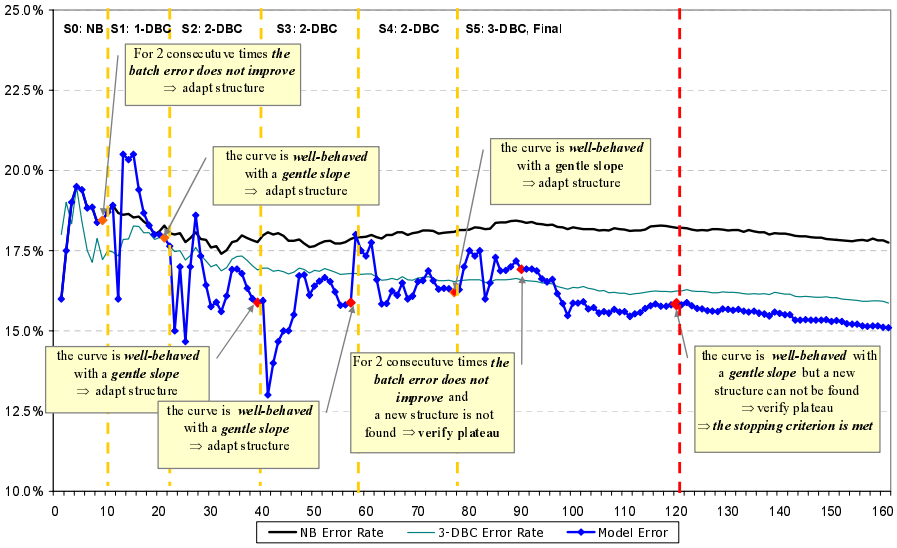

Figure 5.3: Behavior of the model error for the adaptive learning algorithm. Vertical lines indicate the time points at which the structure changed. On top, the resulting structures with their corresponding k-DBC class-models are presented

structures, we can infer that 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. As a result, the structure changed only five times during all the learning process.

5.4.3 The Cost-Quality Handler Algorithm

Algorithm 11 is the *adaptive algorithm* that aims to incorporate new data to the current classifier so as to yield an artful *trade-off* between the *cost* of updating and the *gain* in the performance. For handling the *cost-quality* trade-off the algorithm integrates all the adaptive and control strategies presented in the previous sections based on *bias management* and *adaptation control*. Its rationale has been widely described in Section 5.4.1.

The algorithm is provided with the current hypothesis $h_{\mathcal{C}}$ belonging to the class of k-DBCs, a batch B with the new incoming examples, the current k value, and the values of five parameters: the `kMax` value for the maximum allowable degree of attribute dependence, a boolean variable `bIterativeBayes` to indicate whether to use Iterative Bayes or not, the two thresholds used in the control criteria: `eps1` for the *non-increasing gentle slope* and `eps2` for the *plateau*, and the number of consecutive times, `maxTimes`, the Err_B does not decrease after parameter adaptation that is used in the alternative detection method based on the *batch error*.

First, the current hypothesis $h_{\mathcal{C}}$ is used to classify the examples of B. Next, the Err_B and Err_S are evaluated. The behaviour of the `model-LC` is then analyzed in order to verify if the conditions of *discrete convexity*, *non-increasing trend* and gentle slope are met. If the conditions are met, it is assumed that the performance no longer improves using the current structure and the *structure-updating procedure* is triggered using Algorithm 10. Otherwise, it is assumed that the performance is still improving and the parameters are updated with new data using Algorithm 9. If the batch error decreases after parameter adaptation, it is assumed that the parameter estimates can be still improved using new data. The updating process finishes and the updated hypothesis $h_{\mathcal{C}}$ is returned. Otherwise, if for the predefined number of consecutive times `maxTimes`, the batch error does not improve after parameter adaptation, then it is assumed that the performance stops improving using the current structure. If the current structure is not the NB structure ($k > 0$) then the current structure is adapted by

Algorithm 11 The adaptive algorithm for incorporating new data to the current k-DBC

Require: A current hypothesis $h_{\mathcal{C}} = (S, \Theta_S)$ belonging to the class of k-DBCs, a batch B of m labeled examples, a k value for the current allowable degree of attribute dependence, a value `kMax` for the maximum allowable degree of attribute dependence, the thresholds `eps1` for the non-increasing gentle slope and `eps2` for the plateau, a number of consecutive times `maxTimes` we wait while the batch error does not improve after parameter adaptation, a boolean variable `bIterativeBayes` to indicate whether Iterative Bayes is used or not.

Ensure: the updated hypothesis $h_{\mathcal{C}}$

Predict B with the current hypothesis $h_{\mathcal{C}}$
$Err_B \Leftarrow$ Estimate the current batch error
$Err_S \Leftarrow$ Estimate the current model error
Add $y(t) = (t, Err_S)$ to the `model-LC`
// Observation of the `model-LC`
if `model-LC` is Convex-NonIncreasing-with-GentleSlope(`eps1`) **then**
    performanceState $\Leftarrow$ `STOP-IMPROVING` {criterion 5.4 is met}
    ELSE
    performanceState $\Leftarrow$ `IS-IMPROVING`
// Decision about the best adaptive actions
if performanceState = `IS-IMPROVING` **then**
    updateParameters($h_{\mathcal{C}}$, B, `bIterativeBayes`) {first level of adaptation}
    if consecCounter($Err_B^{t_{\texttt{AFTER}-\texttt{ADAP}}} \geq Err_B^{t_{\texttt{BEF}-\texttt{ADAP}}}$) = `maxTimes` **then**
        `performanceState` $\Leftarrow$ `STOP-IMPROVING`
if performanceState = `STOP-IMPROVING` **then**
    if $k > 0$ **then** updateStructure($h_{\mathcal{C}}$, B, ...) {second level}
    if (not change(S) $\wedge$ $k <$ `kMax`) $\vee$ $k = 0$ **then**
        if $k > 0$ **then** `bUseArcDeletions` $\Leftarrow$ TRUE
        $k+ = 1$ {third level: increment the allowable degree of attribute dependence and continue searching}
        updateStructure($h_{\mathcal{C}}$, B, `bUseArcDeletions`,...){use Algorithm 10}
        if not change(S) **then**
            $k- = 1$ {if the structure doesn't change, we recover the previous k value}
    // Observation of the `model-LC` *to verify if it has reached a plateau*
    if (not change(S) $\wedge$ `model-LC` has-Plateau(`eps2`)) **then**
        StopAdapting $\Leftarrow$ TRUE {stopping criterion 5.5 is met}
return the updated $h_{\mathcal{C}}$

searching for new dependencies or removing existing ones using the hill-climber algorithm. If after updating the resulting structure remains the same or the current structure is the NB ($k = 0$), then the algorithm moves to the *third level* of adaptation. It increments k by one and continues searching, now in the extended search space. If after adaptation, the resulting structure still continues the same, then the *stopping criterion* is met, and hence, the adaptation process is stopped. As a result, the algorithm returns the updated hypothesis $h_{\mathcal{C}}$.

5.4.4 Related Work

A complete discussion about the main drawbacks in the sequential updating of the structure of Bayesian networks (BNs) as well as an overview of the incremental approaches developed so far, along with a new proposal, is given by Roure in [119]. All these approaches are based on hill-climbing search and mainly aimed at optimizing the *computational cost* and/or the *memory space* inherent to the problem of incremental learning of the structure of BNs. The approaches proposed by Buntine [15] and Friedman and Goldszmidt [45] are very similar to each other. They proposed to maintain not one, but a set of alternative BNs following an *evolutionary updating strategy* and a *Bayesian approach*. First, new data is used to update the sufficient statistics and thereof to update the posterior probabilities of alternative structures. After that, additional search over the space of alternative hypotheses is performed. Both approaches maintain only the sufficient statistics for the more promising structures, thus optimizing the memory space, and both create mechanism to efficiently update these statistics in the light of new data. Lam and Bachus in [82], instead, followed a *hybrid updating strategy* by using new data to build a new structure. After that the old structure is combined with the new one. For that reason they refer their approach as *refinement* rather than as incremental learning.

The research done by Roure in [119] is of particular relevance to the work described in this thesis. Roure proposed some heuristics in order to transform a *batch hill-climbing algorithm* into an *incremental* one. The incremental algorithm is provided with some control strategies in order to detect when the current network structure should be updated with new data. To detect when adaptating the structure, he proposed to analyze the order of the different hypotheses induced during the search process in the previous learning step by means of their scores. If when new data is available the order of the models in the search path is altered according to their scores, then he considered that new data provides new information that is no taken into account in the current hypothesis. In this case, a structure-adaptation process is triggered; otherwise, the structure is not adapted with new data. In addition, Roure proposed some heuristics that manipulates an `AD-tree` in order to avoid storing *sufficient statistics* that are unlikely to be useful for the learning task. `AD-tree` [100] is a *tree-like* structure allowed us be more efficient in the way the *sufficient statistics* are stored in the memory space.

Approaches for Scaling up Learning Algorithms

Data mining is concerned with very huge databases available in data streams which usually do not fit in main memory. Our adaptive proposal in some way is related to the *scaling-up methods* to handle massive data streams in *data mining* applications (see [112] for a survey). A such a *scaling-up* method was proposed by Domingos and Hunten in [37, 61]. Their method is applicable to essentially any induction algorithm based on *discrete search*. Thus, this is suitable for learning the structure of BNs using a *hill-climbing learning procedure*. Their methods learn structures with a desirable quality using the minimum number of examples. In order to determine this minimum number, Domingos and Hunten derived some bounds based on the *Hoeffding* bound [60] - a statistical bound that gives the number of examples needed to obtain a "*good*" empirical estimate of a random variable (within ϵ of its true value). Therefore, at each search step, the algorithm uses only as much data from the stream as required to preserve the desired global quality. As a result, the model is built as fast as possible, using the minimum possible data.

As pointed out in [64], using a sample from the database can speed up the data mining process, but this is only acceptable if it does not reduce the quality of the mined knowledge. Thus, the crucial question in *sampling design* is how to determine the optimal sample size. When the number of available instances N is not sufficiently large, the plateau, and even the entire middle portion, can be missing from curves. Otherwise, when N is sufficiently large, the plateau region can constitute the majority of curves. If we are able to determine the sample size, N_{min}, at which the learning curve reaches its final plateau, we can obtain the same accuracy by using only a sample of the data of the found size with a considerable reduction of the computational cost. However, as stated in [111], "*convergence detection remains a challenging problem on which significant research effort should be focused*".

Different theoretical approaches provide estimates for N_{min}. Vapnik Chervonenkis theory [136] (also known as *VC-theory*) is the most comprehensive description of learning from examples from a statistical learning theory point of view. One of the most important concepts in which this theory relies is the *VC-dimension*. The *VC-dimension* allows to predict a probabilistic upper bound on the generalization error of a classifier as a function of the training error and the size of the training set. Therefore, there have been identified several

limitations in order to apply the derived bounds in the practice (e.g. see [14]).

We can, instead, empirically determine a statistical estimate of N_{min}. John and Langley in [64] have provided a comparison between the two main sampling strategies in order to choose the optimal sample size: *i*) *static sampling* where some statistical tests are used to decide whether a sample is sufficiently similar to the whole dataset (e.g χ^2 hypothesis for testing whether the sample and the large database come from the same distribution, the PCE criterion [64], etc.); *ii*) *progressive sampling* where the knowledge about the behaviour of the learning algorithm is used for determining N_{min}. It stars with a small sample and uses progressively larger samples until model accuracy no longer improves. One of the methods commonly used for monitoring the learning algorithm is the *extrapolation of learning curve* (ELC) method [14, 54, 95, 111]. The ELC method uses all the historical data to fit a parametric learning curve and then extrapolate the learning accuracy at the full length. Their effectiveness can be measured in terms of two performances: *i*) *fitting performance*: how well they fit a full-learning curve; *ii*) *predicting performance*: how well a fitted part-length curve (using a sample of the data) can predict (extrapolate) the learning accuracy at the full length. The models most widely used to fit learning curves are *power law models* (e.g. $y = a + b * x^{-c}$). An empirical study comparing six potentially useful models by fitting learning curves using C4.5 and a logistic discrimination learning algorithm was presented in [54]. Their results provides empirical support for applying the *power law model* to fitting learning curves for large datasets. A more recent version of the ELC method for convergence detection is proposed in [14]. The authors propose a method for an early assessment of the classification performance by detecting the point at which the learning curve starts *behaving well*, that is, where the learning curve becomes *monotonically decreasing* and the conditions of *discrete convexity* are met. When the learning curve reaches these conditions, the estimates of the future performance are computed.

5.5 Concept Drift Management

The future performance of adaptive learning systems depends not only on how the system is adapted, but also on the characteristics of the data it will predict. As argued in [62], machine learning systems can learn incorrect models when they erroneously assume that the

underlying concept is *stationary* if in fact it is *changing* or *drifting*. Concept drift scenarios require adaptive algorithms, able to track such changes and to quickly adapt to them.

In the last few years the notion of *concept drift* has received an increasing amount of attention in the literature and the research done has been applied in many real-world applications ranging from *network monitoring*, *information filtering*, *user modeling* to *data mining*. The basic idea underlying most concept drift trackers is that in changing environments recent data is more relevant than older one. Hence old examples (and hypotheses based on these) should eventually be *forgotten*, as they are examples of an old target concept that may be quite different from the concept one is trying to learn.

Several available concept drift trackers employ different approaches that include some control strategies for deciding whether adaptation is in fact necessary because a concept change has actually occurred. Deciding whether adaptation is necessary requires being able to detect changes first. Hence, coping with dynamic environments falls into two subtasks:

1. Detect concept drift;

2. Adapt to the changes, accordingly.

Our method to *handle concept drift* also follows such an approach. This relies on *Statistical Quality Control*. The methods provided in *Statistical Quality Control* are *on-line* or *in-process* quality control procedures that monitors an *on-going* production process. *Shewhart control charts*[5] [126] represent the basic monitoring tools of *Statistical Quality Control* for distinguishing *trends* and *out-of-control* conditions in a process. For handling concept drift, we propose to use a *Shewhart* `P-Chart` that monitors the behaviour of the batch error.

In the next section we describe some basic concepts related to the problem of *concept drift* in supervised learning. Then, in Section 5.5.2 we will explain how the `P-Chart` can be used as a monitoring tool for concept drift detection. In Section 5.5.3 we further present the general algorithm to handle concept drift based on `P-Chart`. Finally, in Section 5.5.4 we will provide an overview of related work.

[5]Shewhart control charts are named after W. A. Shewhart, a statistician at the AT&T Bell Laboratories, who is generally credited as being the first to introduce the control charts.

5.5.1 Concept Drift in Supervised Learning

In many real-world applications, when the data is collected over an extended period of time, the target concepts are often not stable but change with time. A typical example is *information filtering*, which concerns the adaptive classification of documents with respect to a particular user interest [70]. The *user's interest* can change over time and a filtering system should be able to adapt to such concept changes. Another example is the *customers' buying preferences*, which may also change with time [135]. For classification systems, which attempt to learn a discrete *target function* given examples of its inputs and outputs, this problem takes the form of changes in the *target function* over time and is known as *concept drift*.

Concept drift can more formally be defined as follows. Assume that the data generation itself is time-dependent. Let $\Omega_{\mathbf{X}}$ be the input space, i.e., the space of all possible examples, $\Omega_{\mathbf{X}} = \Omega_{X_1} \times \ldots \times \Omega_{X_n} \subset \Re^n$. Let Ω_C be the output space, i.e., the space of possible classes. Let $f : \Omega_{\mathbf{X}} \rightarrow \Omega_C$ be the target function to be learned over time.

Definition 44. Concept drift represents the changes in the target function $f : \Omega_{\mathbf{X}} \rightarrow \Omega_C$ over time.

As stated in section 3.2, the *target function* is called *target concept* in the particular case of *binary classification*, that is, the task of classifying the members of a given set of objects into two groups on the basis of whether they have some property or not. To better illustrate the concept drift problem in the context of concept learning we introduce an artificial problem, referred to as the "STAGGER" concepts [124], which has become a standard benchmark for testing *adaptive learning systems* that deal with concept drift.

In this problem, the domain objects are described by three attributes, each of them, taking on three values:

(A1) `size` $= \{small, medium, large\}$

(A2) `color` $= \{red, green, blue\}$

(A3) `shape` $= \{circle, triangle, rectangle\}$

Thus, there are 27 possible object descriptions in the representation space. 120 training instances are presented to the learning algorithm with the target concept changing every 40 steps. The STAGGER concepts are then defined as a sequence of three target concepts:

(C1) $(\texttt{size} = small) \wedge\ (\texttt{color} = red)$

(C2) $(\texttt{color} = green) \vee\ (\texttt{shape} = circular)$

(C3) $\texttt{size} = (medium \vee\ large)$

Figure 5.4[6] depicts the visualization of these three target concepts. Figure 5.5 shows the results from the application of one adaptive algorithm to cope with concept drift using the Naïve Bayes (described in [80]) against its *non-adaptive* version. After a concept drift occurs, the performance of both algorithms suffer a significant deterioration. However, the adaptive algorithm shows a better *recoverability* capability than its *non-adaptive* version, trying more quickly to improve its performance back.

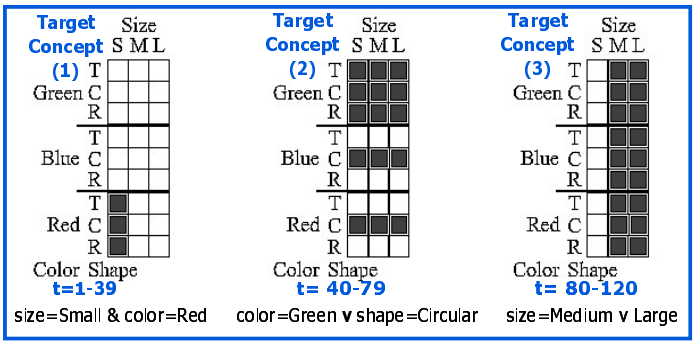

Figure 5.4: Visualization of the STAGGER concepts

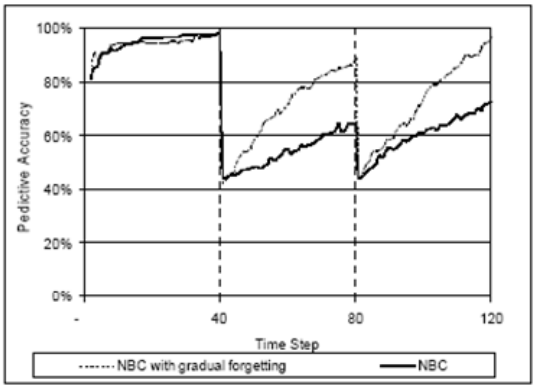

Figure 5.5: Predictive accuracy of Naïve Bayes when a concept-drift handler algorithm is used against its non-adaptive version for the STAGGER domain

[6]Figure 5.4 is taken from [94] and Figure 5.5 is taken from [80].

Real Concept Drift *versus* Virtual Concept Drift

Concept drift is usually referred to as unforeseen changes in the distribution underlying the data that can also lead to changes in the target concept over time, that is, to *real* concept drift. The term *virtual* concept drift is more oftenly associated to changes in the data distribution in the input space. *Virtual concept drift* can occur when the order of training instances for learning is skewed, so that different types of instances are not evenly distributed over the training sequence [142]. This kind of drift is also referred to as *population drift* or *sampling drift* [122]. Virtual concept drift can occur alone. For instance, in the context of a spam categorization task, while our understanding of an unwanted message may remain the same over time (i.e. the target concept remains constant), the relative frequency of different types of spam may change drastically with time which can lead to the necessity in adapting the current model [135]. However, both *virtual* and *real* concept drift can more frequently occur together. In information filtering, for example, both the *user's interest* and the *document content* can change over time. Since unforeseen changes in the data distribution can lead to changes in the target concept, from the practical point of view it is not important to make any distinction. In both cases, the current hypothesis needs to be adapted anyway.

Hidden Contexts

Another aspect of the concept drift problem in many real-world domains is that the concept of interest may depend on some *hidden context*, not given explicitly in the form of predictive features. Daily experience shows that in the real world the meaning of many concepts can heavily depend on some given context such as *season*, *weather*, *geography*, and so on. Changes in the context can be hidden, and can induce more or less radical changes in the target concept [143]. A typical example is weather prediction rules that may vary radically with the season. Another example [135] is the patterns of *customers' buying preferences* that may change with time, depending on the current day of the week, inflation rate, etc. Typically, context changes are *gradual*, whereas changes in user preferences may be *abrupt*. Hidden changes in the context may not only be a cause of changes of the target concept, but may also cause a change of the data distribution in the input space. Thus, the problem of tracking drifting concepts can be viewed as the problem of tracking context changes over time.

Concept Drift *versus* Concept Shift

Let us briefly introduce some useful definitions related to the *concept drift problem*[7].

Definition 45. The *extent of drift* is the degree of dissimilarity of successive concepts, quantified in terms of the relative error between the concepts.

Depending on the *extent of drift*, concept changes can be divided into *abrupt (sudden)* changes and *gradual* changes. We further follow the definitions proposed in [70] and use the term *concept shift* to represent abrupt changes and the term *concept drift* to represent gradual changes. It is more difficult to deal with *concept drift* than with *concept shift*. Abrupt changes lead to an abrupt deterioration of the predictive accuracy, and it is easier to detect and to remedy. In this case, it would be more appropriate to learn a new model with new data, thus forgetting all the old data as they are examples of the old target concept.

On the contrary, *gradual concept changes* lead to a gradual deterioration of the predictive accuracy. As pointed out in [143], when concepts will change gradually, it creates a period of uncertainty between stable phases where both the old and new concepts appear. We define this period as the *drift phase*. The new concept only gradually takes over, and some examples may still be classified according to the old concept. The rate at which gradual changes occur affects the ability to detect changes. Depending of the *drift rate*, gradual drifts can be further divided into *moderate* and *slow* drifts. While greater it is the rate, more gradual it is the drift. Thus, recognizing the *drift rate* is also an important feature of adaptive learning systems that deal with concept drift. Widmer and Kubat in [143] introduced the definition of the *speed of drift* to model scenarios with gradual concept changes, which they found to be a more natural dimension for practical scenarios that the notion of *drift rate*.

Definition 46. The *speed of drift* is the time it takes for a new concept to completely take over in a *drift phase*.

Figure 5.6 depicts the function α that model the *speed of drift* for gradual changes as defined in [143]. The function α represents the degree of dominance of the old concept A over the new concept B. $\alpha = 1$ means that A is fully in effect, $\alpha = 0$ means that B has completely taken over. The x axis represents the number of examples processed so far. Assuming that the

[7]For a more in depth reading the reader is referred, for instance, to the work of Widmer and Kubat [143].

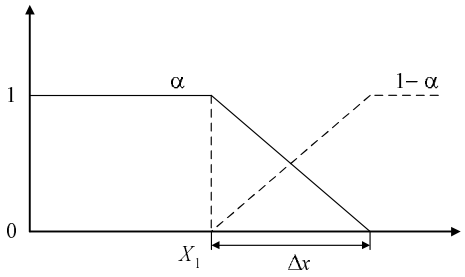

Figure 5.6: The function α to model the speed of drift for gradual changes

examples arrive at constant intervals this can be regarded as a dimension of time. $X1$ is the point where the concept begins to drift. The slope of the function α can be characterized by Δx (the *drift rate*) defined as the number of training instances until α reaches *zero*. Between $X1$ and $X1 + \Delta x$, $\alpha * 100\%$ of the examples belongs to the old concept A, and, what is obvious, $(1 - \alpha) * 100\%$ belongs to the new concept B. Further, in Section 6.2.3, we will describe how we make use of the function α to generate artificial concept drift scenarios for evaluating our adaptive algorithms.

Concept Drift *versus* Noise

Another difficult problem in handling concept drift is distinguishing between *true concept drift* and *noise*. Some trackers may overreact to noise, erroneously interpreting it as concept drift, while others may be highly robust to noise, adjusting to changes too slowly [135]. As argued in [143], an ideal adaptive strategy should combine robustness to noise and sensitivity to concept drift.

5.5.2 Using `P-Chart` for Detecting Concept Drift

The main idea behind *Statistical Quality Control*[8] is to monitor the stability of one or more quality characteristics in production processes. The values of the quality characteristic generally show some variation, which can be caused by either some "*natural causes*" inherent in the production process or by some "*special causes*" that can be traced to a particular problem. Whereas "*natural causes*" are presented all the time, "*special causes*" can occur

[8]For a more in-depth study of Statistical Quality Control the reader is referred to the books [36, 99].

at unpredictable times. A process can be run in either of two mutually exclusive states: an *in-control* state or an *out-of-control* state. An *in-control* state means that the successive values of the quality characteristic, as they are observed over time, show a stable random variation about a target value (variations caused by "*natural causes*"). Otherwise a process is *out-of-control*. A process is in *statistical control* if "*special causes*" have been detected and removed, so these sources of variability will not influence the process in the future. The *Shewhart controls charts* [126] are a useful process monitoring technique that helps distinguish whether a process is *in-control* or *out-of-control*. If an *out-of-control* situation is detected then the process can be corrected by removing the sources of variability.

Figure 5.7 shows a Shewhart control chart. The values of some quality characteristic v_t are plotted on the chart in time order and connected by a line. The chart has a *center line* (`CL`), a *upper control limit* (`UCL`) and a *lower control limit* (`LWL`). Points that fall outside the control limits mark statistically significant changes in the process. If a characteristic value v_t falls outside the control limits, it is assumed that the process is *out-of-control*, that is, some "*special causes*" have shifted the process off target. In addition, the control chart can include an *upper warning limit* (`UWL`) and a *lower warning limit* (`LWL`), which help to increase its sensitivity. Warning limits are usually set somewhat closer to the `CL` than the *control limits*. If a characteristic value falls outside the warning limits but is still inside the control limits, the process might still be in control. However, this situation can also indicate a trend towards a conceptual change. To make use of the warning limits, the position of successive characteristic values is often considered. For example, a certain number of successive warnings can indicate that the process is *out-of-control* and thus trigger a remediation action.

If the distribution of the characteristic values v_t is *Normal* (or *approximately Normal*) with mean μ and variance σ^2, then approximately $99,7\%$ of the observations will fall within 3σ of the mean of the statistics. Therefore, the use of *three-sigma* control limits is a reasonable choice. Assume that both the mean μ and the standard deviation σ can be estimated on the basis of some historical data. If μ and σ are known we can use them to set the lines of the

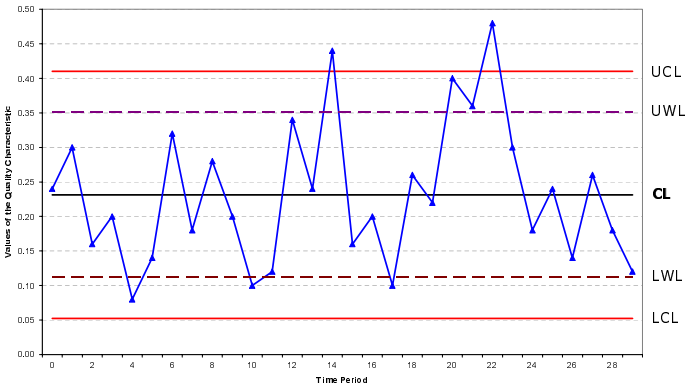

Figure 5.7: A Shewhart control chart

control chart, as follows:

$$\texttt{CL} = \mu, \tag{5.7}$$

$$\texttt{LCL} = \mu - 3\sigma; \quad \texttt{UCL} = \mu + 3\sigma, \tag{5.8}$$

$$\texttt{LWL} = \mu - \alpha\sigma; \quad \texttt{UWL} = \mu + \alpha\sigma; 0 < \alpha < 3. \tag{5.9}$$

In general, note that smaller values of α increase the risk of a false alarm, that is, indicating change when there is none.

The `P-Chart` Control Chart

The *Shewhart controls charts* are classified according to the type of quality characteristic that they monitor: *variables* or *attributes*. Attribute data is also known as *count variable*. We further focus on the `P-Chart` - an *attribute control chart* that monitors the sample proportion of a *count variable*. The statistical principles underlying the `P-Chart` are based on the *binomial distribution*.

The binomial distribution describes the behavior of a count variable X if the following conditions are met: *i*) each observation is the realization of a Bernoulli random variable with one of two outcomes (e.g. *success* or *failure*) and parameter p (e.g. the probability of "*success*"); *ii*) the successive observations are independent each other; *iii*) the number of observations n is fixed. If these conditions are met, then X has a *binomial distribution* with parameters n and p. Suppose X is the count of *successes* in a group of n observations and p

represents the probability of "*success*". The *sample proportion* of successes $\hat{p}$ is then defined as the ratio of the count random variable X to the sample size n, that is, $\hat{p} = \frac{X}{n}$.

If we know that X has a binomial distribution with mean np and variance $np\,(1-p)$, then the distribution of the random variable $\hat{p}$ can be obtained from the binomial distribution. By the multiplicative properties of the mean, the mean of the distribution of $\hat{p}$ is equal to the mean of X divided by n, or $np/n = p$. This proves that the sample proportion $\hat{p}$ is an *unbiased estimator* of the population proportion p. Thus, the *mean* and *standard deviation* of $\hat{p}$ can be obtained from the parameters p and n of the binomial distribution as follows:

$$\mu = p \;;\; \sigma = \sqrt{\frac{p(1-p)}{n}} \tag{5.10}$$

If the sample sizes are large ($n \geq 30$), then the distribution of both the *count variable* X and the *sample proportion* $\hat{p}$ approaches the *Normal* distribution, a result derived from the *Central Limit Theorem*. There are therefore statistical arguments for applying the *three-sigma* control limits to the `P-Chart`. Suppose that we can obtain an estimate $\hat{p}$ of the population proportion p from previous data. By replacing Equations 5.10 into Equations 5.7–5.9 we obtain the equations for the lines of the `P-Chart` for each individual t^{th} sample with size n_t as follows:

$$\texttt{CL} = \hat{p}\,, \tag{5.11}$$

$$\texttt{UCL} = \hat{p} + 3\sqrt{\frac{\hat{p}(1-\hat{p})}{n_t}};\;\; \texttt{LCL} = max\left\{0, \hat{p} - 3\sqrt{\frac{\hat{p}(1-\hat{p})}{n_t}}\right\}, \tag{5.12}$$

$$\texttt{UWL} = \hat{p} + \alpha\sqrt{\frac{\hat{p}(1-\hat{p})}{n_t}};\;\; \texttt{LWL} = max\left\{0, \hat{p} - \alpha\sqrt{\frac{\hat{p}(1-\hat{p})}{n_t}}\right\}, 0 < \alpha < 3 \tag{5.13}$$

Further we call $\hat{p}$ the *target value*. An usual procedure to estimate the target value is to use the weighted average of m preliminary sample proportions (as a rule, m is taken to be 20 or 25).

Using the `P-Chart` for Handling Concept Drift

In quality control, the `P-Chart` is usually used to monitor the proportion of *nonconforming* items in a production process. For a learning process, we propose to use the `P-Chart` for monitoring the *batch error* Err_B - the sample proportion of the *misclassified* examples in a

given batch of examples. As pointed out in [85], using batches of examples and the *batch error* as an indicator of the current performance offers the advantage to detect changes by observing the deviation from the current *batch error* value to the previous observed values without paying to much attention to *outliers*. The following crucial question in the design of the `P-Chart` for learning purposes thus arises: *How does one estimate the target value* $\hat{p}$ *to set the* center line?

The sequence of observed batch errors $p(t) = Err_B^{(t)}$ represents a discrete random process which has some inherent variation. In classical *Shewhart control charts* it is assumed that successive sample proportions should exhibit a stable random variation around the *target value* over time. However, such a behavior is not observed in a *prequential learning scenario* where it is assumed that concept changes are likely to happen. In this scenario, the data stream can be analyzed as a sequence of different *stable phases* between *drift phases* over time. On the other hand, as argued above we are interested in controlling the performance of the current model using the current structure by the means of the current *model-LC* without taking into account the error committed in the past, when other structures were used for classification. Suppose that at time t a new structure begins to be used. At the beginning, while the learner is still able to learn using the current structure, the successive batch errors should exhibit a *downward trend* that reflects the steeply sloping part in the *model-LC*. At once a concept change occurred, an opposite, *upward trend* in the successive batch errors is immediately observed. In principle, a learning process is *in-control* only when it becomes exhibit stable random fluctuations around a *resistance level*, that is, when the performance has reached its *plateau*.

Based on these facts we propose to dynamically estimate the target value taking into account the natural behaviour of the learning process. Since all the time when a lower value of the current model error Err_S is achieved, the learner will try to improve, or at least, to keep its performance level, we propose to maintain a *minimum value* of the model error Err_S and set the target value $\hat{p}$ to this minimum value instead of using some average of previously observed values. We denote the minimum value by Err_{min} and proceed in the following way. Whenever a new structure S is found, Err_{min} is initialized to some big number. Then, at each time step t if $Err_S^{(t)} + SErr_S^{(t)} < Err_{min}$ then Err_{min} is set to $Err_S^{(t)}$, where $SErr_S^{(t)}$ is the standard deviation of the current model error. Taking into account the way we estimate

the target value, we can state that our `P-chart` is not a typical statistical chart since we don't use a statistical well-founded estimator to estimate the target value.

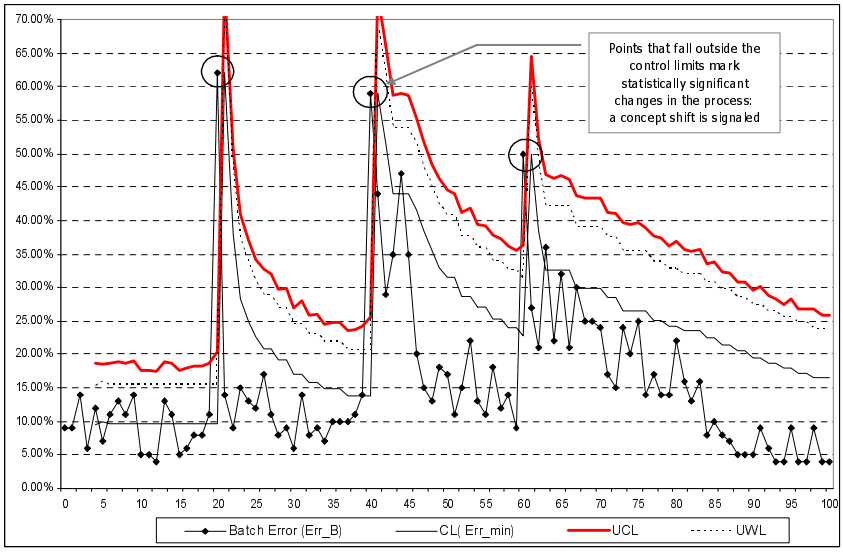

Figure 5.8: A `P-Chart` for detecting concept drift

Figure 5.8 illustrates an example of a `P-Chart` for detecting concept drift. At each time point t, $\hat{p}$ is set to Err_{min} and the P-Chart's lines are computed using Equations 5.12 and 5.13. Since a low error is always desirable, we do not need to use the low limits here. In this implementation, we use 2-sigma *warning limits* (that is, set $\alpha = 2$). 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`, then a *concept shift* is signaled. If for the first time $p(t)$ falls between the `UCL` and the `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 `UWL` we assume that the learner is *in control*. Algorithm 12 describes the pseudo-code of the method for detecting concept drift using the `P-Chart`.

5.5.3 The Concept-Drift Handler Algorithm

Algorithm 13 describes the pseudo-code of the adaptive algorithm for handling concept drift, which is independent of the classifier used. In each time step, the algorithm evaluates the current batch error $Err_B^{(t)}$ and the current `P-Chart` is then used to asses the current state of the learning system using Algorithm 12. The *adaptive strategy* mainly consists of manipulat-

Algorithm 12 The algorithm for detecting concept drift using the `P-Chart`

Require: A current time point t, a current *batch error* p(t)=Err_B^t, a current `P-Chart`, a number of examples in the batch n_t, a value to set the warning line α

Ensure: The current state of the `P-Chart`, the updated `P-Chart`

Add the new point $(t, p(t))$ to the `P-Chart`
Update Err_{min}
$CL \Leftarrow \texttt{Err}_{\texttt{min}}$ {set the center line}
$Sigma \Leftarrow sqrt(CL * (1 - CL)/n_t)$
$UCL \Leftarrow CL + 3 \times Sigma$ {set the control line}
$UWL \Leftarrow CL + \alpha \times Sigma$ {set the warning line}
// observe where p(t) falls
if $p(t) > UCL$ **then**
    state $\Leftarrow$ `CONCEPT SHIFT`
else if $p(t) > WCL$ **then**
    if LastAlert=t-1 **then**
        state $\Leftarrow$ `CONCEPT DRIFT` {for two or more consecutive alerts}
    else
        state $\Leftarrow$ `CONCEPT DRIFT ALERT`
else
    state $\Leftarrow$ `IN-CONTROL` {no significative changes was detected}
return the detected `state` and the updated `P-Chart`

ing a *short-term* memory, called `SHORT-MEMORY`, that stores those examples that we suspect to belong to a new concept different from the current one. Whenever an abrupt or gradual *concept drift* is detected, the examples of the current batch are added to the `SHORT-MEMORY`; otherwise, the `SHORT-MEMORY` is cleaned. Only if an abrupt change is suspected, that is *concept shift* is signaled, the adaptation process is triggered from its initial level. The examples of the `SHORT-MEMORY` are used to re-build a new hypothesis. Afterwards, the `SHORT-MEMORY` is cleaned for future uses. Otherwise, if for two or more consecutive times the warning situation was detected, that is, *concept drift* is signaled, the adaptation process is temporarily stopped, which means, that the new examples are not used to update the current model. This way we force a great degradation of the performance. As a result, the successive batch errors will more quickly jump outside the control line and the `P-Chart` will more quickly be able to signal a *concept shift* thus forcing to build a new model. Finally, only if the learner is *in control* or a *concept drift alert* is signaled, then the current model is updated with the examples of the current batch according to the updating method employed for each particular classifier. In the case when k-DBCs are used, we can proceed with Algorithm 11 to update

the classifier with new data according to the current behavior of the `model-LC`.

Algorithm 13 The adaptive algorithm for handling concept drift

Require: A current classifier $h_{\mathcal{C}}$, a current time point t, a batch B with new incoming examples, a current `P-Chart`, a short-term data memory `SHORT-MEMORY` with a portion of the past examples that it is suspected to belong to a new concept

Ensure: The updated classifier $h_{\mathcal{C}}$ taking into account the changes in the target concept

1: predictions $\Leftarrow$ predict(B, $h_{\mathcal{C}}$)
2: observed $\Leftarrow$ getFeedback(B) {get feedback}
3: $Err_B^{(t)} \Leftarrow$ evaluate(predictions, observed)
4: state $\Leftarrow$ getPChartState(t, Err_B, `P-Chart`, ...){use Algorithm 12}
5: **if** `state` is CONCEPT SHIFT **then**
6: Add B to `SHORT-MEMORY`
7: AdaptiveAction($h_{\mathcal{C}}$, `SHORT-MEMORY`, `INITIAL LEVEL`) {*build a new hypothesis*, see Alg. 15}
8: Clean `SHORT-MEMORY`
9: **else if** `state` is CONCEPT DRIFT ALERT $\vee$ CONCEPT DRIFT **then**
10: Add B to `SHORT-MEMORY`
11: **else**
12: // state is IN CONTROL
13: Clean `SHORT-MEMORY`
14: **if** state is IN-CONTROL $\vee$ CONCEPT DRIFT ALERT **then**
15: update($h_{\mathcal{C}}$, B) {update the classifier with the new data (classifier-dependent), use Algorithm 11 for k-DBCs}
16: **return** the updated classifier $h_{\mathcal{C}}$

5.5.4 Related Work

Motivated by the pioneering work on the STAGGER algorithm [124], several available *online adaptive systems* have implemented different forgetting mechanisms over either *the model memory* or the *data memory* in order to deal with *concept drift* [21, 48, 62, 68, 69, 70, 80, 85, 94, 122, 142, 143]. Forgetting as a means of adjusting to concept drift have been used through two main techniques: *i*) *weighted examples* - the weight of an example decreases as a function of its age, which is based on the simple idea that the importance of an example should decrease with time; *ii*) *time windows* - a *partial-memory* model where only the examples from a *window* that moves over recently incoming examples are used to induce the current hypothesis. The FLORA family of algorithms proposed by Widmer and Kubat in [142, 143] is of particular relevance among the *window-based* approaches. Other relevant algorithms based on *time windows* have been proposed, for instance, in [68, 69, 70, 85, 94, 122].

The main issue in *window-based* approaches is on choosing an appropriate window size.

A small window can assure a fast adaptability in phases with concept changes but in more stable phases it can affect the learner performance. A large window would produce good and stable learning results in stable phases but cannot react quickly to concept changes. In the simplest case the window is of fixed size. More complex algorithms, instead, use some heuristic that allow adjusting the window size according to the current extent of concept drift. For instance, the window adjustment heuristic introduced in the FLORA algorithm in the work [143] 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 in the domain of *information filtering* that have provided the basis for our drift detection method are the work of Klinkenberg and Renz [70] and the work of Lanquillon [85]. To detect concept drift, Klinkenberg and Renz proposed a window adjustment heuristic that monitors the values of three performance indicators: *accuracy*, *recall* and *precision*. At each time step, the *mean* μ and the *standard deviation* σ are computed for each of these indicators based on the last M batches (where M is a predefined parameter). Each current indicator value then is compared to the confidence interval $\mu \pm \alpha \times \sigma$ ($\alpha > 0$), where the confidence level α is a user-defined constant. If the current indicator value is smaller than the *lower end point* of this interval, a concept change is suspected, which is equivalent to use an α-sigma `LCL` limit in a *Shwehart* control chart. In this case, a further test determines whether the change is abrupt (*concept shift*) or rather gradual (*concept drift*). If the current indicator value is smaller than its predecessor β times (a user-defined constant such that $0 < \beta < 1$), a *concept shift* is suspected; otherwise a *concept drift* is signaled. If a *concept shift* is detected then the window is reduced to its minimal size, that is, the size of one batch ($|B|$), thus dropping the no longer representative old examples as fast as possible. If a *concept drift* is recognized, the window is reduced less radically by a user-defined reduction rate γ, ($0 < \gamma < 1$). This way some of the old data is kept, because it still is at least partially representative for the current concept.

Similar to our approach, Lanquillon [85] employs *Statistical Quality Control* to detect concept changes in the context of an adaptive information filtering system. One of the main problems when performance indicators are monitored for tracking concept changes is that indicators which are based on classification results generally require the true class labels

in order to be evaluated. His methodology using quality control in information filtering attempted to detect changes without expensive user feedback and to adapt a filtering system only if necessary to maintain classification performance. Three performance indicators are monitored using two control charts but not the `P-Chart`: *i*) the *sample error* (similar to the batch error Err_B); *ii*) the *expected error rate*; and *iii*) *virtual rejects*. The *sample error* requires only partial feedback while the two last measures don't require any feedback. A representative training set is maintained through storage of new examples for which the true class labels have been provided by the user. If the monitor has detected some changes, the filtering system is adapted based on the current training set by running the learning process from scratch. The target value in the control chart is estimated by using the weighted average of the indicator values on recent batches only if they are within the warning limits of the chart.

In a previous work [21] we explored how two alternatives `P-Chart`s (we called them `PAvg` and `PMin`) can be used to detect concept changes. These `P-Chart`s differ only by the way they estimate the *target value*. `PAvg` uses weighted averaging while `PMin` uses the minimum value of the error rate. We presented a general algorithm to handle concept drift based on the `P-Chart` in an on-line learning framework, which has served as base to our current proposal. The experimental results in the context of a user modeling prediction task using a Naïve Bayes classifier showed that both `P-Chart`s consistently recognize concept changes, and that, in general, the proposed method allows the learner to adapt quickly to these changes in order to maintain its performance level. However, for purpose of estimation of the target value we state that it is more appropriate to consider `PMin` than `PAvg` because: *i*) `PMin` doesn't require any parameter to be tuned; *ii*) `PMin` better follows the natural behaviour of the learning process.

Another drift detection method that controls the *error rate* of on-line learning algorithms also based on similar statistical principles related to the *binomial distribution* is proposed in [48]. The method was tested with a set of artificial datasets and a real world dataset by using three learning algorithms: a *perceptron*, a *neural network* and a *decision tree*. The experimental results showed that this method improves the learning capability of the algorithm when modeling *non-stationary* problems.

5.6 The Adaptive Prequential Learning Framework

In this section we describe the `AdPreqFr4SL` as a whole learning framework that integrates all the above described control and adaptive strategies for handling *cost-performance* and *concept drift*.

Algorithm 14 The algorithm of the adaptive prequential framework for supervised learning

Require: A classifier class-model $\mathcal{M}$, a dataset $\mathcal{D}$ of *i.i.d.* labelled examples $< \mathbf{x}, c = f(\mathbf{x}) >$ divided in batches B of m examples

Ensure: A classifier $h_{\mathcal{C}} \in \mathcal{M}$ updated at each time point

1: Initialize $h_{\mathcal{C}}$ with one of the hypothesis from M
2: **for** each batch B of m examples of $\mathcal{D}$ **do**
3: **for** each example $\mathbf{x}$ in B **do**
4: $h_{\mathcal{C}}(\mathbf{x}) \Leftarrow$ predict($\mathbf{x}$, $h_{\mathcal{C}}$)
5: $f(\mathbf{x}) \Leftarrow$ getActualClass($\mathbf{x}$)
6: `numIncorrected`$+ = \delta(\mathbf{x}, f(\mathbf{x}), h_{\mathcal{C}}(\mathbf{x}))$ {the 0-1 loss is used}
7: indicators $\Leftarrow$ assesIndicators(`numIncorrected`, ...)
8: state $\Leftarrow$ estimateState(indicators, monitoring-tools)
9: adapt($h_{\mathcal{C}}$, B, state)
10: **end for**
11: **return** $h_{\mathcal{C}}$

As stated, 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. We assume that at each time point data arrives in batches and the main goal is to sequentially predict the classes of the next batch. Moreover, we maintain an *unique hypothesis* $h_{\mathcal{C}}$ defined as a pair (S, Θ_S), where S is the structure and Θ_S are the parameters for that structure. In order to achieve a desirable performance even when dealing with concept drift, the `AdPreqFr4SL` includes some *monitoring tools* that controls the value of some performance indicators.

Algorithm 14 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 actual class is observed and some performance *indicators* are assessed using the current classifications. Then, the indicator values are used to estimate the actual system's *state*. Finally, the model is adapted according to the estimated state.

5.6.1 Defining the Indicators, States and Adaptive Actions

Defining the Indicators and States

In the `AdPreqFr4SL` two performance indicators are monitored over time:

`I1` - the `BATCH ERROR` Err_B

`I2` - the `MODEL ERROR` Err_S

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

`S1` - `IS IMPROVING`: the performance is improving

`S2` - `STOP IMPROVING`: the performance stops improving

`S3` - `CONCEPT DRIFT ALERT`: a first alert of concept drift was signaled

`S4` - `CONCEPT DRIFT`: there is a gradual concept change

`S5` - `CONCEPT SHIFT`: there is an abrupt concept change

`S6` - `STABLE PERFORMANCE`: the performance achieves a plateau. It is the goal state.

In real on-line systems it is not always possible to obtain feedback for all the examples. In the `AdPreqFr4SL` we assume that for the most part of examples feedback will be obtained. In this case, we may draw a sample only from those examples of a batch for which we have the correct class and estimate, for instance, the batch error.

Defining the adaptive actions

According to the adaptation purposes we can discriminate adaptive actions into two groups:

1. adaptive actions for *incorporating new data* to the current model as described in Algorithm 15:

   `A1` - `UPDATE PARAMETERS`

   `A2` - `UPDATE STRUCTURE`

   `A3` - `AUGMENT DEPENDENCIES`

   `A4` - `BUILD MODEL`

2. adaptive actions for *handling concept drift* as described in Algorithm 13. These actions mainly consist of manipulating the *short-term memory* with those examples that we suspect belongs to a new concept

A5 - ADD EXAMPLES TO SHORT MEMORY

A6 - CLEAN SHORT MEMORY

Algorithm 15 The adaptive actions for incorporating new data to the current k-DBCs

Require: A classifier $h_{\mathcal{C}} = (S, \Theta_S)$ belonging to the class of k-DBCs, a batch B of labeled examples of $< \mathbf{X}, C >$, the **level** of adaptation, the current k value, the `kMax` value for the maximum allowable k, a boolean parameter `bIterativeBayes` to indicate whether to use Iterative Bayes or not

Ensure: An adaptive action over the classifier $h_{\mathcal{C}}$

```
1: if INITIAL level then
2:     k ⇐ 0 {A0: build a new model using NB}
3:     learnNaiveBayes(SHORT-MEMORY)
4: else if FIRST level then
5:     updateParameters(h_C, B, bUseIterativeBayes) {A1}
6: else if SECOND level then
7:     updateStructure(h_C, B, ...) {A2}
8: else if THIRD level then
9:     if k < kMax then
10:        k+ = 1 {A3}
11:    updateStructure(h_C, B, ...) {A2}
12: return the updated h_C
```

Deciding on the best adaptive actions

The `AdPreqFr4SL` is provided with some controlling tools for deciding the best adaptive actions according to the current learning goals. Control strategies for bias management are mainly based on the observation of the `model-LC`. The main goal is to detect when start adapting the current structure and stop the adaptation process. Adaptation to concept drift is based on the findings detected by the `P-Chart`. Thus, at each learning step, given the current state's estimate, a decision maker must select the best adaptive actions, such that the desired current goals can be achieved. We use a simple *rule-based* model for defining decision functions: IF we observe that state THEN do this adaptive action.

5.6.2 The Algorithm for Learning k-DBCs in the `AdPreqFr4SL`

Algorithm 16 depicts the pseudo-code of the algorithm for learning k-DBCs in the `AdPreqFr4SL` which handles the *cost-performance* trade-off and *concept drift*. The algorithm is provided with the values of five parameters: the `kMax` value for the maximum allowable degree of attribute dependence, a boolean variable `bIterativeBayes` to indicate whether to use Iterative

Bayes or not, the two thresholds used in the control criteria for bias management: `eps1` for the *gentle slope* and `eps2` for the *plateau* and the number of consecutive times `maxTimes` the Err_B does not decrease after parameter adaptation used in the alternative detection method based on the *batch error*. At each learning step, the system accepts a batch B of examples. The whole procedure can be summarized by the following main steps:

1. **Prediction**: classify the examples of B using the current hypothesis $h_{\mathcal{C}}$.

2. **Performance Estimation**: estimate the batch error $Err_B^{(t)}$ and the model error $Err_S^{(t)}$.

3. **Observation of the** `P-Chart`: add the new point $p(t) = Err_B^{(t)}$ to the `P-Chart` and recover its current state (using Algorithm 12). If *concept shift* is signaled, go to the *initial level* of adaptation. The examples of the `SHORT-MEMORY` are used to re-build a new hypothesis. Afterwards, the `SHORT-MEMORY` is cleaned for future uses. If a *concept drift* is detected (for two or more consecutive times the warning situation was detected), temporarily stop adapting. Otherwise, if the `P-Chart` is *in control* or a *concept drift alert* is signaled, then proceed with the next step.

4. **Observation of the** `model-LC`: add the new point $y(t) = Err_S^{(t)}$ to the `model-LC` and analyze the behaviour of its most recent points in order to verify if the conditions of *discrete convexity*, *non-increasing trend* and gentle slope are met. If the conditions are met, it is assumed that the performance no longer improves in a desirable tempo using the current structure and the *structure-updating procedure* is triggered, that is, go directly to Step 7. Otherwise, it is assumed that the performance is still improving and proceed with the next step.

5. **Parameter Adaptation**: perform the *first level* of adaptation, that is, only update the parameters with B using Algorithm 9.

6. **Observation of the Batch Error**: compare the batch error Err_B before and after parameter adaptation. If for the pre-defined number of consecutive times `maxTimes`, the batch error does not improve, then it is assumed that the performance no longer improves using the current structure. Move to the *second level* of adaptation, that is, go to the next step. Otherwise, it is assumed that the parameter estimates can be still

Algorithm 16 The algorithm for learning k-DBCs in the `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 NB classifier*, see Alg. 15}
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`){use Algorithm 12 for 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`) {use Algorithm 15: build a NB}
        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 5.4 are met}
        else
            state ← IS IMPROVING
    if `state` IS IMPROVING ∨ CONCEPT DRIFT ALERT **then**
        AdaptiveAction($h_{\mathcal{C}}, B$, `FIRST LEVEL`, `bIterativeBayes`){update parameters using Algorithm 9}
        if consecCounter($Err_B^{t_{\text{AFTER}-\text{ADAP}}} \geq Err_B^{t_{\text{BEF}-\text{ADAP}}}$) = `maxTimes` **then**
            `state` ← STOP IMPROVING
    if `state` STOPS IMPROVING **then**
        if $k > 0$ **then** AdaptiveAction(k-DBC, B, `SECOND LEVEL`,...) {update structure using Algorithm 10}
        if (not change(S) ∧ $k <$ `Maxk`) ∨ 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 the updated $h_{\mathcal{C}}$

improved using new data. The adaptation process using the batch B is completed and the updated hypothesis $h_{\mathcal{C}}$ is returned.

7. **Structure Adaptation**: if the current structure is not the NB structure ($k > 0$) then perform the *second level* of adaptation: adapt the current structure using Algorithm 10. If after updating the resulting structure remains the same or the input structure is the NB structure ($k = 0$), then move to the *third level* of adaptation: increment k by one and continue searching using Algorithm 10, now in the extended search space. If after adaptation, the resulting structure still continues the same, then verify the *stopping criterion*. If the *stopping criterion* is met, then stop doing further adaptations while no significant change in the performance will be observed. Return the updated $h_{\mathcal{C}}$.

5.7 Concluding Remarks

In this chapter we have provided new adaptive algorithms for BNCs into the unified framework `AdPreqFr4SL`, which attempts to handle the *cost vs. performance* trade-off and cope with *concept drift*. Instead of selecting a particular class of BNCs and using it during all the learning process, we propose the use of the class of k-DBCs and start with the simple NB ($k = 0$). 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 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. 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. The benefit of our method is that this is a simple, well-argued, statistically-driven method and independent of the learning algorithm, which makes it broadly applicable. The following chapter describes the results and analysis of conducted experiments that demonstrate the advantages of our adaptive algorithms in comparison against its non-adaptive versions.

Chapter 6

Experimental Evaluation

6.1 Introduction

We carried out a series of experiments in order to evaluate the adaptive algorithms for BNCs in the `AdPreqFr4SL`, using both, *artificially* generated domains and *benchmark problems* from the UCI repository [102]. 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*. Most of the *benchmark problems*, instead, are based on real-world domains, which allow us to test our adaptive algorithms in real-world problems.

To test the *bias management* capability we primarily investigated whether our adaptive algorithms are actually capable of adjusting the complexity of the current hypothesis to suit the available training data, thus attempting to select the optimal *class-model* for the available amount of training data. To this end we compared the BNCs induced by our adaptive algorithms under the `AdPreqFr4SL` described in Algorithm 16 against the Naïve Bayes (NB) classifier and several k-DBCs induced in the *prequential*, *non-adaptive revolutionary* learning scenario described in Algorithm 8. Since *revolutionary* updating is essentially optimal in terms of the quality of the induced classifiers, this kind of experiments allowed us to evaluate whether the adaptive algorithms are able to approach the performance of the *best* k-DBC (i.e. the k-DBC with the best performance) induced from scratch using all the data seen so

far at each learning step while considerably reducing the cost of updating.

To test the *concept drift management* capability we artificially generated five *concept shift scenarios* (CSSs) and five *concept drift* scenarios (CDSs) using randomly generated k-DBCs. Both, CSS and CDS 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 CDS (a 1-DBC for the first context, a 2-DBC for the second one, etc.). The main goal was to evaluate whether the `P-Chart` is able to consistently recognize concept changes, both *abrupt* and *gradual*, and to adapt quickly to these changes, thus verifying a good *recoverability capability*.

In all the experiments we present below we evaluated two versions of the adaptive algorithm for learning k-DBCs in the `AdPreqFr4SL` using Algorithm 16. We call these two versions Adap1 and Adap2. Adap2 additionally implements the Iterative Bayes procedure for improving the parameter estimates as described in Algorithm 6. To implement Adap1 and Adap2 we provided Algorithm 16 with the values of four parameters: the `kMax` value for the maximum allowable degree of attribute dependence and the three parameters used in the control criteria for bias management described in Section 5.4.2: *i*) `eps1` - the threshold for the gentle slope; *ii*) `eps2` - the threshold for the plateau; and *iii*) `maxTimes` - the number of consecutive times that Err_B does not decrease after parameter adaptation. 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 for this parameter 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 `eps1` for the *gentle slope* and `eps2` for the *plateau* were also set according to the domain complexity. The smaller the value of `eps1` the slower the model's complexity increases over time. The smaller the value of `eps2` the later the adaptation process is stopped. Intuitively, we chose lower thresholds for more complex domains. In Section 6.3.2 we will provide the results of a study that evaluates how different threshold settings can affect the behavior of the adaptive algorithms.

For almost the experiments we used batches of 100 examples except with generated CDSs where we used batches of 50 examples. In order to avoid the effect of example ordering, all the

indicators' values here presented were obtained as average values over a number of randomly generated samples. Finally, we compared the performance of different learning algorithms using different scores, which allowed us to verify whether our adaptive algorithms perform well independently of the score used.

6.2 Evaluation with Artificial Datasets

We performed three types of studies using artificially generated datasets:

- **Study I**: we simulated stability in the target function and focused on testing the aspects concerning the *bias management* capability of adaptive algorithms.
- **Study II**: we simulated *concept shift scenarios* where the target function changes abruptly at four times and tested the *concept shift management* capability of adaptive algorithms.
- **Study III**: we simulated *concept drift* scenarios where four gradual concept changes occur and focused on testing the aspects concerning the *concept drift management* capability of adaptive algorithms.

Dataset Generation

We generated datasets from randomly generated k-DBC models, which are composed by 9 binary attributes and a binary class node as follows:

1. We randomly generated five different classifiers for five k-DBCs class-models, varying k from 1 to 5. Thus, for each k value we generated five k-DBCs. We denote them by k-DBC-j, $j = 1 \ldots 5$.
2. From each generated k-DBC-j, we randomly generated 10 samples of 10100 examples, $\mathcal{D}_{k_j}^{(i)}, i = 1 \ldots 10$.

Overall, there are 25 different k-DBC-j models that were generated and for each k-DBC-j there are 10 datasets.

Parameter Settings

The following settings were intuitively chosen as threshold values and found to perform well for all the conducted studies with artificial datasets: `eps1`=0.05 and `eps2`=0.005.

6.2.1 Study I - Evaluating Bias Management Capability

Primarily, we want to investigate if our adaptive algorithms Adap1 and Adap2 are able to scale up the model's complexity of k-DBCs while improving their performance over time. With this aim, we carried out an empirical study comparing the performance of our adaptive algorithms against its non-adaptive versions, that is, against the NB and different k-DBCs induced from scratch at each learning step. We chose for this study only three scores among those that are most commonly used in learning BNs: one score that favours more simple structure (MDL), another that tends to favours more complex structures (BDeu) and a third score that favors models of intermediate complexity as the Bayesian score does. The Bayesian score is simple the marginal likelihood. We further call it Bayes or BD for short. Our main goal was to verify if the adaptive algorithms are able to perform no worse than the best k-DBC induced with the same score at each learning step while reducing the cost of updating. We demonstrate the results using just one of the five generated models in each k-DBC class-model. However, we observe similar results for all models.

Analysis of the Error Rate

Figure 6.1 shows the learning curves that depict the error rate for the five selected artificially problems, which were generated using different k-DBCs with $k = 1, 2, 3, 4, 5$. For each artificial problem and score, we compare the learning curves of the adaptive algorithms Adap1 and Adap2 against the learning curves of the NB and several k-DBCs induced with the batch algorithm at each learning step. In addition, to serve as baseline, we also show the learning curves obtained with the *true generative model* (True Model) and with a k-DBC induced by using the true structure and only incrementally learning its parameters (True Struct). Results show that for all the scores adaptive algorithms have the behavior expected. In most of cases, the learning curves approach the performance of the best k-DBC over time. Moreover, for more complex generative models, Adap2 outperforms Adap1. This may indicate that for

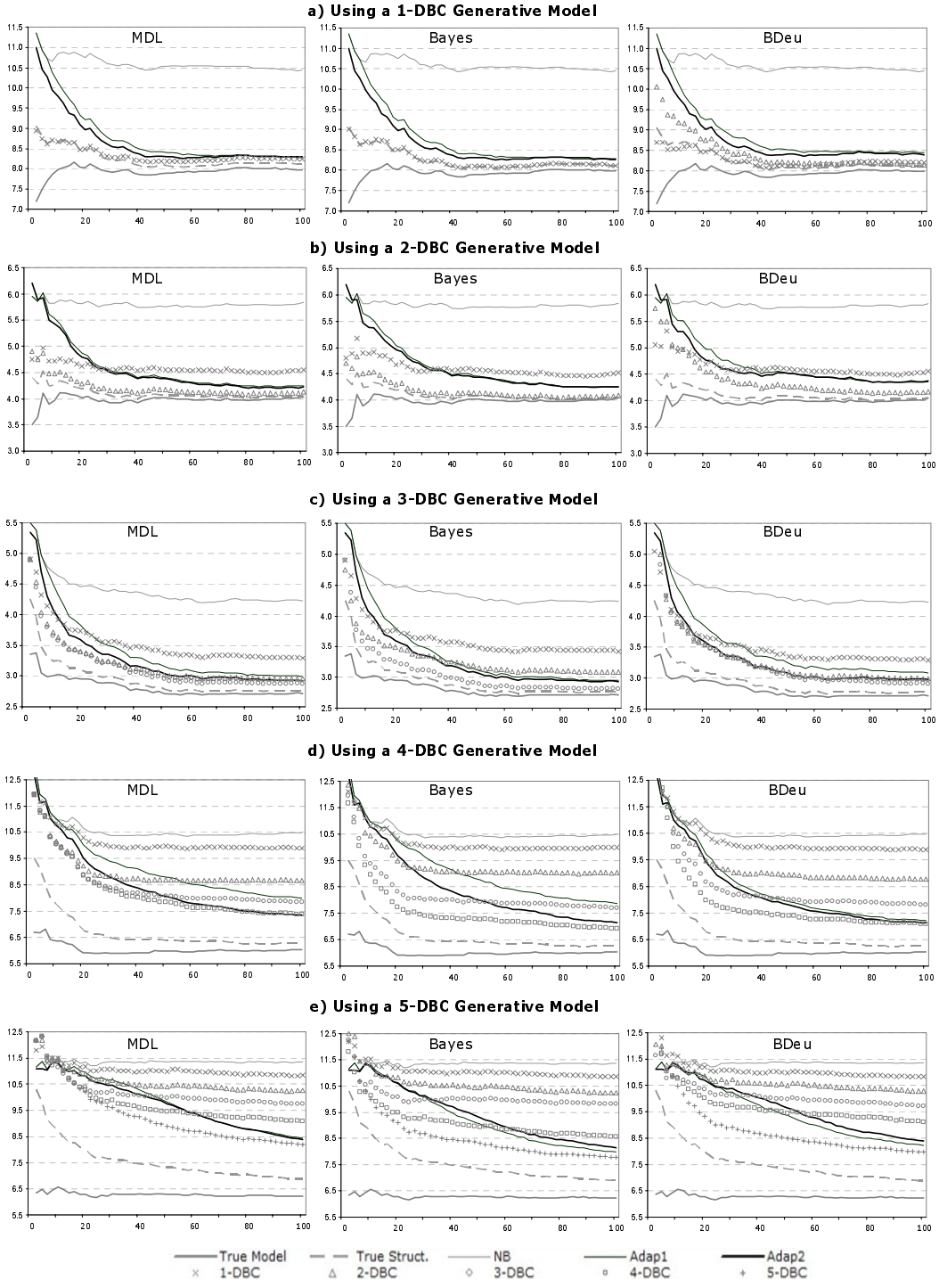

Figure 6.1: Error rate of Adap1 and Adap2 against NB, several k-DBCs, True Model and True Struct per generative k-DBC *class-model* and *score*

more complex domain by using IB for parameter refinement the AdPreqFr4SL can also improve the parameter estimates, which allows not only a reduction of the bias resulting from the *modeling error* but also a reduction of the bias resulting from the *estimation error*.

Analysis of the Model Error

As stated, one of the main goals in our adaptive framework is to control the performance of the induced k-DBCs without considering the error committed in the past. To this end, during all the learning process we control the behavior of the model error Err_S by means of its learning curve, model-LC, as described in Section 5.4.2. In Figure 6.2 we compare the model-LC of adaptive algorithms against the learning curves of the NB, the True Model the True Struct and several k-DBCs. Note that the *model error* is recomputed each time a new structure is used whereas the *error rate* is based on the *cumulative error* taking into account all the examples classified so far. We observe that the adaptive algorithms have the behavior expected, that is, the model-LC approaches the behavior of the best k-DBC induced with the batch approach at each learning step. Moreover, in some cases, the model-LC approaches the behavior of the True Model, a phenomenon that is more pronounced in scenarios with simpler generative models and when the MDL score is used.

Analysis of the Final Error

Table 6.2 shows the *batch error* of the last incoming batch of examples, which was not used to update the classifier, for the five selected artificially problems using different generative k-DBCs. These results were obtained at the last learning step, when 10000 training examples were used to induce the different classifiers. For each generative model and score we first show the error obtained with the NB, the True Model and the True Struct. Then we show the errors obtained with different k-DBCs (varying k from 1 to 5). The best results that give lower errors among the different k-DBCs are reported with bold text and are placed separately in line (4) in order to compare the results obtained with the adaptive algorithms Adap1 and Adap2 against the best k-DBC. The last lines of Table 6.2 show some comparative studies of the performance for a pair of approaches:

I - This study compares the reduction of the error obtained by using the true model (line

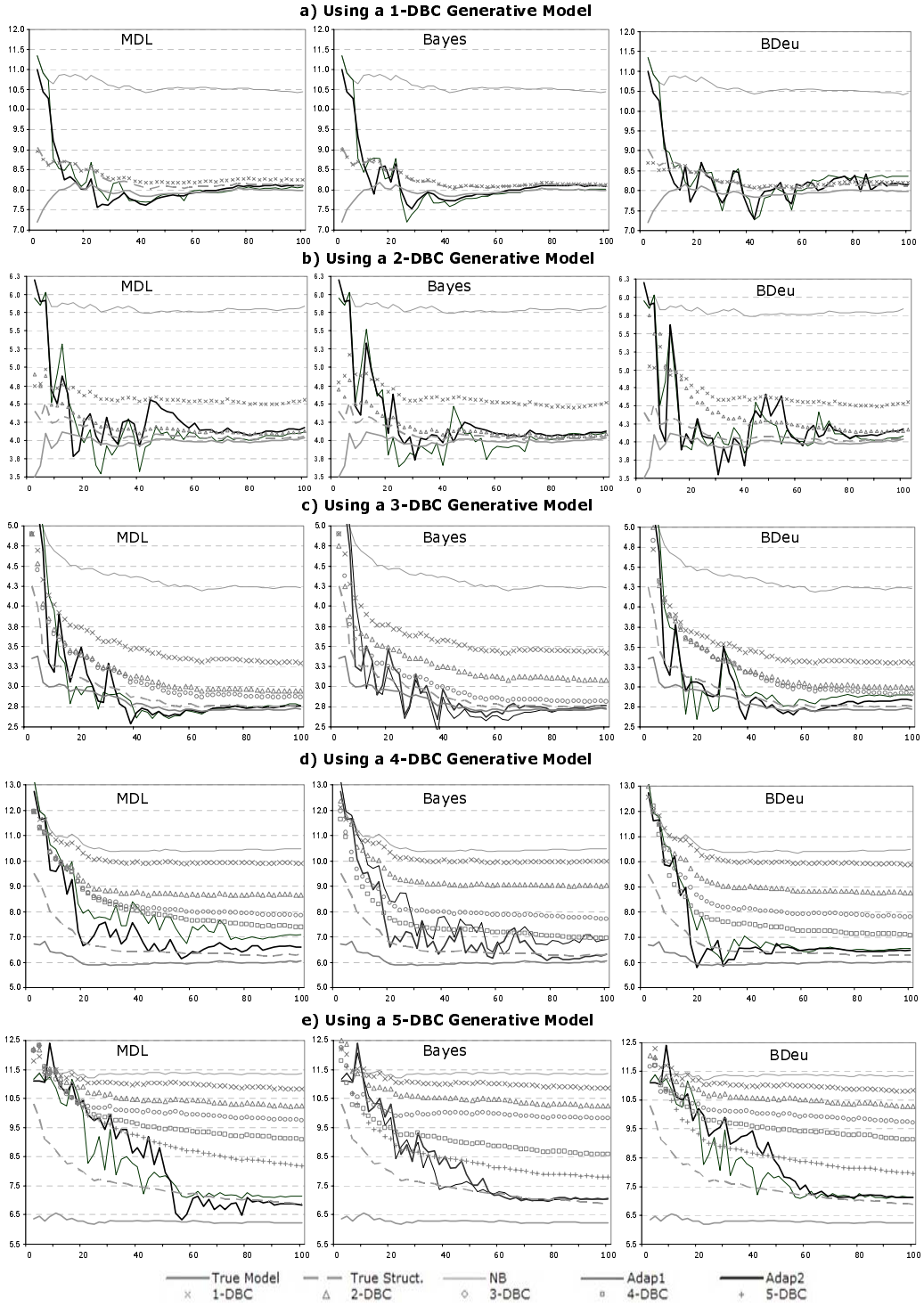

Figure 6.2: Model error of Adap1 and Adap2 against the error rate of NB, several k-DBCs, True Model and True Struct per generative k-DBC *class-model* and *score*

"(2)vs.(1)") or the `true structure` (line "(3)vs.(1)"), instead of using the NB. By looking at these differences we know what is the maximum possible range of reduction of the error of the NB which we can obtain by knowing the `true model` or the `true structure`. The last line "(3)vs.(2)" shows the differences between the final error of the `true structure` and the `true model`. These differences come mainly from the bias resulting from the parameter estimates in the classifier induced with the `true structure`.

II - This study summarizes the differences in the final error between the best k-DBC and the `true model`(line "(4)vs.(2)") or the `true structure` (line "(4)vs.(3)"), respectively. In general, the differences in the final performance are quite small. However, as we can verify by looking at the plots of the *error rate* and the *model error* in Figures 6.1 and 6.2, the best results over time were obtained for those learning scenarios where simpler generative models were used. In learning scenarios with a 4-DBC or a 5-DBC generative *class-model*, the differences in the performance's behavior between the best k-DBC and the classifier induced with the *true structure* become greater. Note that the best k-DBC was induced using a *hill-climbing search algorithm*, which approximates the optimal solution, that is, the *true structure*. The hill-climbing procedure can require more training data to better approximate more complex models, but it can also be trapped in local maximums during the searching process.

III - This study is the most relevant for the evaluation of the adaptive algorithms, since our main goal was to investigate whether Adap1 and Adap2 are able to approach the best k-DBC induced with the same underlying learning algorithm and score. This compares the results of Adap1 and Adap2 against the best k-DBC (lines "(5)vs.(4)" and "(6)vs.(4)", respectively). As stated, since learning from scratch uses all the data provided so far, the induced k-DBCs are essentially optimal in terms of the quality of the model if we choose the appropriate k value at each learning step. Specially for more complex generative models (e.g. a 4-DBC or 5-DBC class-model) and for the `BDeu` score we can observe significative differences in the results for different k-DBCs varying the k value. However, in most cases, the differences in the final error between an adaptive algorithm (Adap1 or Adap2) and the best k-DBC are quite small. In general, Adap1 and Adap2 not only approach the best k-DBC but, in some cases, they can outperform

it. Table 6.1 summarizes how many times the final error of the classifier induced with Adap1 and Adap2 was *less*, *equal* or *greater* than that of the best k-DBC.

Table 6.1: Number of times the final error of the adaptive algorithms was *less*, *equal* or *greater* than that of the best k-DBC for the five artificially generated problems

Adap1		Adap2	
Event	♯ Times	Event	♯ Times
(5)<(4)	3/15	(6)<(4)	6/15
(5)=(4)	4/15	(6)=(4)	2/15
(5)-(4) $\leq$ 0.5	8/15	(6)-(4) $\leq$ 0.5	7/15

IV - This study compares the reduction of the error obtained by using Adap2 against Adap1. In most cases the results show that a more significant reduction of the error can be achieved when the adaptive algorithm is combined with the Iterative Bayes procedure. By using `AdPreqFr4SL` with the Iterative Bayes, 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.

All the presented results give us some evidence that our adaptive algorithms are able to select an appropriate class-model (i.e. an appropriate k value) for the current amount of training data. Further we provide some results about the complexity of the induced models that will help us to corroborate this hypothesis.

Table 6.2: Error on the last incoming batch of examples per generative k-DBC *class-model*, *score* and *adaptive algorithm*

CLASS-MODEL		**1-DBC**			**2-DBC**			**3-DBC**			**4-DBC**			**5-DBC**		
(1)	NB		11.10			7.10			3.70			10.80			12.60	
(2)	True Model		8.50			4.80			2.50			6.50			6.30	
(3)	True Struct.		8.60			4.70			2.50			6.40			6.70	
	k-DBCs	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu
	1-DBC	8.70	**8.60**	8.80	5.50	5.60	5.40	2.60	2.70	2.90	10.40	10.90	10.30	11.80	11.80	12.00
	2-DBC	**8.60**	8.60	**8.50**	**4.70**	**4.70**	**4.80**	2.90	2.80	2.80	9.90	10.00	9.80	10.50	10.30	11.00
	3-DBC	8.60	8.60	9.00	4.70	4.70	4.90	**2.60**	**2.50**	2.80	8.10	8.30	8.00	8.90	9.40	8.70
	4-DBC	8.60	8.60	8.80	4.70	4.70	5.00	2.60	2.50	**2.70**	7.00	7.10	6.90	8.40	8.20	8.60
	5-DBC	8.60	8.60	8.90	4.70	4.70	5.10	2.60	2.50	2.70	**6.50**	**6.80**	**6.50**	**7.30**	**7.70**	**7.40**
(4)	Best k-DBC	8.60	8.60	8.50	4.70	4.70	4.80	2.60	2.50	2.70	6.50	6.80	6.50	7.30	7.70	7.40
(5)	Adap1	8.70	8.80	8.80	4.70	4.70	4.90	2.70	2.70	2.80	6.80	6.70	6.80	7.20	7.40	7.40
(6)	Adap2	8.40	8.40	8.90	4.90	4.70	5.10	2.70	2.80	2.70	7.00	6.60	7.00	7.10	7.30	7.00
I	(2) vs. (1)		-2.60			-2.30			-1.20			-4.30			-6.30	
	(3) vs. (1)		-2.50			-2.40			-1.20			-4.40			-5.90	
	(3) vs. (2)		0.10			-0.10			0.00			-0.10			0.40	
II	(4) vs.(2)	0.10	0.10	0.00	-0.10	-0.10	0.00	0.10	0.00	0.20	0.00	0.30	0.00	1.00	1.40	1.10
	(4) vs.(3)	0.00	0.00	-0.10	0.00	0.00	0.10	0.10	0.00	0.20	0.10	0.40	0.10	0.60	1.00	0.70
III	(5) vs (4)	0.10	0.20	0.30	**0.00**	**0.00**	0.10	0.10	0.20	0.10	0.30	**-0.10**	0.30	**-0.10**	**-0.30**	**0.00**
	(6) vs (4)	**-0.20**	**-0.20**	0.40	0.20	**0.00**	0.30	0.10	0.30	**0.00**	0.50	**-0.20**	0.50	**-0.20**	**-0.40**	**-0.40**
IV	(6) vs.(5)	**-0.30**	**-0.40**	0.10	0.20	0.00	0.20	0.00	0.10	**-0.10**	0.20	**-0.10**	0.20	**-0.10**	**-0.10**	**-0.40**

Analysis of the Model Complexity

Figure 6.3 shows the k values per adaptive algorithm and score for the five selected learning scenarios. We can make the following observations:

1. Adap1 and Adap2 are able to scale up the model's complexity as it is shown by the increasing value of the k value over time. Note that at some time point the k value stops increasing and reaches some resistance level, which can evidence that the adaptation process could be stopped.

2. When adaptive algorithms are used with the `MDL` and the `Bayes` scores, we can observe that they are able to approach the real degree of attribute dependence existing in each domain: k approaches 1 if a 1-DBC generative model is used, k approaches 2 if a 2-DBC model is used, and so on. However, for the `BDeu` score, in most cases, k approaches 5 (the maximum degree of allowable attribute dependence), even in the case when simpler generative models are used. This may indicate that, in fact, `BDeu` favours the choice of models with greater complexity. Further, we will provide some extra analysis to better understand why we obtain these results with the `BDeu` score.

3. Specially for more complex domains, the increasing slope of the k value using Adap2 is more gradual than that using Adap1. As a result, Adap2 can induce less complex classifiers. Adap2 can get trapped in less complex structures while reducing the bias on the parameter estimates.

Table 6.3 summarizes the number of arcs added to the NB structure, that is, the number of added attribute dependencies in all the resulting classifiers. These results give us an idea of the difference in the complexity of the models induced by different algorithms and scores. For each generative model and score we present the number of existing attribute dependencies in the `true structure` (line (1)) and the number of added dependencies using different k-DBCs class-models (varying k from 1 to 5). Then the number of attribute dependencies in the best k-DBC [1] are placed separately in line (2) in order to compare this number against the number of dependencies added to the NB structure by Adap1 and Adap2, respectively.

[1]The "best" k-DBC is the k-DBC with the best performance, according to the results of the final error shown in Table 6.2.

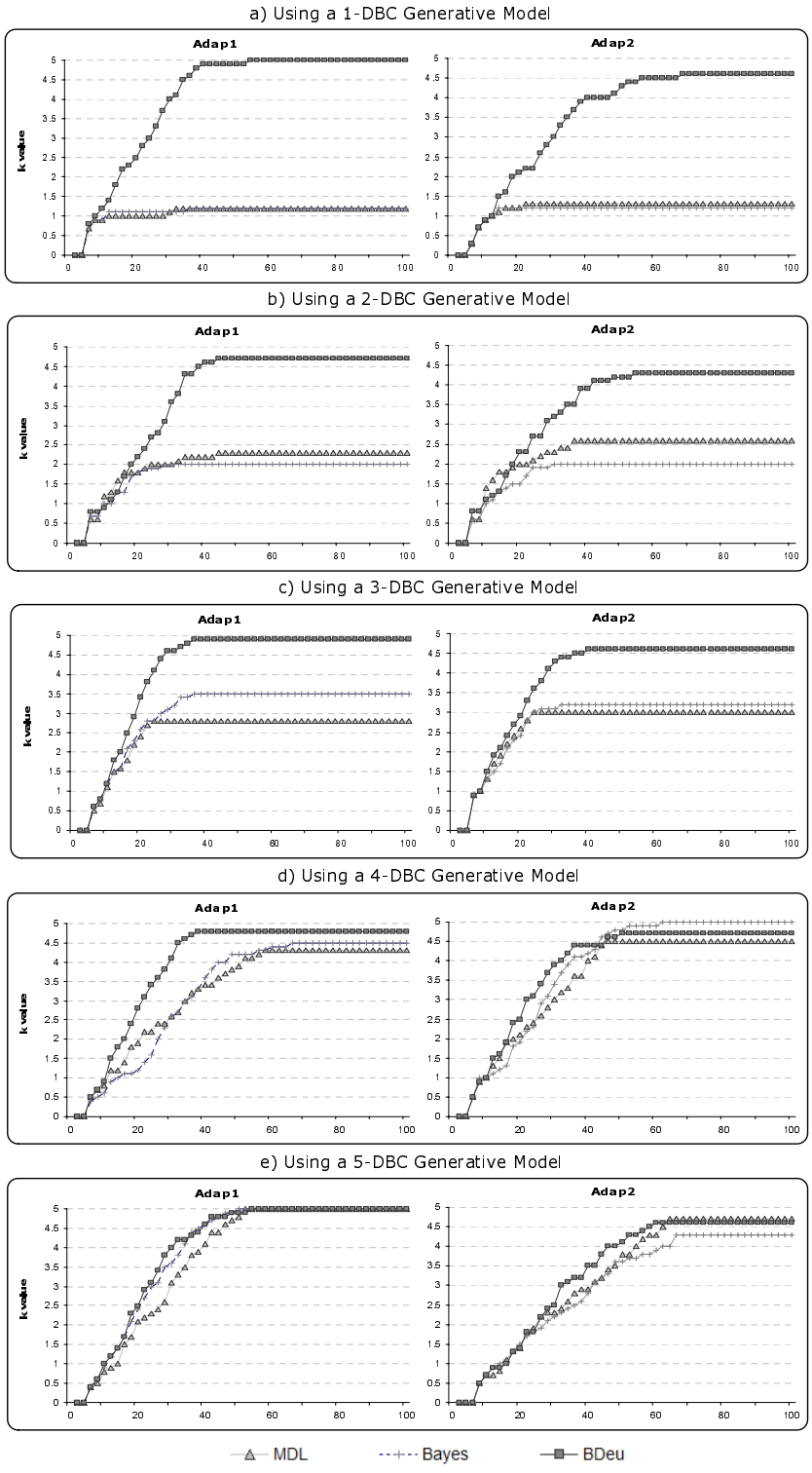

Figure 6.3: k values of Adap1 and Adap2 per generative k-DBC *class-model* and *score*

The last lines of Table 6.3 present some comparative studies:

V - This study summarizes the differences in the number of attribute dependencies found when a *hill-climbing search* algorithm is used to approximate a Bayesian network structure against the real number of existing dependences.

VI - This study compares the number of attribute dependencies found by Adap1 and Adap2 against those found by the best k-DBC. In most cases the differences in the number of added dependences between the best k-DBC and adaptive algorithms are small. The only exception is when the `BDeu` score and simpler generative models (for $k \leq 3$) are used. As stated, we further provide an analysis to explain why the results with the `BDeu` score are not satisfactory in terms of the complexity of induced classifiers.

VII - This study compares the number of attribute dependencies found by Adap2 against Adap1. In most cases, the number of dependencies found by Adap2 is less than those found by Adap1. Results give us extra evidence that there is a reduction of the complexity in the resulting classifiers when the `AdPreqFr4SL` is combined with IB.

Analysis of the Bias-Variance Decomposition for the Error using `BDeu` score

In fact, `BDeu` favours complex structures because the addition of an arc will always increase the likelihood of a model. When the complexity grows significantly, `BDeu` can lead to severe *overfitting* and consequently to a great deterioration of the performance. The overfitting produced by `BDeu` was previously illustrated in the case-study of Chapter 4 with the *nursery* dataset (see, for instance, Figure 6.4). We further show the results of a conducted case study with the artificial domain generated by a `1-DBC` model, which aims to investigate how `BDeu` handles the *bias-variance* trade-off for different k-DBCs. Figure 6.4 shows the *bias-variance* decomposition of the *test error* in the next batch of examples for several k-DBCs varying k from 0 to 5 at three selected time points: $t = 5$ (using 500 training examples), $t = 50$ (using 5000 training examples) and $t = 100$ (using 10000 training examples). To serve as baseline we also show the *bias-variance* decomposition for the *true model*.

For this particular domain the resulting test error of different k-DBCs does not show significant differences with the increasing of the k value. This may indicate that overfitting does not take place. Note that all the variables in the artificial domain are *binary*, so the

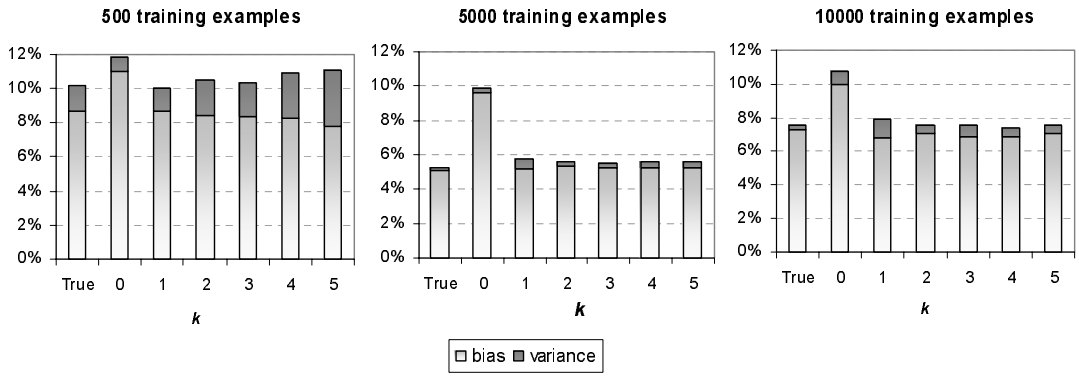

Figure 6.4: Bias-variance decomposition of the test error of several k-DBCs against the true model for the `BDeu` score using a 1-DBC generative model at three selected time points

number of parameters does not grow so much as k increases. Consequently, the induced models present *low variance*. Since the variance represents only a small portion of the *test error* in this domain, the reduction of the bias component is still more crucial to obtain the desirable improvements in the performance with time. However, in spite of the fact that induced models become more and more complex as k values increases (see the number of added arcs for the `BDeu` score and the 1-DBC class-model in Table 6.3), it is noticed from the bias-variance decomposition, similar behaviors in *bias* and *variance*. Thus, in these artificial problems our adaptive algorithms with the `BDeu` score are not able to find less complex models and to approach the true degree of attribute dependence because when k is increased (i.e. the search space is expanded) a model with a higher `BDeu` score can be found, which also presents a good performance. In the next sections, we will prove that this situation does not take place with other domains where there is clear evidence that *overfitting* takes place, specially for smaller amount of training data and more complex k-DBC class-models.

Analysis of the Adaptive Actions and Control Strategies

Table 6.4 shows the number of times different states have been detected and different adaptive actions have been carried out per *class-model*, *score* and *adaptive algorithm*. The list of states and adaptive actions corresponds to the definitions provided in Section 5.6.1. From the results we can observe that in most cases the number of times that the state `S2=STOP IMPROVING` was detected is quite small. These values represent the number of times during the whole

learning process at which the adaptive algorithm triggered a structure-adaptation action. 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 a total of 100 learning steps.

Figure 6.5 shows the decomposition of the number of times a structure adaptation has been carried out (the values represented in the line that corresponds to the action A2 in Table 6.2) into two components: *i*) the number of times that the structure actually changed after structure adaptation; *ii*) the number of times that the structure remains the same after adaptation.

From the graphics on Figure 6.5 we can make the following observations:

1. The proportion of the number of times the structure actually changed when a search procedure was invoked is satisfactory for all the scores, 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 dependencies.

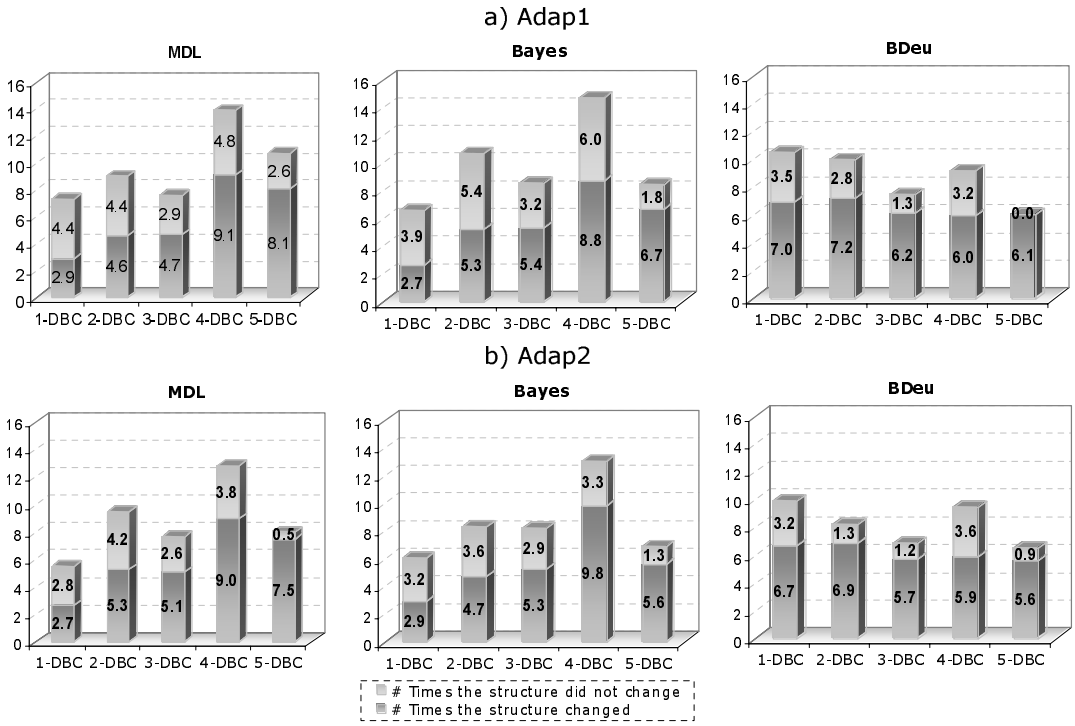

Figure 6.5: The decomposition of the number of times a structure adaptation has been carried out per *adaptive algorithm*, *score* and k-DBC *generative class-model*

Table 6.3: The number of arcs added to the NB's structure per generative k-DBC *class-model*, *score* and *adaptive algorithm*

CLASS-MODEL		**1-DBC**			**2-DBC**			**3-DBC**			**4-DBC**			**5-DBC**		
(1)	True Struct.	7.0			7.0			12.0			22.0			30.0		
	k-DBCs	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu
	1-DBC	7.0	7.0	7.4	5.9	5.9	6.8	7.0	7.0	7.4	8.0	8.0	8.0	8.0	8.0	8.0
	2-DBC	7.4	7.0	11.5	8.5	6.9	10.4	12.0	12.0	13.0	14.7	14.7	14.9	14.6	14.8	15.0
	3-DBC	7.4	7.0	15.1	9.1	6.9	14.2	14.2	14.0	18.8	19.6	20.8	20.6	20.4	21.0	21.0
	4-DBC	7.4	7.0	18.2	9.1	6.9	18.0	14.2	14.0	23.3	23.1	23.7	25.5	24.9	26.0	25.7
	5-DBC	7.4	7.0	21.3	9.1	6.9	21.5	14.2	14.0	27.0	24.7	25.3	29.5	28.0	30.0	29.7
(2)	Best k-DBC	7.4	7.0	11.5	8.5	6.9	10.4	14.2	14.0	18.8	24.7	25.3	29.5	28.0	30.0	29.7
(3)	Adap1	6.9	7.0	29.5	7.3	7.1	27.8	12.8	14.6	29.3	22.0	24.4	29.0	28.6	29.5	29.9
(4)	Adap2	6.9	7.0	26.8	7.6	7.0	25.8	13.4	14.2	28.0	23.4	26.4	28.3	27.8	25.5	27.8
V	(2) vs.(1)	0.40	0.00	4.50	1.50	-0.10	3.40	2.20	2.00	6.80	2.70	3.30	7.50	-2.00	0.00	-0.30
VI	(3) vs (2)	-0.50	0.00	18.00	-1.20	0.20	17.40	-1.40	0.60	10.50	-2.70	-0.90	-0.50	0.60	-0.50	0.20
	(4) vs (2)	-0.50	0.00	15.30	-0.90	0.10	15.40	-0.80	0.20	9.20	-1.30	1.10	-1.20	-0.20	-4.50	-1.90
VII	(4) vs.(3)	0.00	0.00	-2.70	0.30	-0.10	-2.00	0.60	-0.40	-1.30	1.40	2.00	-0.70	-0.80	-4.00	-2.10

Table 6.4: Number of states detected and adaptive actions carried out per generative k-DBC *class-model*, *score* and *adaptive algorithm*

CLASS-MODEL		**1-DBC**			**2-DBC**			**3-DBC**			**4-DBC**			**5-DBC**		
SCORE		MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu	MDL	Bayes	BDeu
STATES	Number of times (in 100 learning steps) the corresponding state has been detected															
S2	Adap1	6.3	5.6	9.5	8.0	9.7	9.1	6.7	7.7	6.7	12.9	14.0	8.2	9.7	7.5	5.1
	Adap2	4.5	5.1	9.0	8.5	7.4	7.2	6.8	7.3	6.1	11.9	12.1	8.5	7.1	5.9	5.5
S3	Adap1	0.7	0.7	0.7	1.5	0.9	1.4	0.4	0.3	0.3	0.4	0.3	0.2	0.3	0.2	0.4
	Adap2	0.6	0.9	0.8	1.5	1.2	1.3	0.2	0.4	0.3	0.6	0.3	0.7	0.5	0.6	0.4
S4	Adap1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Adap2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S5	Adap1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Adap2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ACTIONS	Number of times the corresponding adaptive action has been carried out															
A2	Adap1	7.3	6.6	10.5	9.0	10.7	10.0	7.6	8.6	7.5	13.9	14.8	9.2	10.7	8.5	6.1
	Adap2	5.5	6.1	9.9	9.5	8.3	8.2	7.7	8.2	6.9	12.8	13.1	9.5	8.0	6.9	6.5
A3	Adap1	6.3	5.5	5.4	6.8	6.9	5.2	6.3	7.0	5.0	6.2	5.9	6.1	4.6	4.4	4.3
	Adap2	5.0	5.0	5.7	7.4	5.9	4.5	6.4	6.7	4.8	6.2	5.6	5.9	4.1	4.5	4.1
A4	Adap1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Adap2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2. The use of the MDL score leads to an increase of the number of structure adaptations as k increases. Since MDL prefers simpler structures, adaptive algorithms are forced to trigger more adaptations on the structure when the generative models becomes more complex. On the contrary, since BDeu favors more complex structures, adaptive algorithms reduce the number of structure adaptations as k increases. These results reflect the efforts made by the adaptive algorithms to compensate for the limitations of both scores, that is, for avoiding *underfitting* or *overfitting*.

3. There is a clear rupture in the general tendency of the behavior in all these graphics for the artificial scenario that was generated using a 4-DBC class-model. We can observe that for all the scores and the two adaptive algorithms the total of performed structure adaptations was superior to the total of other class-models. Looking at the results from Table 6.2 we can verify that the best performance in this scenario generated by a 4-DBC is obtained with a 5-DBC classifier instead of a 4-DBC. This evidences that adaptive algorithms continued searching for new dependencies after k has overcome the value 4, while continuing to improve the performance.

4. For all the scores and generative class-models the total of adaptations performed in the structure using Adap2 is inferior to the total obtained with Adap1. On the other hand, we had already shown that in most cases, a more significant reduction of the error can be achieved when the AdPreqFr4SL is combined with the IB procedure for parameter refinement. This may indicate that Adap2 ensures a best balance between the *cost* of updating and the *gain* in performance.

Finally, the results that correspond to the number of times that the state S4= CONCEPT DRIFT and S5=CONCEPT SHIFT has been detected provide evidence that our control strategies for detecting concept drift worked quite well. It was never detected neither a concept drift nor concept shift during the whole learning process. Only at some sporadic situations a concept drift alert was signaled (the results corresponding to the state S3). Consequently, the action A4=BUILD MODEL that builds a new model was never activated.

Analysis of the Stopping Criterion

The graphics in Figure 6.6 show the learning step for which the adaptation process has been stopped (i.e. the *stopping point*) as a function of the k-DBC generative class-model for Adap1 and Adap2, respectively. Thus, each point in a line represents the *stopping point* of the related k-DBC generative model for a particular score. When scores that favour less complex structures, such as, `MDL` and `Bayes` are used, there is a clear tendency to delay the stopping point as generative models become more complex. Since `MDL` and `Bayes` need more training data to find more complex structures, adaptive algorithms should force the search procedure to continue searching for new dependencies. For `BDeu`, instead, the results are not clear. For less complex generative models (e.g. $k = 1, 2$) the stopping point occurred much more late than with the other scores. This behavior can be explained from the analysis previously done using the *bias-variance* decomposition of the test error for the `BDeu` score and a 1-DBC generative model. As we observed in Figure 6.4, for less complex generative k-DBCS, when the k value is increased the search algorithm was able to find a model with a higher `BDeu` score that also presents a good performance. This is why, for less complex generative models, the adaptive algorithms using `BDeu` were not able to stop the adaptation earlier.

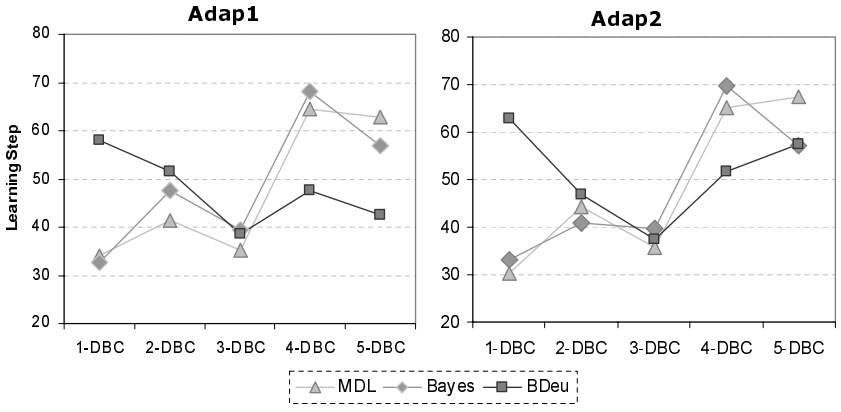

Figure 6.6: The stopping point as a function of the k-DBC generative class-model per *adaptive algorithm*

Summary of Results

In this particular experiment all the results show that adaptive algorithms performed as expected, independently of the score used. They were able to significantly improve the performance of NB over time and approach the performance of the best k-DBC induced from scratch at each learning step with the same underlying learning algorithm and score. On the other hand, we have demonstrated that a considerable reduction of the cost of updating is achieved, as shown by the small number of adaptations performed on the structure during the whole learning process. This fact 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 dependencies. Moreover, we showed that the adaptive algorithms are able to scale up the model's complexity and to perform an artful *bias management* during the whole learning process. The k value increased over time and, in most of cases, approached the real degree of attribute dependencies. Although both, Adap1 and Adap2, show the desirable behavior, results evidence that Adap2 ensures a best *cost-performance* trade-off for more complex domains: the number of structure adaptations and the resulting error are smaller.

6.2.2 Study II - Concept Shift Scenarios

The main purpose of this study was to verify whether our control and adaptive strategies are able to detect abrupt concept changes and to adapt quickly to these changes thus verifying a good *recoverability capability*. In this study we use only the Bayes score.

Generation of Concept Shift Scenarios

We generated five different *concept shift scenarios*, CD-I, CD-II, ..., CD-V, each of them representing a sequence of five different learning contexts with four abrupt concept changes. Each scenario was associated to a k-DBC generative class-model: CD-I to a 1-DBC, CD-II to a 2-DBC and so on. For each scenario we generated 10 different datasets with 10000 examples where the underlying generative distribution model was forced to change after every 2000 training examples. Therefore, we can assume that each generated sample represents a sequence of five different learning contexts with four abrupt concept changes, as shown in Figure 6.7.

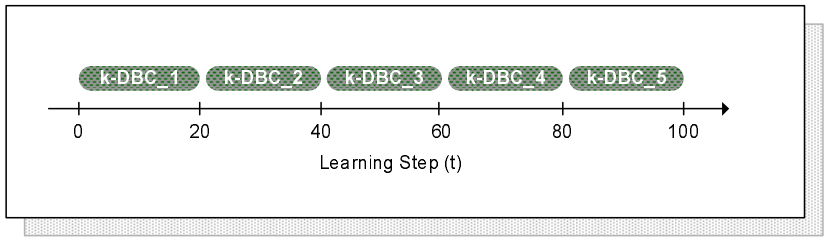

Figure 6.7: Artificially generated concept shift scenarios

We used the 25 randomly generated k-DBCs to generate the five concept shift scenarios. We composed each sample $\mathcal{D}_k^{(i)}, i = 1 \ldots 10$ for the concept shift scenario associated to the k-DBC class-model as follows:

1. We used the first 200 examples from each randomly generated dataset $\mathcal{D}_{k_j}^{(i)}$ for $j = 1 \ldots 5$. We denote the sample of these 200 examples by $B_{k_j}^{(i)}$.

2. We composed $\mathcal{D}_k^{(i)}$ in this way:

$$\mathcal{D}_k^{(i)} = B_{k_1}^{(i)} \bigcup B_{k_2}^{(i)} \bigcup B_{k_3}^{(i)} \bigcup B_{k_4}^{(i)} \bigcup B_{k_5}^{(i)}$$

A Case-Study of the Concept Shift Management

Figure 6.8 illustrates a `P-chart` for concept shift detection using one of the randomly generated samples in the scenario CS-II (associated to a 2-DBC generative class-model). As described in Section 5.5.2, in each time point, the target value is set to Err_{min} - the minimum value of the model error Err_S using the current structure. Then, the chart lines are adjusted according to the target value. In this particular sample, the `P-Chart` was able to detect the four concept shifts that actually are in the data. As a result, a new model was re-built after each shift detection.

Figure 6.9 compares the behavior of the *model error* against the *true model error* over time in this particular concept shift scenario. Vertical light-grey dotted lines indicate the time points at which the current structure was adapted. Vertical black dashed lines indicate the time points at which the current structure was rebuilt using a NB structure and the examples from the `SHORT-MEMORY`. On top, the resulting sequence of different structures and their corresponding k-DBC class-models is presented. The structure was rebuilt four times

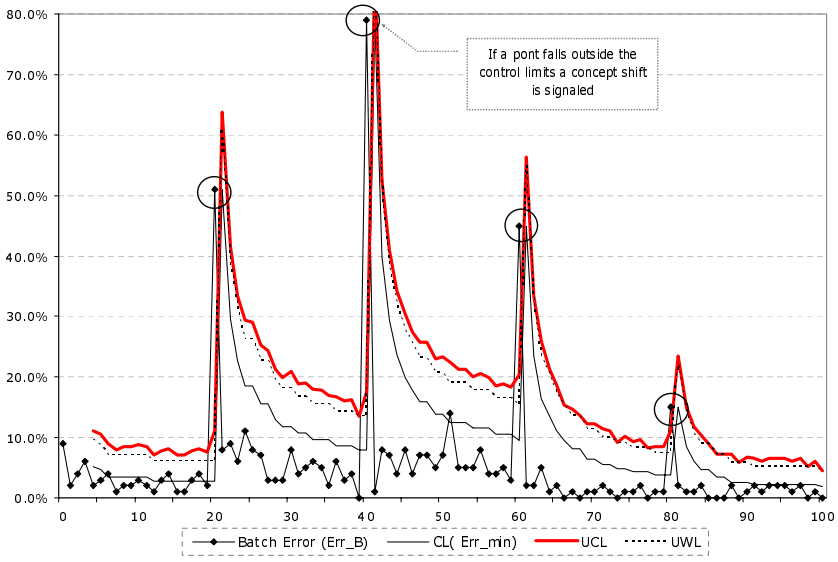

Figure 6.8: A `P-Chart` for handling concept shift in the artificial scenario CS-II

at time points that exactly correspond to the abrupt concept changes. The *model error* decreases with time and, in most cases, approaches the error of the *true model*, except when a concept shift occurs and the performance suffers a significant deterioration. However, the adaptive algorithm shows a good *recoverability capability*. This was able to control the performance, trying to improve it back to a level, that even approaches the performance of

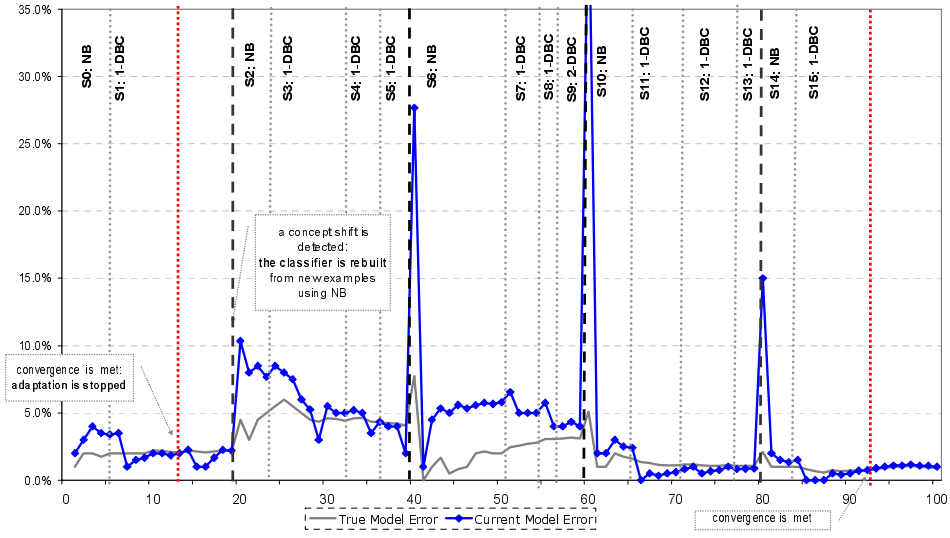

Figure 6.9: Behavior of the model error for the artificial scenario CS-II

the *true model*. Finally, vertical dark-grey dotted lines indicate the time points at which the adaptation process was stopped. At t=14 the adaptation process was temporarily stopped until at t=20 a concept shift was detected and the adaptation procedures were once again activated in order to recover the performance. At t=94, the performance reached its plateau and the adaptation process was definitively stopped.

Global Analysis of the Concept Shift Management

Figure 6.10 allow us to globally analyze the performance of adaptive algorithms for the five generated concept shift scenarios. Plots on the left compare the error rate of the adaptive algorithms Adap1 and Adap2 against the NB and one k-DBC induced by scratch at each learning step. For the scenario SC-I we induced a 1-DBC; for the SC-II, we induced a 2-DBC, and so on. While there is no concept drift, all the classifiers show improvements in the performance as time increases. After a concept shift occurs, the performance of all the algorithms suffers a significant deterioration. However, Adap1 and Adap2 show a good recoverability capability. Results evidence that significant improvements in the performance are achieved by using adaptive algorithms with concept-drift detection instead of their non-adaptive versions.

Plots on the middle depict the *model error* of Adap1 and Adap2. To serve as baseline we also plot the error rate of the true model for each learning context. For all the scenarios, the behavior of the model error reflects the effort made by the adaptive algorithms for recovering the performance after a concept shift occurs. Moreover, in most cases, the model error of Adap1 and Adap2 approaches the error of the true model in each learning context, a phenomenon that is more pronounced in scenarios with simpler generative k-DBCs (e.g. CS-I, CS-II and CS-III). Finally, plots on the right show the k values per adaptive algorithm. We can make two observations. First, the k value gradually increases in stable phases. After a concept shift occurs, in most cases, the k value falls to 0, which evidences that a concept shift has been detected and a new NB has been built. Second, the maximum value of k accomplished by the adaptive algorithms, as a rule, increases from scenario to scenario. This provides evidence that adaptive algorithms attempt to select a class-model that approaches the complexity of the generative distribution.

However, from the plots of the k values we can also observe that adaptive algorithms were

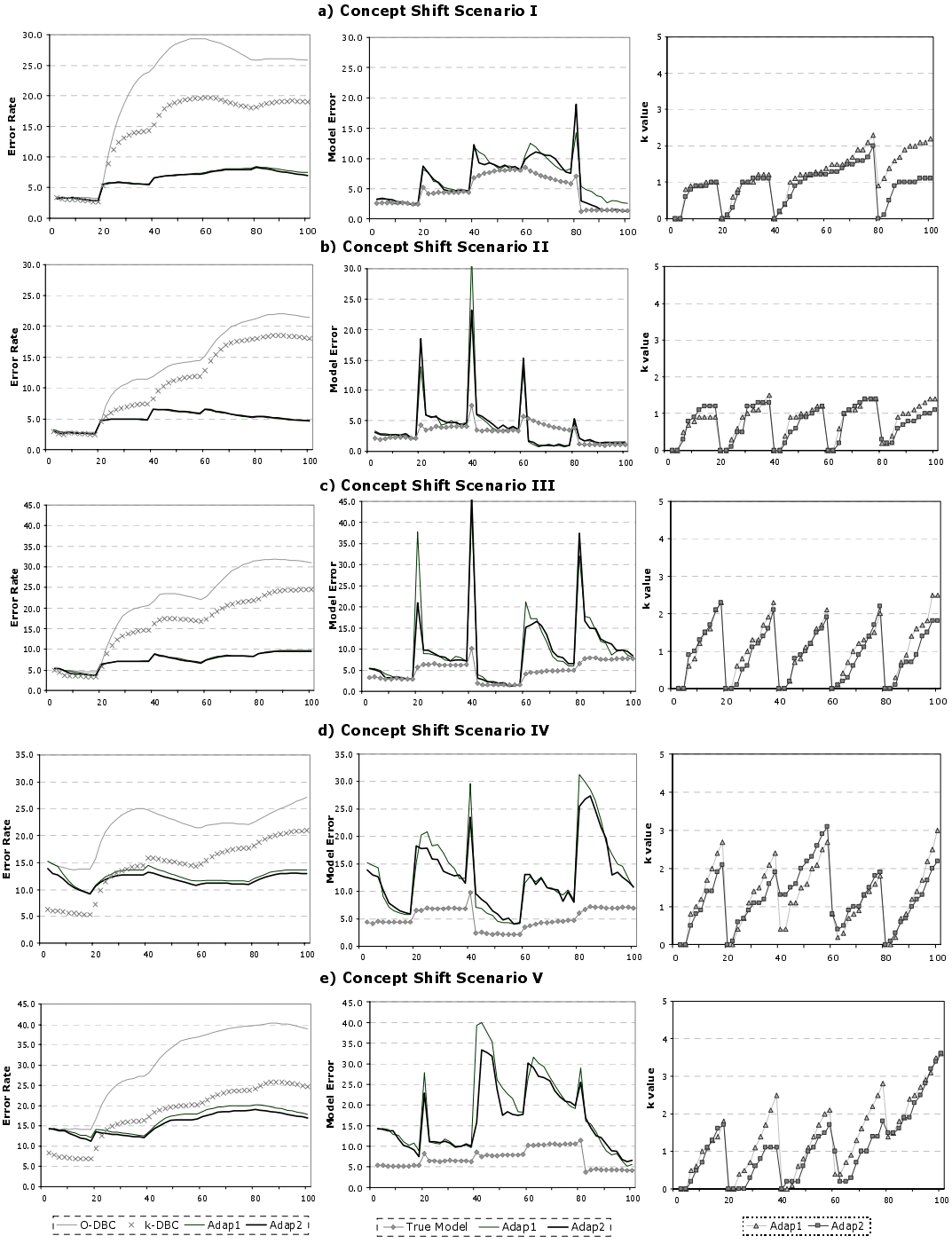

Figure 6.10: Behavior of the error rate, model error and k values of the different algorithms for the five concept shift scenarios

not always capable of detecting concept shift in all the 10 generated samples. We can observe that the averaged k values do not fall to 0, for instance, in the scenario CS-I at t=60 and in

the scenario CS-V at t=80. Nevertheless, looking at the plots of the model errors for these two scenarios at these time points, we see that adaptive algorithms show an acceptable behavior. This may indicate that at these time points the changes in the generative distribution may not have been so abrupt and, hence, would not lead to a significant deterioration of the performance so that the batch error falls outside the control line and the `P-Chart` could detect a concept shift situation.

Analysis of the Adaptive Actions and Control Strategies

Table 6.5 presents the number of times and learning steps at which concept shift was detected for all the generated samples of each concept shift scenario and adaptive algorithm. Only in the scenario CS-III for all the 10 generated samples the four concept shifts were detected. In the scenarios CS-II and CS-IV, in most cases, the four concept shifts were also detected. In the scenarios CS-I and CS-V, instead, we observe a less number of shift detections. However, from the plots of the error rate and the model error in Figure 6.10, we can see that these failures in the detection of concept shift do not seem to have a negative effect in the overall performance.

Table 6.6 shows the number of times different states have been detected and different adaptive actions have been carried out per *concept shift scenario* and *adaptive algorithm*[2]. We can observe that in average the number of times that the state `S5=CONCEPT SHIFT` was detected is satisfactory and, in most cases, this number approaches 4. The results for the state `S3` represent the number of times a first warning was detected. The results for the state `S4` represent the number of times a *concept drift* was signaled, that is, the number of times a certain number of successive warnings was detected. Note that the total of warning situations detected is greater for scenarios with a less number of shift detections (e.g. for CS-I and CS-V). This may indicate that, in some cases, concept drift alerts were signaled instead of concept shift. As argued above, we suspect that at some shift time points, the differences in the generative distributions were not so significant. Otherwise, the performance should have shown a great deterioration. As depicted in the plots of Figure 6.10 this situation did not take place. On the other hand, the values that correspond to the state `S6` represent the number of times the *stopping criterion* was meet during the whole

[2]The list of states and adaptive actions corresponds to the definitions provided in Section 5.6.1.

Table 6.5: Number of times and learning steps at which concept shift was detected per *concept shift scenario* and *adaptive algorithm*

	CS-I		**CS-II**		**CS-III**		**CS-IV**		**CS-V**	
	ADAP1									
Sample	♯S5	Steps	♯S5	Steps	♯S5	Steps	♯S5	Steps	♯S5	Steps
1	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80	3	20,40,60
2	3	20,40,80	3	20,40,60	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80
3	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,61,80	2	20,40
4	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80
5	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,61,80	3	20,40,60
6	2	20,40	4	20,40,60,80	4	20,40,60,80	3	20,65,80	4	20,40,60,80
7	2	20,40	4	20,40,60,80	4	20,40,60,80	3	20,60,80	3	20,40,80
8	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80
9	2	20,40	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80	3	20,40,60
10	3	20,40,80	4	20,40,60,82	4	20,40,60,80	4	20,40,60,80	3	20,40,60
Avg	2.70		3.90		4.00		3.80		3.30	
	ADAP2									
Sample	♯S5	Steps	♯S5	Steps	♯S5	Steps	♯S5	Steps	♯S5	Steps
1	3	20,40,80	4	20,40,60,80	4	20,40,60,80	3	20,60,80	3	20,40,62
2	3	20,40,80	3	20,40,60	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80
3	3	20,40,80	4	20,40,60,80	4	20,40,60,80	2	20,80	3	20,40,62
4	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80	2	20,40
5	3	20,40,80	4	20,40,60,80	4	20,40,60,80	3	20,60,80	4	20,40,60,80
6	3	20,40,80	4	20,40,60,80	4	20,40,60,80	3	20,62,80	3	20,40,62
7	3	20,40,80	4	20,40,60,80	4	20,40,60,80	3	20,60,80	4	20,40,60,80
8	3	20,40,80	4	20,40,60,80	4	20,40,60,80	3	20,60,80	3	20,40,62
9	3	20,40,80	4	20,40,60,80	4	20,40,60,80	4	20,40,60,80	3	20,40,60
10	3	20,40,80	4	20,40,60,82	4	20,40,60,80	4	20,40,60,80	3	20,40,60
Avg	3.00		3.90		4.00		3.30		3.20	

learning process. We can see that the adaptation process was stopped at least once when less complex generative class-models were used (e.g. for the CS-I and CS-II). As illustrated in Figure 6.9, adaptive algorithms can converge to a particular classifier and then stop doing any adaptation. However, the monitoring process will continue working. If any significant change in the behavior is observed, then the adaptation process is re-launched.

From Table 6.6 we can also observe that the number of times that the state `S2=STOP IMPROVING` was signaled fluctuates between 11.1 and 16.9 times over a total of 100 leaning steps. These values represent the number of times that the adaptive algorithm triggered a

Table 6.6: Number of states detected and adaptive actions carried out per *concept shift scenario* and *adaptive algorithm*

SCENARIO		**CS-I**	**CS-II**	**CS-III**	**CS-IV**	**CS-V**
STATES		Number of times the state has been detected				
S2	Adap1	16.0	14.5	16.9	16.2	15.0
	Adap2	14.5	12.5	12.8	13.0	11.1
S3	Adap1	0.9	0.5	0.0	0.4	1.0
	Adap2	0.9	0.4	0.1	0.4	0.9
S4	Adap1	0.0	0.3	0.0	0.3	0.2
	Adap2	0.0	0.1	0.0	0.2	0.3
S5	Adap1	2.7	3.9	4.0	3.8	3.3
	Adap2	3.0	3.9	4.0	3.3	3.2
S6	Adap1	1.4	1.2	0.3	0.1	0.1
	Adap2	1.0	0.6	0.4	0.0	0.1
ACTIONS		Number of times the action has been executed				
A2	Adap1	17.1	15.2	17.0	16.3	15.1
	Adap2	15.2	12.9	13.1	13.0	11.2
A3	Adap1	12.1	9.3	13.5	13.9	12.9
	Adap2	10.7	8.4	11.6	11.8	10.4
A4	Adap1	2.7	3.9	4.0	3.8	3.3
	Adap2	3.0	3.9	4.0	3.3	3.2

structure-adaptation procedure. We can observe that, in general, this number is superior in concept shift scenarios than in stationary scenarios. Note that in concept shift scenarios the classifier is rebuilt using the simplest NB structure whenever a change is detected. As

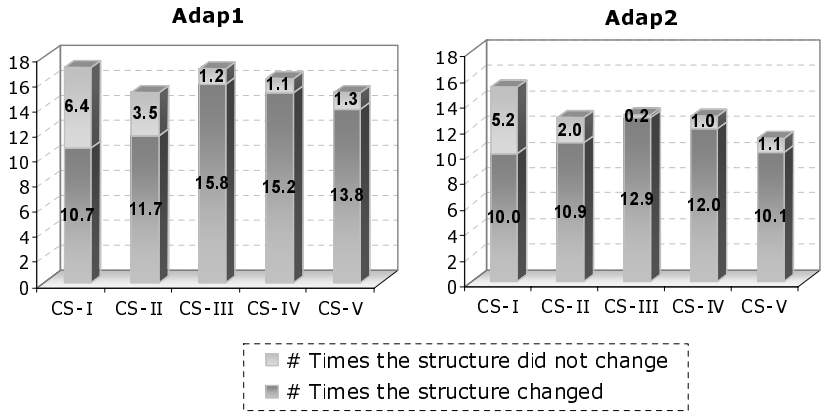

Figure 6.11: The decomposition of the number of times a structure adaptation has been carried out per *adaptive algorithm* and *concept shift scenario*

a result, the adaptive algorithm is forced to trigger new adaptations in the structure until a new satisfactory level of performance is once again reached. Nevertheless, the number of adaptations performed in the structure, even in concept shift scenarios, is acceptable. Figure 6.11 indicates the number of times the structure really changed when a search procedure was invoked. Results suggest that, in general, adaptation in the structure was triggered when the search algorithm was actually capable of finding a different structure.

6.2.3 Study III - Concept Drift Scenarios

The main purpose of this study was to verify whether our control and adaptive strategies are able to detect *gradual* concept changes and to adapt quickly to these changes, thus showing a good *recoverability* capability. Unlike previous experiments where we used batches of 100 examples, in this study we reduced the number of examples in one batch from 100 to 50. In changing environments where gradual changes are more likely to occur, it is more appropriate to assess the performance of learning algorithms more frequently, at shorter time intervals. In this study we also use the Bayes score.

Generation of Concept Drift Scenarios

The simulation of concept drift scenarios is a more difficult task since we need to simulate gradual changes of the target function. Similarly to the previous experiments, we used the 25 artificially generated k-DBC-j in order to produce five different *concept drift scenarios*. We denote them by CD-I, CD-II, ..., CD-V, respectively. Each concept drift scenario is defined as a sequence of five learning contexts, each of them associated to a different k-DBC class-model. Thus, whereas k remains constant in concept shift scenarios, we used k-DBCs of increasing complexity for generating concept drift scenarios (a 1-DBC for the first context, a 2-DBC for the second one, etc.). Thus, for each scenario, we generated 10 different datasets with 10000 examples where each sample represents a sequence of five learning contexts with four gradual changes, as shown in Figure 6.12.

Following the methodology proposed in [143] based on the α function depicted in Figure 5.6, we provided the simulation procedure with the following parameter settings, which were arbitrarily set and not as a result of preliminary experiments:

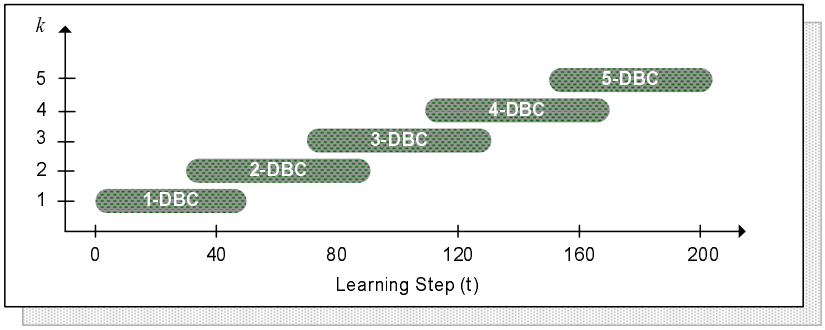

Figure 6.12: Artificially generated concept drift scenarios

- $t_1 = 37, t_2 = 77, t_3 = 117, t_4 = 157$ - the time points where the concept begins to drift.
- $\Delta = 300$ - the number of examples in the *drift phase* (a slow drift).
- $\alpha = 3/4$ - each 3 examples of the old concept appears one example of the new concept.

We generated each sample $\mathcal{D}_j^{(i)}, i = 1 \ldots 10$ associated to the j^{th} concept drift scenario for $j = 1, 2, 3, 4, 5$ as follows:

1. From the generated datasets $\mathcal{D}_{k_j}^{(i)}$ we selected the required number of examples for stables and drift phases according to the current parameter settings. We denote the set of selected examples for stable phases by $B_{k_j}^{(i)}$ for $k = 1, 2, 3, 4, 5$ and the set of examples for drift phases by $T_{k,k+1_j}^{(i)}$ for $k = 1, 2, 3, 4$.
2. We composed the sample $\mathcal{D}_k^{(i)}$ is this way:

$$\mathcal{D}_j^{(i)} = B_{1_j}^{(i)} \bigcup T_{1,2_j}^{(i)} \bigcup B_{2_j}^{(i)} \bigcup T_{2,3_j}^{(i)} \bigcup B_{3_j}^{(i)} \bigcup T_{3,4_j}^{(i)} \bigcup B_{4_j}^{(i)} \bigcup T_{4,5_j}^{(i)} \bigcup B_{5_j}^{(i)} \tag{6.1}$$

A Case-Study of the Concept Drift Management

Figure 6.13 illustrates how the `P-chart` is used to handle gradual changes for one of the generated concept drift scenarios using Adap2. Parallel light-grey dotted lines identify the beginning and the end of each drift phase. Points inside dark circles are those points that fall outside the control limit where a *concept shift* was signaled. Points inside light circles are those points that fall between the warning and control limits where a *concept drift* was signaled. Figure 6.14 depicts the behavior of the *model error* and the *true model error* over

time for this concept drift scenario. Vertical light-grey dotted lines and black dashed lines indicate the time points at which the current structure was adapted or rebuilt, respectively. On top, the sequence of different structures is presented. Vertical dark-grey dotted lines indicate the time points where the adaptation process was stopped.

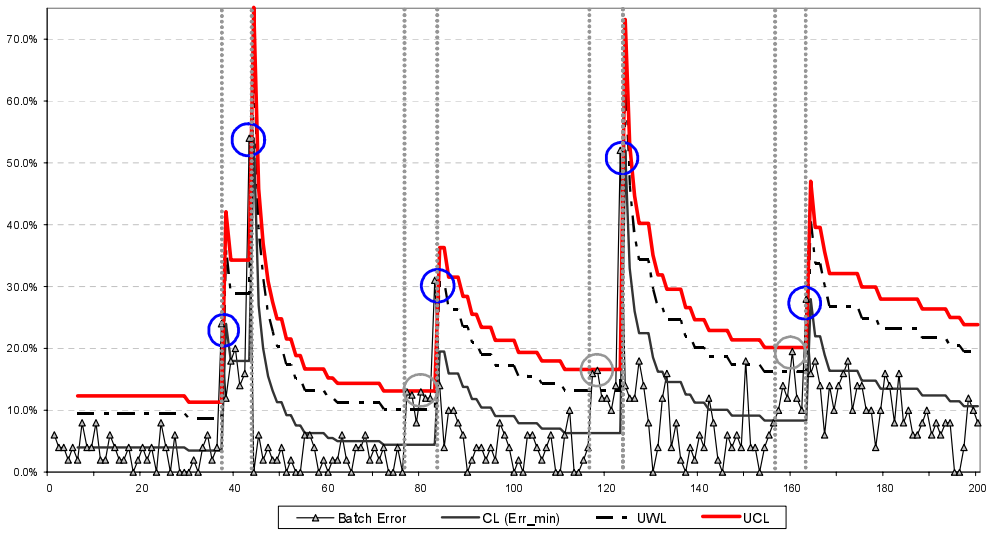

Figure 6.13: The `P-Chart` for one generated concept drift scenario

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 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 shows a good behavior in this context. In this particular sample, the adaptive algorithm shows a good recoverability capability in

order to deal with concept drift scenarios.

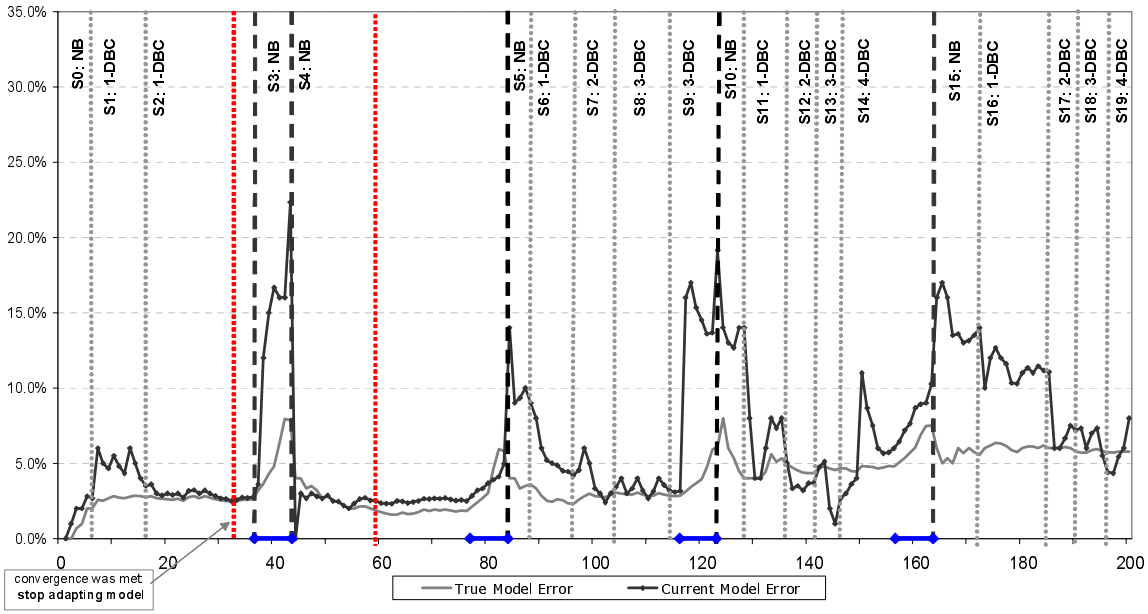

Figure 6.14: Behavior of the model error for a concept drift scenario

Global Analysis of the Concept Drift Management

Figure 6.15 allows us to globally analyze the performance of adaptive algorithms for the five concept drift scenarios. Results evidence that significant improvements in the performance are achieved by using adaptive algorithms with concept-drift detection capability instead of their non-adaptive versions. Moreover, the performance of both adaptive versions is very similar. Only in the CD-I and the CD-III Adap2 can slightly outperfoms Adap1. The model error of adaptive algorithms also shows the behavior expected. In stable phases the model error approaches the error of the *true model*, specially in the first three learning contexts where simpler generative k-DBCs were used. After drift phases the behavior of the *model error* reflects the effort made by our adaptive algorithms for recovering their performance level. We can observe that the k value falls to 0 in most of drift phases, which evidences that a concept shift has been detected and a new NB classifier has been built. The only exception is the scenario CD-III, where the adaptive algorithms were not always capable of detecting a concept shift and, hence, rebuilding the model. Nevertheless, it does not seem to have a negative effect in the overall performance. Finally, we can verify that adaptive algorithms are able to scale up

the model's complexity in stable phases. The k value gradually increases with time. Moreover, as expected, the maximum value of k accomplished by the adaptive algorithms increases from context to context. This evidences that adaptive algorithms attempt to approach the appropriate degree of attribute dependence associated to each learning context.

Analysis of the Adaptive Actions and Control Strategies

Table 6.7 shows the number of times different states have been detected and different adaptive actions have been carried out per *concept drift scenario* and *adaptive algorithm*. We can observe that the number of times that the state `S5=CONCEPT SHIFT` was detected is located in the range between 3.7 (for CD-III) and 5.0 (for CD-I, IV). These values, in average, correspond to the number of times a new classifier was rebuilt during all the learning process (i.e., when the action `A4` was carried out). Comparing these results with the results from Table 6.6 we can observe that the number of concept drift alerts (state `S3`) and concept drift alarms (state `S4`) is greater than these numbers for concept shift scenarios. These results reflect that adaptive algorithms were able to detect concept drift situations. Moreover, the number of times a concept drift alert `S3` was detected is greater for scenarios with a less number of shift detections. In fact, in the scenario CD-III was detected a great number of warning situations. As a result, the number of times the classifier was rebuilt was less than in the other scenarios. This evidences that gradual changes in the context may not always lead to such a great deterioration of the performance so that the values of the error sample fall outside the control line of the `P-Chart`. Finally, the values that correspond to the state `S6` 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.

From Table 6.7 we can also observe that, in most cases, the number of times that the state `S2=STOP IMPROVING` was signaled fluctuates between 19.9 and 33.3 times over a total of 100 leaning steps. When we compare these results with the results from Table 6.6 using concept shift scenarios, we can see an increase in the number of adaptations performed in the structure. This increase happens for several reasons: in concept drift scenarios the learning process suffers of instability phases during more time. Hence, the probability that the error sample falls outside the control line is higher. Moreover, in this experiment we increase the model's complexity from context to context. Therefore, adaptive algorithms need to search

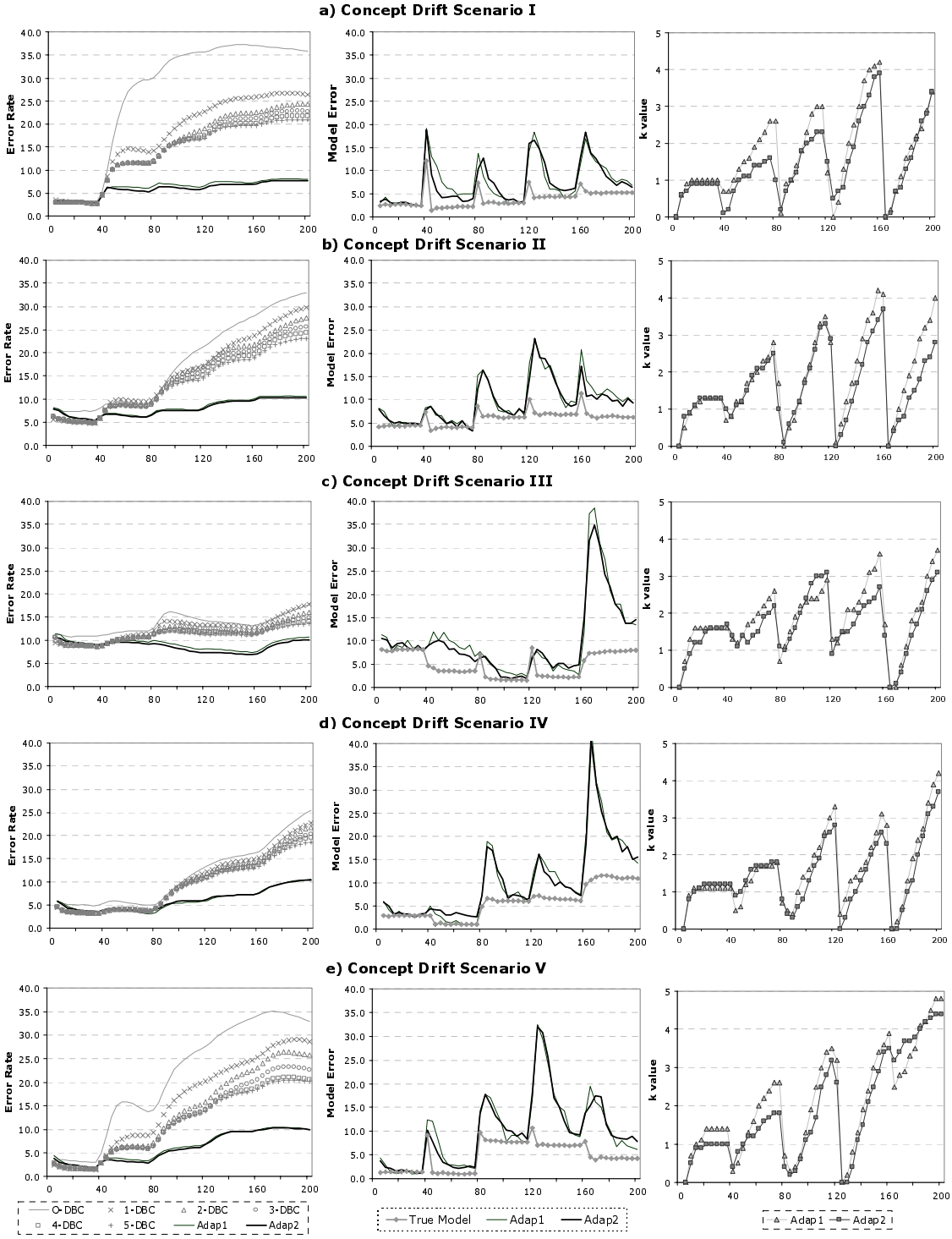

Figure 6.15: Behavior of the error rate, model error and k values of the different algorithms for the five concept drift scenarios

Table 6.7: Number of states detected and adaptive actions carried out per *concept drift scenario* and *adaptive algorithm*

SCENARIO		**CD-I**	**CD-II**	**CD-III**	**CD-IV**	**CD-V**
STATES		Number of times the state has been detected				
S2	Adap1	32.8	33.3	31.0	28.5	27.2
	Adap2	22.3	27.4	24.6	23.8	19.9
S3	Adap1	4.2	5.9	8.0	5.6	4.3
	Adap2	4.0	5.3	7.1	5.3	3.0
S4	Adap1	1.7	2.7	3.0	2.2	1.4
	Adap2	1.4	2.2	2.9	1.5	1.0
S5	Adap1	5.0	4.7	3.7	5.0	4.3
	Adap2	5.0	4.2	3.8	4.5	4.0
S6	Adap1	1.0	0.6	0.7	1.3	0.8
	Adap2	1.7	0.4	0.8	1.1	1.8
ACTIONS		Number of times the action has been executed				
A2	Adap1	33.2	33.8	31.5	28.9	27.6
	Adap2	23.3	27.8	25.2	24.2	20.8
A3	Adap1	21.3	22.8	18.8	17.4	18.9
	Adap2	16.8	19.3	15.3	16.5	13.6
A4	Adap1	5.0	4.7	3.7	5.0	4.3
	Adap2	5.0	4.2	3.8	4.5	4.0

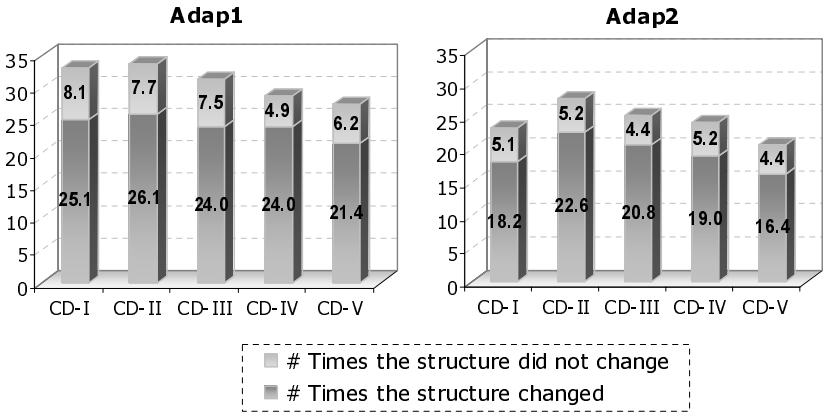

Figure 6.16: The decomposition of the number of times a structure adaptation has been carried out per *adaptive algorithm* and *concept drift scenario*

for more complex structures as time increases in all the concept drift scenarios. Figure 6.16 illustrates the number of times the structure really changed when a search procedure was invoked. These graphics indicate that in concept drift scenarios, in general, adaptations in

the structure also was triggered when the search algorithm was actually capable of finding a different structure.

Finally, note that similarly to the previous experiments, the number of times an adaptation structure action has been carried out during the whole learning process using Adap2 is inferior to the number obtained using Adap1. Moreover, we have shown that in this study both algorithms show a similar performance and that in some contexts Adap2 slightly outperforms Adap1. Therefore, for concept drift scenarios, Adap2 can also ensure a best balance between the *cost of updating* and the *gain in performance*.

6.3 Evaluation with UCI Datasets

In this section we will show through experiments on a set of classification problems from the UCI repository [102] that adaptive algorithms are able to perform an artful *cost-performance* trade-off independently of the score used. We started with experiments involving the same three datasets (*balance*, *nursery* and *adult*) and experimental settings used in the empirical study with k-DBCs of Chapter 4 so that we can give continuity to this study. The main purpose was to compare the performance of adaptive algorithms against their *non-adaptive* versions for five selected scores. Since we use different scores for the same learning algorithm, these experiments allowed us to verify that adaptive algorithms perform as expected independently of the score used.

6.3.1 Evaluating the Behavior for Different Scores

The main purpose of these experiments was to test the hypothesis that adaptive algorithms are able to approach the performance of the best k-DBC induced with the same underlying learning algorithm and score using a temporal batch learning approach while considerable reducing the cost of updating. To this end we compared the performance of adaptive and non-adaptive algorithms using five scores: `MDL`, AIC, Bayes (BD), Preq and `BDeu`. All the indicators' values were obtained as the average over 10 generated samples

The results presented below were obtained using the following parameter settings in the `AdPreqFr4SL`. We set `maxTimes` to 3 which means that if for three consecutive times the batch error does not improve after parameter adaptation a structure adaptation action is launched.

The thresholds eps1 and eps2 were slightly smoothed in relation to the values chosen with artificial problems. We set eps1=0.01 and eps2=0.001 for the *balance* and *nursery* domains and eps1=0.001 and eps2=0.0001 for the *adult* domain. Since *adult* is a more difficult domain to learn, our intuition was to set smaller thresholds for this particular domain. In Section 6.3.2 we present the results of a conducted study involving other three classification problems from the UCI repository [102] where the main goal was to evaluate the effect of different parameter settings on the behavior of the adaptive algorithms.

Analysis of the Performance versus Complexity over Time

Figures 6.17, 6.18 and 6.19 allow us to analyze the evolution of the *performance* and *complexity* of the induced classifiers using adaptive algorithms against their non-adaptive versions for the five selected scores and the three datasets. As before, plots on the left depict the error rate of the classifiers obtained with Adap1, Adap2 against the NB and several k-DBCs induced from scratch at each learning step. Plots on 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 averaged k values per adaptive algorithm, which give us an idea about the increasing complexity of resulting classifiers over time.

In most cases adaptive algorithms showed the expected behavior for all the datasets and scores: the performance of adaptive algorithms approach the performance of the best k-DBC induced with the same score. There is only one exception, for the *adult* dataset and BDeu score, when Adap1 cannot outperform NB. Analyzing the results we found that for one of the generated samples of the *adult* dataset at some time point a concept shift was detected and a new model was built[3]. As a result, the averaged performance was strongly penalized. Moreover, Adap2 outperforms Adap1 for more complex domains. For the *adult* dataset it is specially evident the advantages of using the Iterative Bayes procedure for parameter adaptation. This domain is very hard to learn, presents a great number of parameters, and hence, more bias and variance in the parameter estimates. Thus, the reduction of the error rate observed with Adap2 is also due to a reduction of the bias component in the parameter estimates. On the other hand, for the *balance* domain and for all the scores except MDL, the

[3]This situation is evident from the results corresponding to the state S5 and the adaptive action A4 for the *adult* dataset and the BDeu score in Table 6.10.

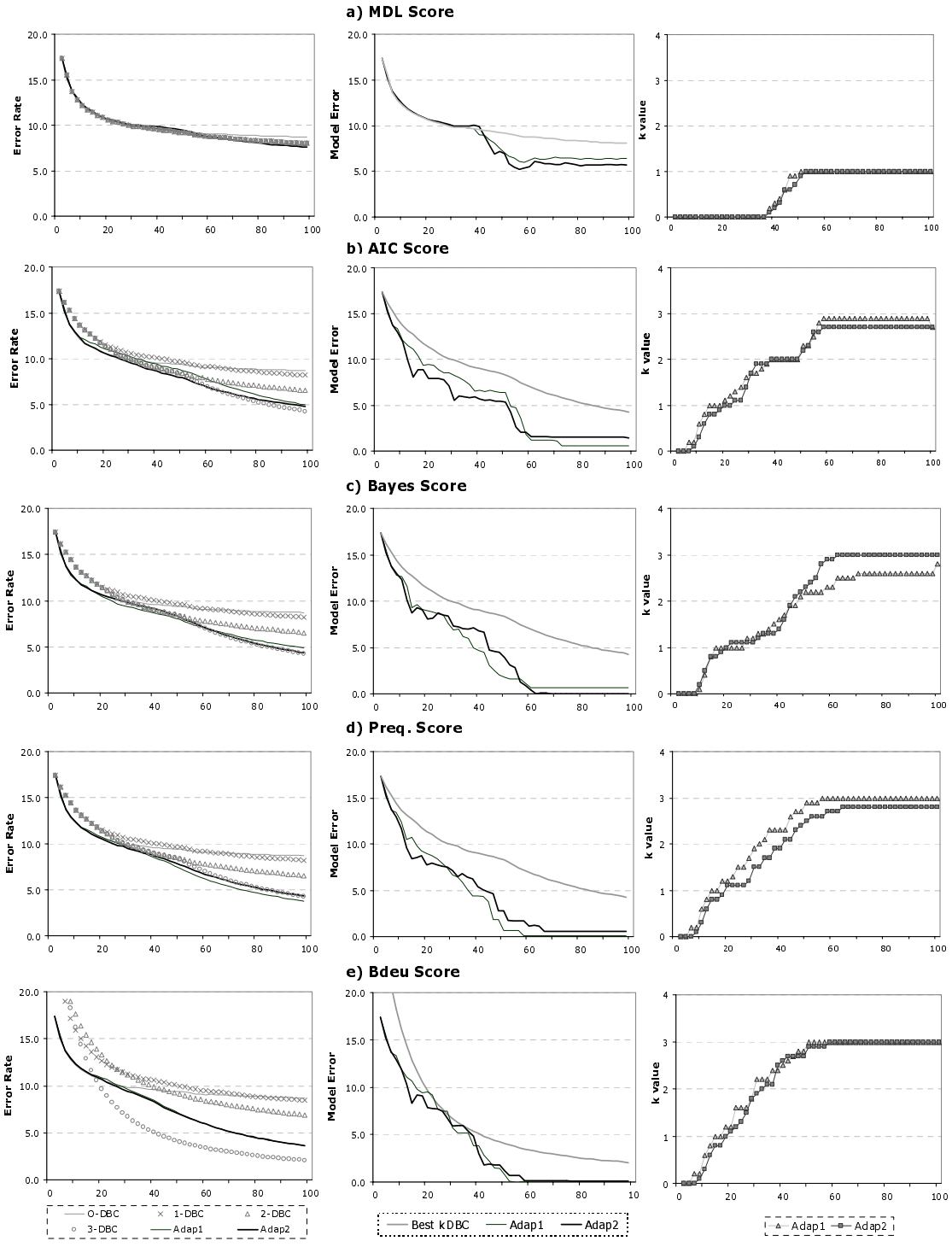

Figure 6.17: Behavior of the error rate, model error and k values of the different algorithms per score for the *balance* dataset

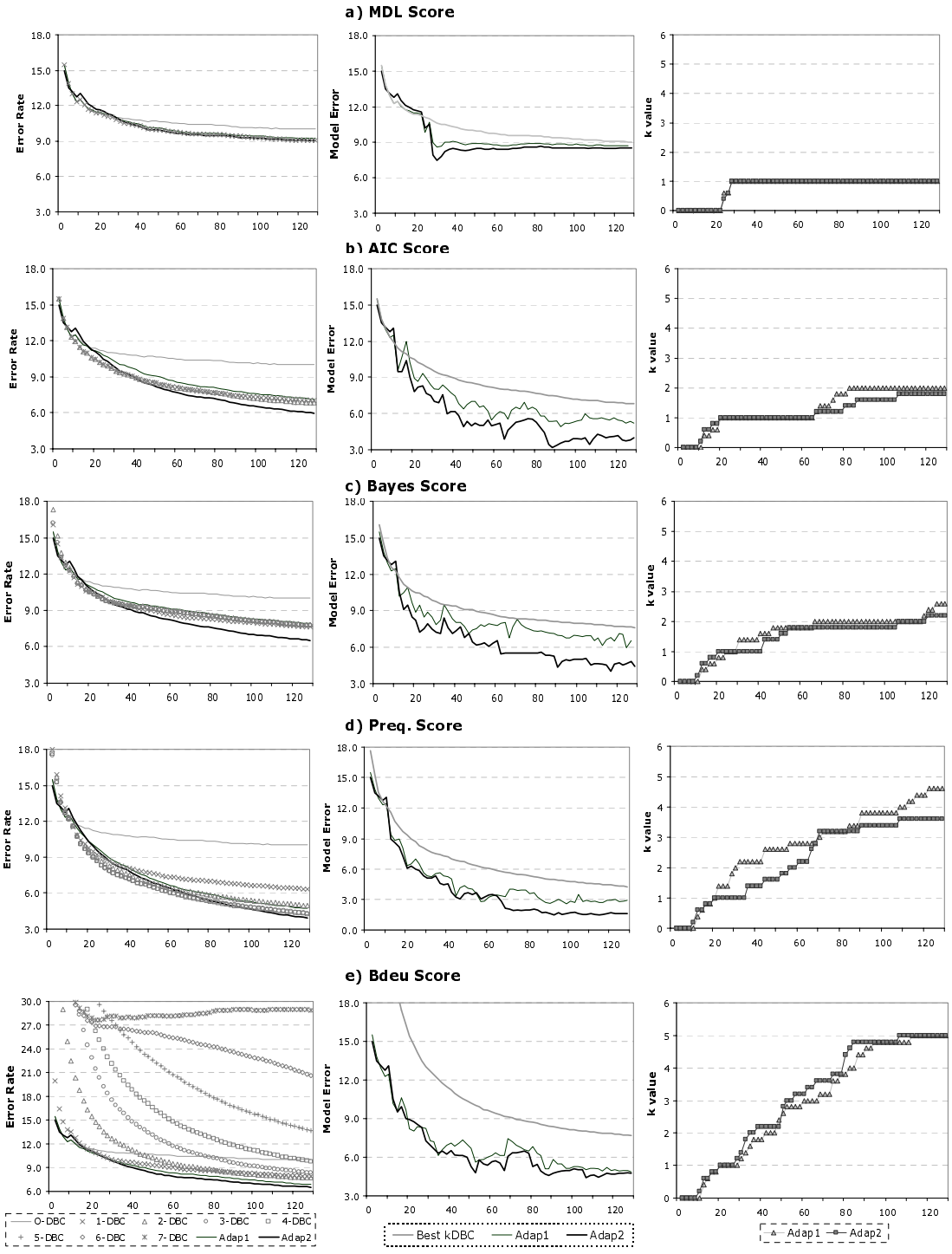

Figure 6.18: Behavior of the error rate, model error and k values of the different algorithms per score for the *nursery* dataset

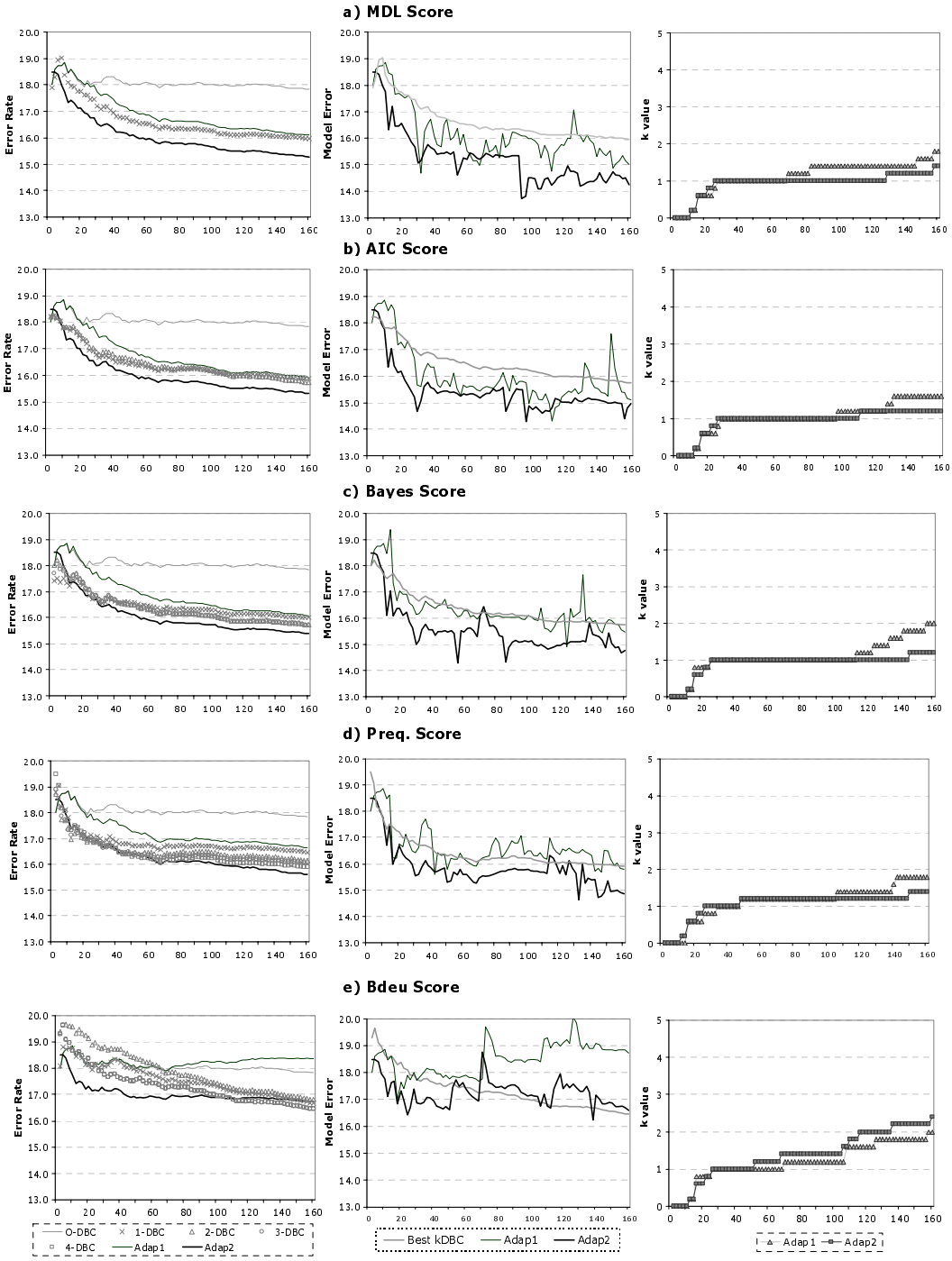

Figure 6.19: Behavior of the error rate, model error and k values of the different algorithms per score for the *adult* dataset

model error 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 dependencies.

Finally, when we compare the behavior of the k values for the *nursery* dataset in Figure 6.18 with the graphics of the *bias-variance* decomposition of the test error depicted in Figures 4.1 and 4.2, we can observe that in most cases, k approaches the k value of the optimal class-model that we have found at the selected time points. Specially the results obtained with `BDeu` in the *nursery* dataset evidence that by gradually increasing the k value over time, we can avoid severe *overfitting* and obtain the desirable performance.

Analysis of the Final Performance versus Complexity

Although we are more interested in showing the advantages of our adaptive algorithms over time, that is during all the learning process, Table 6.9 helps us to compare the final performance and complexity of all the classifiers induced with different algorithms. For each dataset and score we show the *batch error* of the last incoming batch of examples, which was not used to update the classifier. To serve as baseline, we first show the batch error obtained with the NB and with different k-DBCs. As before, the best results that give lower errors among the induced k-DBCs are reported with bold text and placed separately in line (2) in order to compare the results obtained with `Adap1` and `Adap2` against the best k-DBC. To give evidence about the differences in the complexity of the induced classifiers, this table also summarizes the number of arcs added to the NB structure for each compared classifier. The number of added arcs for the best k-DBC are reported with bold text and rewritten in line (5). Similarly to the previous study with artificial datasets, the last lines of Table 6.9 show some comparative studies for a pair of approaches:

I - This study compares the reduction of the error obtained with the best k-DBC instead of using the NB. This comparison allow us to know what is the maximum possible range of reduction of the error of the NB which we can obtain by inducing a k-DBC with an appropriate k value, a hill-climbing search procedure and different scores. Note that for the *balance* dataset we can obtain zero error using a 3-DBC in combination with all the scores, except with the `MDL`. For the *nursery* dataset the greatest reduction (10.60%) was obtained with the `Preq` score using a 3-DBC. For *adult* the reductions obtained were more modest, because this domain is more difficult to learn. However, the greater reductions (2.60%) were obtained with the `AIC` and `BDeu` scores using a 1-DBC. Results

evidence that the combination of a particular score with a particular class-model leads to different results in the performance of k-DBCs for different domains.

II - This study compares the results of our adaptive algorithms against the best k-DBC. A number in bold indicates that the differences in the errors are less or equal to zero. In most cases, the differences in the final error between an adaptive algorithm (Adap1 or Adap2) and the best k-DBC are quite small. In general, Adap1 and Adap2 not only approach the best k-DBC but, in some cases, they can outperform it. Table 6.1 summarizes how many times the final error of the classifier induced with Adap1 and Adap2 was *less*, *equal* or *greater* than that of the best k-DBC.

Table 6.8: Number of times the final error of the adaptive algorithms was *less*, *equal* or *greater* than that of the best k-DBC for the UCI's datasets

	Adap1		Adap2	
	Event	♯ Times	Event	♯ Times
less	(3)<(2)	6/15	(4)<(2)	9/15
equal	(3)=(2)	2/15	(4)=(2)	1/15
greater	(3)-(2) $\leq$ 0.5	4/15	(4)-(2) $\leq$ 0.5	3/15
	(3)-(2) $\geq$ 1	3/15	(4)-(2) $\geq$ 1	2/15

III - This study compares the reduction of the error obtained with Adap2 instead of Adap1. A number in bold indicates the cases when Adap2 outperforms Adap1. We can observe that in 10 of a total of 15 cases, a greater reduction of the error is achieved when adaptive methods are combined with the Iterative Bayes procedure.

IV - This study compares the number of attribute dependencies found by Adap1 and Adap2 against those found by the best k-DBC. These differences give us an idea about how different in complexity the induced classifiers are. We observe that these differences become greater for the *adult* dataset when scores that favor the choice of more complex structures, such as `Preq` and `BDeu` do, are used.

Table 6.9: Final performance and complexity per *dataset* and *score*

DATASET		**Balance**					**Nursery**					**Adult**				
		Error on the last incoming batch of examples														
(1)	NB	8.10					12.00					16.80				
	k-DBCs	MDL	AIC	Bayes	Preq	BDeu	MDL	AIC	Bayes	Preq	BDeu	MDL	AIC	Bayes	Preq	BDeu
	1-DBC	**5.60**	5.60	5.80	6.10	5.90	**8.40**	**6.00**	**5.8**	4.2	5.80	**14.80**	**14.20**	**14.60**	**15.00**	**14.20**
	2-DBC	6.40	3.50	4.60	3.70	4.00	8.40	7.00	7.4	3.8	6.40	15.00	14.80	14.60	16.20	16.00
	3-DBC	6.40	**0.00**	**0.00**	**0.00**	**0.00**	8.40	7.00	7.2	**1.4**	5.40	15.00	14.80	14.60	15.20	15.60
	4-DBC	–	–	–	–	–	8.40	7.00	7.2	2.2	**3.60**	15.00	14.80	14.60	15.40	15.60
	5-DBC	–	–	–	–	–	8.40	7.00	7.2	2.2	4.20	–	–	–	–	–
(2)	Best k-DBC	5.60	0.00	0.00	0.00	0.00	8.40	6.00	5.80	1.40	3.60	14.80	14.20	14.60	15.00	14.20
(3)	Adap1	5.90	0.20	0.30	0.00	0.00	8.80	5.00	6.80	3.00	3.20	14.20	14.00	13.60	14.80	16.80
(4)	Adap2	5.10	1.40	0.00	0.02	0.10	8.00	4.40	4.40	1.20	3.80	13.00	14.00	13.20	14.40	16.00
		Number of added arcs to the NB structure														
	k-DBCs	MDL	AIC	Bayes	Preq	BDeu	MDL	AIC	Bayes	Preq	BDeu	MDL	AIC	Bayes	Preq	BDeu
	1-DBC	**3.0**	3.0	3.0	3.0	3.0	**2.0**	**7.0**	**5.4**	7.0	7.0	**11.0**	**13.0**	**12.0**	**12.4**	**13.0**
	2-DBC	3.2	5.0	5.0	5.0	5.0	2.0	10.0	8.0	13.0	13.0	11.8	20.4	18.0	21.8	25.0
	3-DBC	3.2	**6.0**	**6.0**	**6.0**	**6.0**	2.0	10.0	9.0	**17.8**	18.0	11.8	20.4	18.0	29.2	36.0
	4-DBC	–	–	–	–	–	2.0	10.0	9.0	19.6	**22.0**	11.8	20.4	18.0	33.2	36.0
	5-DBC	–	–	–	–	–	2.0	10.0	9.0	20.8	25.0	–	–	–	–	–
(5)	Best k-DBC	3.0	6.0	6.0	6.0	6.0	2.0	7.0	5.4	17.8	22.0	11.0	13.0	12.0	12.4	13.0
(6)	Adap1	2.9	5.9	5.6	6.0	6.0	1.0	10.2	8.0	19.4	25.0	11.6	16.2	16.8	20.0	16.8
(7)	Adap2	3.0	5.7	6.0	5.7	6.0	1.0	9.2	7.0	17.6	25.0	10.8	13.6	12.2	17.2	16.0
I	(2) vs.(1)	-2.50	**-8.10**	**-8.10**	**-8.10**	**-8.10**	-3.60	-6.00	-6.20	**-10.60**	-8.40	-2.00	**-2.60**	-2.20	-1.80	**-2.60**
II	(3) vs (2)	0.30	0.20	0.30	**0.00**	**0.00**	0.40	**-1.00**	1.00	1.60	**-0.40**	**-0.60**	**-0.20**	**-1.00**	**-0.20**	2.60
	(4) vs (2)	**-0.50**	1.40	**0.00**	0.02	0.10	**-0.40**	**-1.60**	**-1.40**	**-0.20**	0.20	**-1.80**	**-0.20**	**-1.40**	**-0.60**	1.80
III	(4) vs.(3)	**-0.80**	1.20	**-0.30**	0.02	0.10	**-0.80**	**-0.60**	**-2.40**	**-1.80**	0.60	**-1.20**	0.00	**-0.40**	**-0.40**	**-0.80**
IV	(6) vs (5)	-0.10	-0.10	-0.40	0.00	0.00	-1.00	3.20	2.60	1.60	3.00	0.60	3.20	4.80	7.60	3.80
	(7) vs (5)	0.00	-0.30	0.00	-0.30	0.00	-1.00	2.20	1.60	-0.20	3.00	-0.20	0.60	0.20	4.80	3.00
V	(7) vs.(6)	0.10	**-0.20**	0.40	**-0.30**	0.00	**-1.00**	3.20	1.60	**-0.20**	3.00	**-0.80**	**-2.60**	**-4.60**	**-2.80**	**-0.80**

Table 6.10: Number of states detected and adaptive actions carried out per *dataset*, *score* and *adaptive algorithm*

DATASET		**Balance**					**Nursery**					**Adult**				
		MDL	AIC	Bayes	Preq	BDeu	MDL	AIC	Bayes	Preq	BDeu	MDL	AIC	Bayes	Preq	BDeu
STATES		Number of times the state has been detected														
S2	Adap1	13.3	4.5	4.5	3.6	3.4	7.2	14.2	18.6	8.2	6.2	6.4	5.4	4.0	3.6	2.0
	Adap2	9.5	3.6	4.2	3.1	3.4	9.0	11.6	11.2	6.0	5.0	3.4	2.0	2.4	2.6	2.4
S3	Adap1	0.1	0.0	0.2	0.1	0.0	1.2	1.0	0.6	0.4	0.4	1.6	1.2	0.4	0.8	5.2
	Adap2	0.3	0.2	0.0	0.0	0.1	0.8	0.2	0.2	0.2	0.2	1.2	1.4	0.2	1.6	3.2
S4	Adap1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6
	Adap2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S5	Adap1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Adap2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ACTIONS		Number of times the action has been executed														
A2	Adap1	14.1	4.6	4.7	3.6	3.4	8.2	14.4	18.8	8.4	6.4	6.4	5.4	4.0	3.8	2.2
	Adap2	10.4	3.9	4.2	3.1	3.4	10.0	11.8	11.6	6.2	5.4	3.4	2.2	2.6	2.6	2.4
A3	Adap1	12.3	4.2	3.2	3.2	3.2	8.2	7.8	14.2	5.6	5.0	2.6	2.8	2.4	2.0	2.2
	Adap2	9.6	3.8	3.1	2.8	3.0	10.0	6.0	7.8	3.8	5.0	1.4	1.2	1.2	1.4	2.4
A4	Adap1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Adap2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

V - This study compares the number of attribute dependencies found by Adap2 against Adap1. In most cases, the number of dependencies found by Adap2 is less than those found by Adap1. Results give us more evidence that there is a reduction of the complexity in the resulting classifiers when the `AdPreqFr4SL` is combined with the *Iterative Bayes* procedure.

All the presented results give us additional evidence to the tested hypothesis that our adaptive algorithms, in most cases, are able to approach the performance of the the best k-DBC induced with the same score.

Analysis of the Adaptive Actions and Control Strategies

Table 6.10 shows the number of times different states have been detected and different adaptive actions have been carried out per *dataset*, *score* and *adaptive algorithm*[4]. Figures 6.20, 6.21 and 6.22 show the graphics of the decomposition of the number of adaptations performed in the structure for the *balance*, *nursery* and *adult* datasets, respectively. Only for the *balance* and *nursery* datasets these figures also show a graphic depicting the *stopping point* for each adaptive algorithm and score. For the *adult* dataset this graphic is not presented because the adaptation process was never stopped for any score. From the presented table and graphics we can make the following observations:

1. The number of times `S2=STOP IMPROVING` was detected is located between 2.0 (for *adult* using `Adap1` with BDeu) and 18.6 (for *nursery* using `Adap1` with Bayes), which represent, in average, the number of structure adaptations performed during the whole learning process. In most cases, this number was always minimal when using `BDeu`. On the contrary, the number of adaptations using `MDL` or `AIC`, was greater. Results reflect the efforts made by the adaptive algorithms for avoiding *overfitting* and *underfitting*.

2. The number of times `S4= CONCEPT DRIFT` and `S5=CONCEPT SHIFT` were detected provide evidence that our control strategies for detecting concept drift worked also well in these problems. Only in one sample for the *adult* dataset was erroneously detected one concept shift.

[4]The list of states and adaptive actions corresponds to the definitions provided in Section 5.6.1.

3. The number of times that the structure was adapted using Adap2 was always inferior that using Adap1 while we had already shown that we obtain similar or lower errors when adaptive methods are combined with Iterative Bayes. Results evidence that Adap2 ensures a best *cost-performance* trade-off also for these three domains.

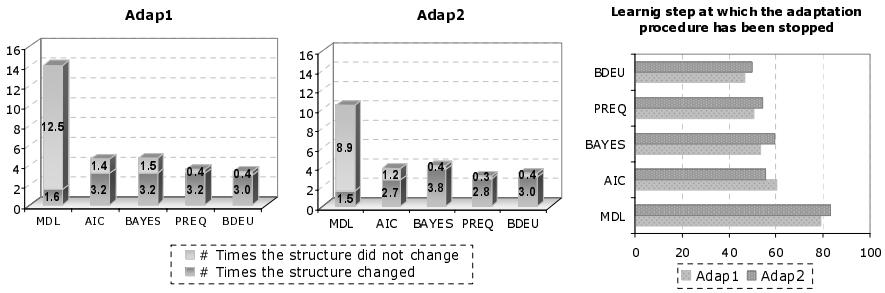

Figure 6.20: Number of structure adaptations and the stopping point for the *balance* dataset

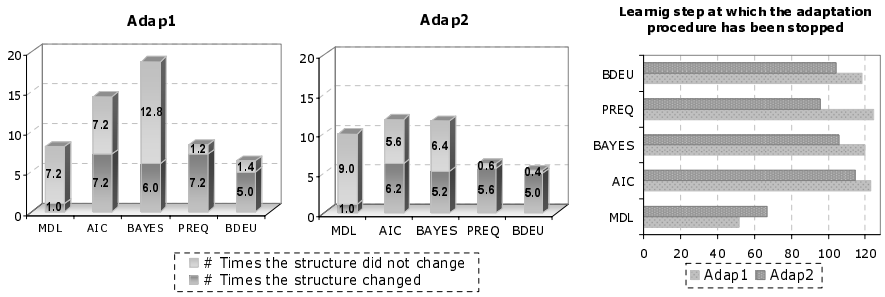

Figure 6.21: Number of structure adaptations and the stopping point for the *nursery* dataset

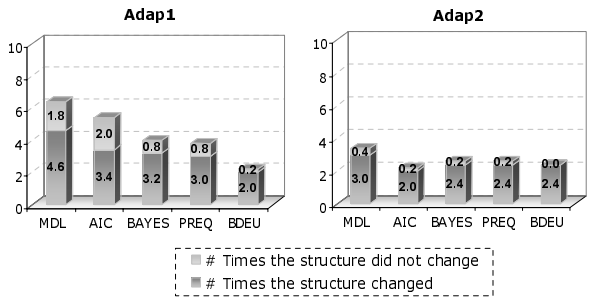

Figure 6.22: Number of structure adaptations for the *adult* dataset

6.3.2 Evaluating the Behavior for Different Parameter Settings

In the following experiments we want to primarily analyze the effect of different threshold settings in the performance of adaptive algorithms.

Experimental Setup

For these experiments we chose three datasets from the UCI repository [102] with large number of examples. The main characteristics of the three datasets are summarized in Table 6.11. This table also shows the accuracy of batch learning algorithms reported in previous work for different classes of BNCs. Superscripts denote references from which these results were taken.

Table 6.11: Datasets used in the experiments with different parameter settings

	#	#	#	Learning	Reported Results with batch learning			
Dataset	Attrib.	Classes	Inst.	Steps	NB	TAN	BAN	GBN
mushrooms	22	2	6300	62	95.79[22]	99.82[22]	100.00[22]	99.30[22]
page-blocks	10	5	5400	53	94.70[76]	n/a	n/a	n/a
letters	17	24	20000	199	74.96[44]	85.86[44]	n/a	75.02[44]

The *mushrooms* problem consists in classifying 23 species of gilled mushrooms in the Agaricus and Lepiota Family. The *page-block* problem consists in classifying all the blocks of the page layout of a document that has been detected by a segmentation process in order to separate text from graphic areas. Finally, the *letters* problem consists in classifying each of a large number of *black-and-white* rectangular pixel displays as one of the 26 capital letters in the English alphabet. All these three problems present numerical attributes and missing values. We used their discretized versions available on-line at [76]. All the indicators' values that we will show below were obtained as the averaged values over 5 generated samples. We performed all the experiments using only the BD score and batches of 100 examples.

Table 6.12: Results with different parameter settings for the *mushrooms*, *page-blocks* and *letters* datasets

		Performance						**Complexity**				**Adaptation**			
		Error Rate		**Model Error**		**Final Batch Error**		**♯ Added Arcs**		**Final k value**		**♯ Struct. Changes**		**When stopped**	
Mushrooms, Learning Steps: 62															
NB		**3.68**		-		**2.20**		-		-		-		-	
Best k-DBC		**0.07**		-		**0.00**		37, 51		2, 3		27.8, 37.6		-	
eps2	eps3	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2
0.05	0.005	1.21	0.67	0.25	0.20	0.40	0.40	27.0	23.0	1.4	1.2	1.6/1.6	1.2/1.2	20.4	16.6
0.01	0.001	1.13	0.62	0.12	0.07	0.20	**0.00**	30.6	27.0	1.6	1.4	2/2	1.8/2	25.4	23.2
0.01	0.0001	1.11	0.59	0.01	0.01	**0.00**	**0.00**	33.0	30.2	1.8	1.6	2.6/2.6	2.2/2.2	29.4	26.4
0.01	0.00001	1.11	0.59	0.01	0.01	**0.00**	**0.00**	33.0	30.2	1.8	1.6	2.6/2.6	2.2/2.2	29.4	26.4
0.001	0.0001	1.12	0.74	0.00	0.24	0.00	0.20	33.0	25.8	1.8	1.4	2.6/2.6	1.8/1.8	32.8	28.2
Page-blocks, Learning Steps: 53															
NB		**5.51**		-		**4.60**		-		-		-		-	
Best k-DBC		**4.60**		-		**3.80**		15.0		2		25.4		-	
eps2	eps3	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2
0.05	0.005	4.83	4.76	3.66	3.45	**2.60**	3.20	21.0	14.6	3.6	2.2	3.4/4.4	2.8/4.2	53	45.8
0.01	0.001	4.92	4.75	3.62	3.73	3.00	**2.40**	19.6	15.2	3.2	2.2	4/4.2	2.8/4.2	53	51.8
0.01	0.0001	4.92	4.75	3.62	3.61	3.00	2.60	19.6	15.4	3.2	2.2	4/4.2	3/3	53	53
0.01	0.00001	4.92	4.75	3.62	3.61	3.00	2.60	19.6	15.4	3.2	2.2	4/4.2	3/3	53	53
0.001	0.0001	5.08	4.77	4.15	3.42	3.20	3.20	14.6	15.2	2.4	2.2	2.8/3.4	2.6/2.8	53	53
Letters, Learning Steps: 199															
NB		**28.77**		-		**26.80**		-		-		-		-	
Best k-DBC		**18.54**		-		**11.40**		16.0		2		24.2		-	
eps2	eps3	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2	Adap1	Adap2
0.05	0.005	19.93	19.70	15.39	15.35	15.00	16.60	15.2	15.0	1.0	1.0	3.6/5.6	3.8/6.4	97.2	93.4
0.01	0.001	19.77	19.38	15.12	15.35	15.80	16.00	15.0	15.0	1.0	1.0	3.4/6.4	3.6/6.8	110.4	112.4
0.01	0.0001	19.10	18.83	12.69	12.79	11.40	12.80	16.2	16.0	1.8	1.6	5/10.2	4.8/10.8	186.4	166.6
0.01	0.00001	19.01	18.63	12.80	11.83	**10.80**	**11.00**	16.6	16.8	2.0	2.0	5.4/15	6/10	199	199
0.001	0.0001	19.41	19.21	12.35	13.22	**10.60**	14.80	16.2	15.6	2.0	1.4	4.2/8.6	3.2/5.2	195.6	149.6

Results with Different Parameter Settings

Table 6.12 summarizes the final results related to the *performance*, *complexity* and *adaptive actions* of the adaptive algorithms `Adap1` and `Adap2` with different threshold settings for the three selected datasets[5]. For each dataset we first show the results obtained with the NB. Then we show the results obtained with the best k-DBC, that is, with the classifier that showed that best performance among the k-DBCs induced from scratch at each learning step. The number (or numbers) that appears in the line "`best k-DBC`" and the column "`Final k value`" corresponds to the k value for that best k-DBC. Finally we present the results obtained by using `Adap1` and `Adap2` with different settings of `eps1` (the threshold for the *gentle* slope) and `eps2` (the threshold for the *plateau*). The number that appears in the column "`When stopped`" corresponds to the learning step for which the adaptation process was stopped. We next summarize the main observations for each dataset.

- **mushrooms dataset:** In this domain, both a 2-DBC and a 3-DBC showed the best performance among the k-DBCs induced from scratch, providing zero final batch errors. The adaptive algorithms behaved as expected for almost all threshold settings. Figure 6.23 depicts the *error rate*, the *model error* and the k-values of the adaptive algorithms with `eps1`=0.01 and `eps2`=0.0001. We have not found significant differences in the behavior of the graphical curves using other threshold settings. However, from the results of the final batch error we can observe that slightly worse results were obtained with `eps1`=0.05 and `eps2`=0.005. In this case adaptive algorithms found a little less dependencies that those that were found using other thresholds because the adaptation process was stopped earlier. This domain has a lot of attributes (a total of 22), and hence, a considerable amount of parameters. It is intuitively more appropriate to use smaller threshold values than those used with the artificial problems where all the variables are binary. We can observe that the best results were obtained with `eps1`=0.01 and `eps2`$\leq$ 0.0001.

- **page-blocks dataset:** In this domain a 2-DBC is the best k-DBC. The adaptive algorithms also showed a good performance for all the threshold settings. The final batch errors of `Adap1` and `Adap2` for all the settings are even lower than those obtained

[5]The remaining parameters remained constant for the following values: `kMax`=5 and `maxTimes`=3.

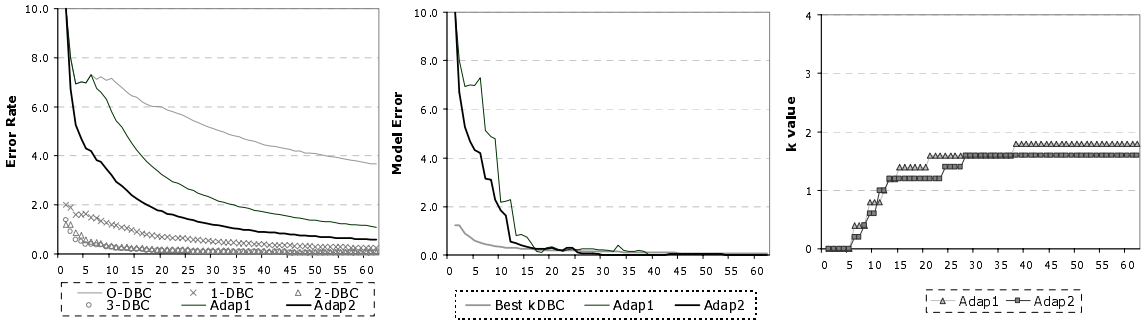

Figure 6.23: Behavior of the error rate, model error and k values of the different algorithms for the *mushrooms* dataset

with the best k-DBC (3.80%). This may indicate that in this domain the choice of different threshold settings have no significant influence on the performance of adaptive algorithms. However, the best results were obtained with `eps1`=0.01 and `eps2`$\leq$ 0.001. Figure 6.24 depicts the graphical behavior of the error rate, the model error and k-values for these threshold values. In general, we have not observed significant differences in the behavior of the graphical curves using other threshold settings.

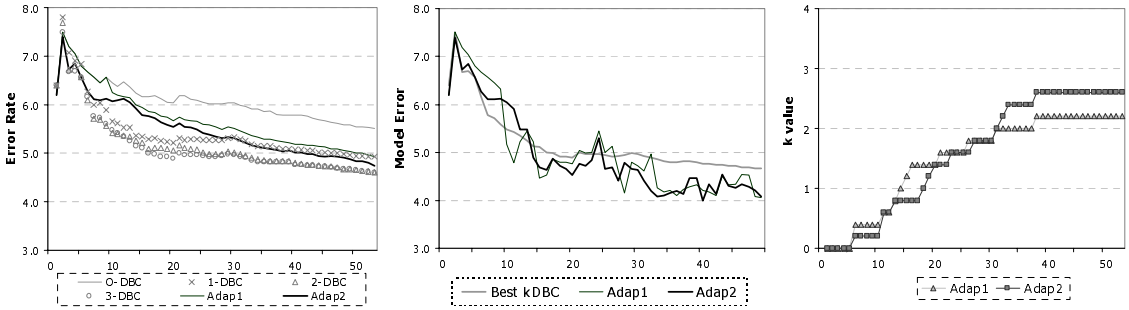

Figure 6.24: Behavior of the error rate, model error and k values of the different algorithms for the *page-blocks* dataset

- **letters dataset**: This domain is more difficult to learn because 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 classifier) to k=2 (the best k-DBC). From the results of Table 6.12 we can see that the error rate falls from 28.77% for the NB to 18.54% for the best k-DBC and the final batch error from 26.80% to 11.40%. These results evidence the advantages of

using BNCs in this complex domain when there is a considerable amount of training data available. On the other hand, the adaptive algorithms behaved as expected for almost all the threshold settings. They were able to scale up the complexity of induced k-DBCs while improving their performance over time. However, from the results it is evident that in this complex domain lower thresholds for the stopping criterion produce better results. The worst results were obtained using greater thresholds `eps1`=0.05 and `eps2`=0.005 whereas the best ones were obtained with lower thresholds `eps1`=0.01 and `eps2`=0.00001.

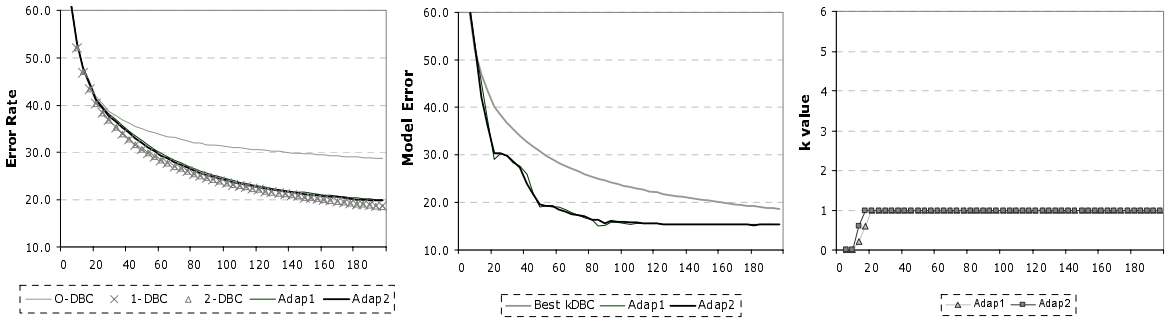

Figure 6.25: Behavior of the error rate, model error and k values of the different algorithms for the *letters* dataset by setting `eps1` = 0.05 and `eps2` = 0.005

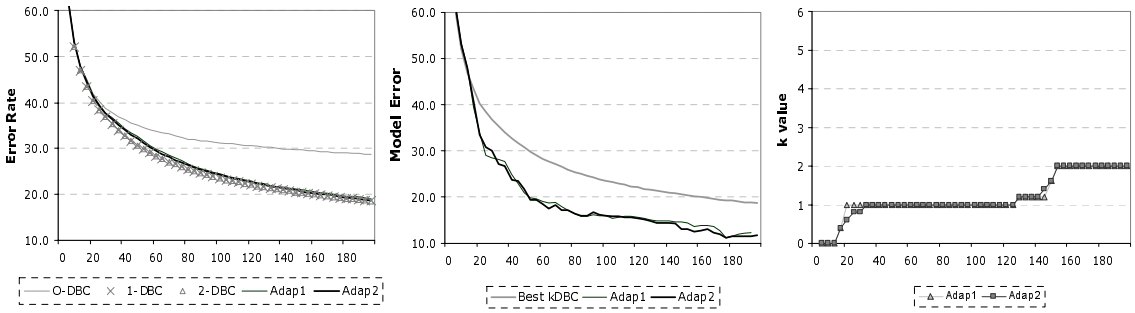

Figure 6.26: Behavior of the error rate, model error and k values of the different algorithms for the *letters* dataset by setting `eps1`=0.01 and `eps2` = 0.00001

Figure 6.25 and 6.26 depict the graphical behavior of the *error rate*, the *model error* and k-values for the *worst* and the *best* threshold settings, respectively. We can observe that when `eps2` is set to 0.005 the adaptation process is stopped very early (at $t = 97.2$ for `Adap1` and at $t = 93.4$ for `Adap2` in a total of 199 learning steps). In average the

number of added dependencies by `Adap1` and `Adap2` was 15.2 and 15.0, respectively. Hence, all the attribute dependencies found by the best k-DBC (a total of 16) could not be captured. Moreover this domain has a great number of parameter estimates. Results indicate that in more complex domains with a great number of parameters we must use lower values for `eps2` to give more time for the adaptation process to continue looking for new dependencies and improving the parameter estimates.

From all the obtained results we can state that the choice of a particular threshold setting can influence the performance of our adaptive algorithms, specially for more complex domains. In less complex domains, as observed, there are only tiny differences in the performance for different parameter settings. As stated, the smaller we choose the value of `eps1`, the slower the model's complexity increases over time. On the other hand, the smaller we choose the value of `eps2`, the later the adaptation process is stopped. The ideal situation is to choose the thresholds so as to produce the best balance between *performance* and *complexity*. The conducted studies with different datasets reveal that we can obtain a better balance if we choose the threshold values in correspondence with the complexity of each particular domain. Thus, we suggest the use of lower thresholds for more complex domains.

Finally, we would like to make some last comments from the results depicted in Table 6.12. The column "♯ `Struct.Changes`" shows the number of times the structure actually changed during all the learning process in relation to the number of performed structure adaptations. The number shown in the line "`best` k`-DBC`" corresponds to the number of times the structure actually changed when the best k-DBC was induced from scratch at each learning step. For example, for the *page-blocks* dataset, in average, 25.4 different structures were found at 53 attempts to update the structure. By using, for instance, the adaptive algorithm `Adap1` with `eps1`=0.01 and `eps2`=0.0001, we can observe that the structure changed 4 times in a total of 4.2 adaptations performed. In general, all the results provide evidence that adaptive algorithms attempt to perform an artful *cost-performance* trade-off. Note that for all these datasets the number of adaptations performed in the structure is small. Moreover, for the *mushrooms* and *page-blocks* domains, in most cases, the number of times the structure actually changed approaches the number of times an adaptation process was launched. This indicates that the structure was adapted only when it was really necessary. For the *letter* datasets, instead, we can observe that the percent of the number of times the structure actually changed

over the number of adaptations performed is lower for lower values of `eps2`. For instance, using `Adap1` with `eps1`=0.01 and `eps2`=0.0001 we obtain 5/10.2 = 49% whereas for `eps2`=0.00001 we obtain 5.4/15 = 36%. By setting a lower threshold for the plateau, the adaptive algorithm is forced to trigger adaptation in the structure more frequently because the slope of the model error becomes more and more gentle as time increases while the required threshold is still not reached. However, although the cost of updating increased we observe a reduction of the final error from 11.40% using `eps2`=0.0001 until 10.80% using `eps2`=0.00001. This reduction of the error takes place because by using a lower threshold value for `eps2`, the adaptive algorithm is able to continue searching for new attribute dependencies and improving the parameter estimates. Note that the number of dependencies added to the NB structure increases from 16.2 using `eps2`=0.0001 until 16.6 using `eps2`=0.0001.

In general, comparing the number of adaptations performed in the structure we can observe a reduction using `Adap2` instead of `Adap1`. On the other hand, for the *mushrooms* and *page-blocks* domains `Adap2` outperforms `Adap1` for almost all threshold settings. For the *letters* domain `Adap2` shows a better performance over time but it does not outperform `Adap1` in the final batch error. For these three datasets we can also state that `Adap2`, in general, performs a better *cost-performance* trade-off than `Adap1`.

6.4 Concluding Remarks

We have evaluated the adaptive algorithms for BNCs, using both, artificial and benchmark problems. The use of artificial domains allowed us to test the two main issues that the algorithm exhibits: *bias management* and *concept drift management* knowing the true degree of attribute dependencies and when concept drift actually occurs. Results show that the adaptive algorithms are able to scale up the model's complexity and to perform an artful *bias management* during the whole learning process. Moreover, in most cases, the resulting k values approach the real degree of attribute dependence. On the other hand, the method for *handling concept drift* based on the `P-Chart` demonstrated to be efficient. Results show that the `P-Chart` is able to consistently recognize concept changes, both *abrupt* and *gradual*, and to adapt quickly to these changes, thus verifying a good *recoverability capability*.

Results in conducted experiments with UCI's benchmark problems show that adaptive

algorithms work as expected, independently of the score used. By gradually increasing the k value adaptive algorithms are capable of improving the predictive accuracy of the Naïve Bayes significantly over time. This evidence that modeling attribute dependencies can improve the classification results. Results also show that, in most cases, the resulting classifier approaches the best k-DBC induced with the same underlying learning algorithm and score but using a temporal batch learning approach. On the other hand, a considerable reduction of the cost of updating is achieved by using adaptive algorithms, as shown by the small number of adaptations performed on the structure during the whole learning process in contrast to the great cost of rebuilding the structure from scratch at each learning step. Results evidence that it is more appropriate to perform adaptations on the structure only when there is some accumulated data and the search procedure is actually able to find new dependencies.

All the results in general give us some evidence that the control criteria for detecting when to start searching for a new structure and when to stop adapting work quite consistent. However, these criteria depend on two thresholds: the threshold for the gentle slope `eps1` and the threshold for the plateau `eps2`. From the results of a conducted study with different threshold settings we observe that the choice of a particular setting can influence the performance of our adaptive algorithms, specially for more complex domains. In less complex domains were observed only tiny differences in the performance for different parameter settings. As suggested, it is more appropriate the use of lower thresholds for more complex domains.

Finally, although both Adap1 and Adap2 show a desirable behavior, results evidence that Adap2, in general, ensures a better *cost-performance* trade-off: 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. Adap2 can get trapped in less complex structures while reducing the bias on the parameter estimates. By using the `AdPreqFr4SL` with the Iterative Bayes, 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.

Conclusions and Future Research

The main contribution of this thesis has been the development of adaptive algorithms for Bayesian network classifiers in a prequential learning scenario, which attempt to handle the trade-off between the *cost of adaptation* and the *gain in performance* and cope with *concept drift*. We approach the *cost-performance* trade-off through *bias management* and *adaptation control*. The method for handling concept drift is explicitly modeled using a Shewhart `P-Chart`.

Our contributions to the area of machine learning are both general and specific. They are general because we integrated all the adaptive algorithms into the `AdPreqFr4SL` - an *unified, adaptive prequential framework for supervised learning*. The `AdPreqFr4SL` is provided with simple and effective controlling mechanisms that try to select the best adaptive actions according to the current learning goals. To this end, two *performance indicators* - the *batch error* and the *model error*, are monitored over time. Since indicators are classifier-independent, our controlling methods are broadly applicable to a range of supervised learning algorithms. Moreover, we believe that almost all of the adaptive policies for bias management could be applied to essentially any supervised learning algorithm based on parametric models and discrete search with a *hierarchical* and *increasing* control over the complexity of its induced hypotheses.

Our contributions are also specific because we applied the adaptive algorithms for the particular class of Bayesian network classifiers. Unlike previous work in the field of learning Bayesian networks, which have mainly focused in batch learning algorithms (with only a few exceptions), we have developed and evaluated the adaptive algorithms in a *prequential learning framework*. We further summarize the main contributions of this thesis to the particular field of Bayesian network classifiers (BNCs):

- Whereas previous studies focused on the comparison of *unsupervised* versus *supervised* scores, we have investigated how the combination of three factors: *i*) the *score*; *ii*) the *class-model*; and *iii*) the *size of the training data*; can affect the performance of BNCs in a *prequential learning framework*. Experimental results suggest that the selection of an appropriate score depends not only on the learning goal, that is, whether the score is specialized for the classification task or not. This selection should be also based on how each score makes the *bias-variance* trade-off for selecting a structure with the appropriate complexity according to the chosen *class-model* and available data. Moreover, we experimentally give more evidence to the fact that not only the *choice of the score* but also the *choice of a class-model* with the appropriate complexity for the available training data is crucial to obtain a good performance of BNCs.

- The selection of an appropriate *class-model* is still more challenging when we learn BNCs in a *prequential learning framework* since the amount of training data varies over time. We propose a simple and practical adaptive strategy based upon *bias management* and *adaptation control* aimed at automatically solving this problem. Instead of selecting a particular class-model of BNCs (e.g. TAN, BAN, etc.) and using it during all the learning process, we use the class of k-DBCs and start with its simplest class-model: the Naïve Bayes classifier. We then attempt to reduce the bias of the Naïve Bayes by gradually adding dependencies between the attributes over time. To this end, we gradually increase the maximum number of allowable attribute dependencies (the k value) so that at each learning step we can use a k-DBC class-model that better suits the available data. On the other hand we reduce the cost of updating by using new data to primarily adapt the parameters. We use simple heuristics aimed at controlling the current performance to decide whether it makes sense to adapt the structure. Moreover we stop doing any adaptation when there is evidence that the use of more training data will not result in significantly improved performance. All these adaptive and control strategies attempt to perform an artful *cost-performance* trade-off.

- We experimentally showed that adaptive algorithms are able to improve the predictive accuracy of the Naïve Bayes significantly over time. This may corroborate that modeling attribute dependencies can improve the classification results, much better when there is enough training data to discover them.

- We experimentally showed that adaptive algorithms produce k-DBCs approaching the best k-DBC induced with the same underlying learning algorithm and score but using a temporal batch learning approach. This *revolutionary* approach for updating the classifier uses all the processing power and memory to do model selection at each learning step, and hence, it is essentially optimal in terms of the quality of its induced models. On the other hand, adaptive algorithms ensure a considerable reduction of the cost of updating since the number of adaptations performed on the structure during the whole learning process is small. Results suggest that it would be more appropriate to perform adaptations on the structure only when there is some accumulated data and the search procedure is able to find new dependencies.

- Although both Adap1 and Adap2 showed a desirable behavior, results evidence that Adap2, in general, ensures a better *cost-performance* trade-off: the number of structure adaptations and the resulting errors are generally smaller. By using the `AdPreqFr4SL` with the Iterative Bayes, 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.

- We experimentally showed that adaptive algorithms performed as expected independently of the score used. For scores more biased toward simplicity, as `MDL`, the number of adaptations performed in the structure is, in most of cases, maximal, when compared with other scores. Adaptive algorithms are forced to trigger more adaptations on the structure, specially in those domains with a more strong degree of attribute dependence. On the contrary, for scores more biased toward complex models, as `BDeu` and `MLC`, the number of adaptations performed in the structure is, in most of cases, minimal. These results reflect the efforts made by the adaptive algorithms to compensate for the limitations of each particular score, thus attempting to avoid *underfitting* or *overfitting* to the current data.

- Knowing when to start searching for a new structure and when to stop adapting are the most difficult aspects of the `AdPreqFr4SL`. Rather than focusing on exact convergence, we are more interested in detecting the moment when it does not make sense

to continue adapting the model because increasing the number of training examples will not result in discovering new attribute dependencies and further improvements on the parameter estimates. The results from conducted experiments give us some evidence that the heuristics for detecting these two situations based on the observation of the `model-LC` work quite well. However, these criteria depend on two thresholds: the threshold for the gentle slope `eps1` and the threshold for the plateau `eps2`. Results with different threshold settings indicate that the choice of a particular threshold can influence the performance of the adaptive algorithms, specially for more complex domains. In less complex domains were observed only tiny differences in the performance for different parameter settings. Results suggest the use of lower thresholds for more complex domains.

- The method for handling concept drift is explicitly modeled using a Shewhart `P-Chart`, a statistically well-argued control method for monitoring the stability of one or more quality characteristics in production processes. Using batches of examples and the *batch error* as the quality characteristic to be monitored offers the advantage to detect changes by observing the deviation from the actual value to the previous observed values without paying too much attention to outliers. Results in simulated concept drift scenarios showed that the `P-Chart` is able to consistently recognize concept changes, both *abrupt* and *gradual*, and to adapt quickly to these changes, thus verifying a good *recoverability capability*.

Future Work

Issues in Sequential Updating of Bayesian Networks

An obvious topic for this line of investigation includes a more systematic investigation of adaptive issues in sequential updating of Bayesian Network structures. Many of the current limitations of the proposed adaptive algorithms come from the assumptions that we made in the implementation of the underlying incremental hill-climber algorithm for BNCs. We have assumed that we can keep in the main memory all the sufficient statistics needed to calculate the scores of candidate structures whenever a search process is invoked. This is a very strong assumption since the memory space for storing all the sufficient statistics of

Bayesian networks is huge (it is exponential in the number of variables). In future works it is desirable to include more sophisticated data structures and methods for storing and computing the sufficient statistics in an incremental fashion that will continue the line of investigation proposed, for instance, in [119]. On the other hand, it is well-known that hill-climbing searchers have the drawback of halting at local maxima. We have implemented a simple backtracking strategy in order to correct previous errors by allowing the exclusion of previously added arcs between the attributes. Nevertheless, in future works we can explore the implementation of more sophisticated backtracking methods, such as, for instance, the floating search algorithm proposed in [109] and adapting it for incremental learning.

Performing Future Subset Selection

Mainly for computational reasons in this actual implementation we have not considered performing feature subset selection. However, as stated above, feature subset selection may help in improving the Naïve Bayes performance, specially when the attribute space is highly dimensional. By reducing the number of attributes we reduce the number of parameters to be estimated, and hence, the variance of the test error. Therefore, it would be desirable in future works to use the hill-climber algorithm for learning k-DBCs including the restriction that not all the attributes can be added to the Naïve Bayes structure, as it was proposed in [8, 86].

Application to Real-World On-Line Systems

Another topic for future research would be the application of the proposed `AdPreqFr4SL` to real-world on-line systems. Although in this thesis we have only used a limited number of real-world problems from the UCI's repository, results encourage us to state that the `AdPreqFr4SL` can be used in those real-world on-line applications where it is needed to refine the initial model on the light of new data and where concept changes are likely to occur. Such a real problem is *user modeling*. User modeling systems are basically concerned with making inferences about the user's assumptions (e.g. *preferences*, *goals*, *interests*, etc.) from observations of the user's behavior during his/her interaction with the system. Observations of the user's behavior can provide data that a machine learning system can use to induce a

model designed to predict future actions [140]. There are two issues that are critical in the use of machine learning for user modeling. The first is related to the *uncertainty* of the collected information about the user. The second is related to the *changes* of the user's behavior with time. Since concept drift can be a regular phenomenon in user modeling, adaptive predictive models, capable to adapt quickly to changes in the user's behavior, are desirable.

Because of its simplicity and specially its incremental nature, the Naïve Bayes classifier has been one of the most commonly used predictive models in user modeling, namely, in *information filtering*, *recommender systems* and *student modeling*. The first two applications concern the adaptive classification of documents or objects according to a particular user interest [107, 5]. The latter involves the construction of the student model to assist adaptive learning environments [20]. However, nowadays there are only few user modeling applications using adaptive learning algorithms that can deal with concept drift. Usually the authors of existing systems address the need for adaptation by regularly rebuilding the model from scratch using the most recent stored data. Results with our adaptive algorithms evidence significant improvements on the performance of the Naïve Bayes over time and that the `AdPreqFr4SL` is able to recognize concept changes and to adapt quickly to these changes. Therefore, we believe that the `AdPreqFr4SL` framework can be successfully implemented in user modeling tasks. For instance, in [19] and [20] we presented an adaptive predictive model for a student modeling prediction task in the context of an Adaptive Educational Hypermedia System. The task consists in determining what kind of learning resources are more appropriate to a particular student according to his/her learning style. This task presents the two critical issues in user modeling: *uncertainty* and *concept drift*. To solve it we implemented our adaptive algorithms for handling concept drift in combination with the Iterative Bayes using Naïve Bayes. The results using artificial students showed that the adaptive algorithm is able to adapt quickly to the changes in the user's behavior thus reflecting the current student's preferences more accurately. In future works it is desirable the use of the `AdPreqFr4SL` and k-DBCs for this kind of user modeling prediction task.

References

[1] E. Aarts and J. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, 1997.

[2] H. Akaike. Information theory as an extension of the maximum likelihood principle. In *Proceedings of the Second International Symposium of Information Theory*, pages 267–281. Akademia Kiado, 1973.

[3] T. V. Allen. Handling uncertainty when you are handling uncertainty: Model selection and error bars for belief networks. Master's thesis, Department of Computing Science, University of Alberta, 2000.

[4] T. V. Allen and R. Greiner. Model selection criteria for learning belief nets: An empirical comparison. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1047–1054. Morgan Kaufmann Publishers, Inc, 2000.

[5] D. Billsus and M. Pazzani. A hybrid user model for news stories classifications. In *Proceedings of the Seventh International Conference on User Modeling*, pages 99–108. Springer Wien New York, 1999.

[6] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213, 1997.

[7] R. Blanco, I. Inza, and P. Larrañaga. Learning Bayesian networks in the space of structures by estimation of distribution algorithms. *International Journal of Intelligent Systems*, 18(2):205–220, 2003.

[8] R. Blanco, I. Inza, I. Merino, M.Quiroga, and P. Larrañaga. Feature selection in Bayesian classifiers for the prognosis of survival of cirrhotic patients treated with tips. *International Journal of Biomedical Informatics*, 38(5):376–388, 2005.

[9] A. Blumer et al. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[10] R. R. Bouckaert. Properties of Bayesian belief network learning algorithms. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 102–109. Morgan Kaufmann Publishers Inc, 1994.

[11] R. R. Bouckaert. Bayesian networks classifiers in Weka. Technical Report 14/2004, University of Waikato, 2004.

[12] H. Bozdogan. Model selection and Akaike's information criterion (AIC): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, 1987.

[13] D. Brain and G. I. Webb. The need for low bias algorithms in classification learning from large data sets. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, volume 2431 of *LNCS*, pages 62–73. Springer-Verlag, 2002.

[14] B. Brumen, I. Golob, H. Jaakkola, T. Welzer, and I. Rozman. Early assessment of classification performance. *Australasian Computer Science Week Frontiers*, 32:91–96, 2004.

[15] W. Buntine. Theory refinement on Bayesian networks. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann Publishers Inc, 1991.

[16] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Trans. On Knowledge And Data Engineering*, 8:195–210, 1996.

[17] K. P. Burnham and D. R. Anderson. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods and Research*, 33:261–304, 2004.

[18] G. Castillo and J. Gama. Bias management of Bayesian networks classifiers. In *Discovery Science - DS 2005, 8th International Conference, Singapore*, volume 3735 of *LNAI*, pages 70–83. Springer Verlag, 2005.

[19] G. Castillo, J. Gama, and A. M. Breda. Adaptive Bayes for a student modeling prediction task based on learning styles. In *User Modeling 2003, 9th International Conference*, volume 2702 of *LNAI*, pages 328–332. Springer Verlag, 2003.

[20] G. Castillo, J. Gama, and A. M. Breda. An adaptive predictive model for student modelling. In *Advances in Web-Based Education: Personalized Learning Environments*, pages 70–92. Information Science Publishing, Idea Group Inc., 2005.

[21] G. Castillo, J. Gama, and P. Medas. Adaptation to drifting concepts. In *Progress in Artificial Intelligence, 11th Portuguese Conference on Artificial Intelligence, EPIA 2003*, volume 2902 of *LNCS*, pages 279–293. Springer Verlag, 2003.

[22] J. Cheng and R. Greiner. Comparing Bayesian network classifiers. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 101–108. Morgan Kaufmann Publishers Inc, 1999.

[23] J. Cheng and R. Greiner. Learning Bayesian belief network classifiers: Algorithms and system. In *Proceedings of the 14th Canadian Conference on Artificial Intelligence*, volume 2702 of *LNCS*, pages 141–150. Springer Verlag, 2001.

[24] J. Cheng, R. Greiner, J. Kelly, D. Bell, and W. Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137(1–2):43–90, 2002.

[25] V. Cherkassky. *Learning from Data: Concepts, Theory and Methods*. John Wiley & Sons, Inc., 1998.

[26] D. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research Group, Nov. 1994.

[27] D. M. Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, Feb. 2002.

[28] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Information Theory*, 14(11):462–467, Nov. 1968.

[29] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[30] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., 1991.

[31] R. G. Cowell. Conditions under which conditional independence and scoring methods lead to identical selection of Bayesian network models. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 91–97. Morgan Kaufmann Publishers, Inc, 2001.

[32] R. G. Cowell. On searching for optimal classifiers among Bayesian networks. In *Proceedings of the 8th International Conference on Artificial Intelligence and Statistics*, pages 175–180. Morgan Kaufmann Publishers, Inc, 2001.

[33] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer Verlag, 1999.

[34] R. G. Cowell, A. P. Dawid, and D. J. Spiegelhalter. Sequential model criticism in probabilistic expert systems. *IEEE Trans. Pattern Anal. Mach. Intell*, 15(3):209–219, 1993.

[35] A. P. Dawid. Statistical theory: The prequential approach. *Journal of the Royal Statistical Society, Series A*, 147:278–292, 1984.

[36] E. del Castillo. *Statistical Process Adjustment for Quality Control.* John Wiley & Sons, Inc., New York, 2002.

[37] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

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

[39] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, Inc., 1973.

[40] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* Chapman and Hall, London, 1993.

[41] R. A. Fisher. *Statistical Methods and Scientific Inference.* Oliver-and-Boyd, 1956.

[42] J. H. Friedman. Bias, variance, 0-1 loss and the curse of dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

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

[44] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pages 252–262. Morgan Kaufmann Publishers, Inc, 1996.

[45] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 165–174. Morgan Kaufmann Publishers, Inc, 1997.

[46] J. Gama. Iterative Bayes. *Theoretical Computer Science*, 292(2):417–430, 2003.

[47] J. Gama and G. Castillo. Adaptive Bayes. In *Advances in Artificial Intelligence, IBERAMIA 2002*, volume 2527 of *LNAI*, pages 765–774. Springer Verlag, 2002.

[48] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence*, volume 3171 of *LNCS*, pages 286–295. Springer Verlag, 2004.

[49] D. Geiger and D. Heckerman. Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence*, 82(1–2):45–74, 1996.

[50] S. Gemann, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[51] Z. Ghahramani. Unsupervised learning. In *Advanced Lectures on Machine Learning*, volume 3176 of *LNAI*, pages 72–112. Springer Verlag, 2003.

[52] D. Grossman and P. Domingos. Learning Bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the 21th International Conference of Machine Learning*, pages 361–368. ACM Press, 2004.

[53] P. Grunwald. A tutorial introduction to the Minimum Description Length principle. In *Advances in Minimum Description Length: Theory and Applications*. MIT Press, April 2005.

[54] B. Gu, F. Hu, and H. Liu. Modelling classification performance for large data sets. In *Proceedings of the Second International Conference on Advances in Web-Age Information Management*, volume 2118 of *LNCS*, pages 317–328. Springer Verlag, 2001.

[55] J. Han and M. Kamer. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc, 2001.

[56] D. Hand and K. Yu. Idiot's Bayes - not so stupid after all? *International Statistical Review*, 69:385–398, 2001.

[57] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York, 2001.

[58] D. Heckerman. Tutorial on learning in Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research Group, 1995.

[59] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[60] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[61] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Minining*, pages 525–531. ACM Press, 2002.

[62] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Minining*, pages 97–106. ACM Press, 2001.

[63] I. Inza, P. Larrañaga, R. Blanco, and A. J. Cerrolaza. Filter versus wrapper gene selection approaches in DNA microarray domains. *Artificial Intelligence in Medicine*, 31(2):91–103, 2004.

[64] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 367–370. AAAI Press, 1996.

[65] M. I. Jordan. *Learning in Graphical Models*. MIT Press, 1999.

[66] E. Keogh and M. Pazzani. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics*, pages 225–230. Morgan Kaufmann Publishers, Inc, 1999.

[67] R. King, C. Feng, and A. Shutherland. STATLOG: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):259–287, 1995.

[68] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

[69] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proceedings of the 17th International Conference on Machine Learning*, pages 487–494. Morgan Kaufmann Publishers, Inc, 2000.

[70] R. Klinkenberg and I. Renz. Adaptive information filtering: Learning in the presence of concept drifts. In *Proceedings of the Workshop of Learning for Text Categorization, AAAI/98, ECML/98*, pages 33–40. AAAI Press, 1998.

[71] R. Kohavi. Scaling up the accuracy of naïve-Bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 202–207. AAAI Press, 1996.

[72] R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[73] R. Kohavi and D. H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Proceedings of the 13th International Conference on Machine Learning*, pages 275–283. Morgan Kaufmann Publishers, Inc, 1996.

[74] D. Koller. Learning: Parameter estimation. Available on-line at http://www.ics.uci.edu/ dechter/ics-275b/koller-handouts/handout-17.pdf.

[75] I. Kononenko. Semi-naïve Bayesian classifier. In *EWSL-91: Proceedings of the European working session on learning on Machine learning*, pages 206–219. Springer Verlag New York, 1991.

[76] P. Kontkanen. Public domain datasets and results for several supervised learning algorithms. Available on-line at http://www.cs.helsinki.fi/u/p̃kontkan/Data/.

[77] P. Kontkanen, P. Myllymäki, T. Silander, and H. Tirri. BAYDA: Software for Bayesian classification and feature selection. In *Proceedings of the 4th international conference on Knowledge Discovery and Data Mining*, pages 254–258. AAAI Press, 1998.

[78] P. Kontkanen, P. Myllymäki, T. Silander, and H. Tirri. On supervised selection of Bayesian networks. In *Proceedings of the 15th International Conference on Uncertainty in Artificial Intelligence*, pages 334–342. Morgan Kaufmann Publishers, Inc, 1999.

[79] P. Kontkanen, H. Terry, and P. Myllymaki. Comparing prequential model selection criteria in supervised learning of mixture models. In *Proceedings of the Eighth International Conference on Articial Intelligence and Statistics*, pages 233–238. Morgan Kaufmann Publishers, Inc, 2001.

[80] I. Koychev. Gradual forgetting for adaptation to concept drift. In *Proceedings of the ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*, pages 101–106. Berlin, Germany, 2000.

[81] W. Lam and F. Bacchus. Learning Bayesian belief networks. an approach based on the MDL principle. *Computational Intelligence*, 10(4):269–293, 1994.

[82] W. Lam and F. Bacchus. Using new data to refine Bayesian networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 383–390. Morgan Kaufmann Publishers, Inc, 1994.

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

[84] P. Langley and S. Sage. Induction of selective Bayesian classifiers. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 399–406. Morgan Kaufmann Publishers, Inc, 1994.

[85] C. Lanquillon. *Enhancing Text Classification to Improve Information Filtering*. PhD thesis, University of Madgdeburg, Germany, 2001.

[86] P. Larrañaga. Clasificacion supervisada via modelos graficos probabilisticos. Trabajo de investigacion presentado como documentacion para la segunda prueba de habilitacion para Catedraticos de Universidad en el area de Ciencia de la Computacion e Inteligencia Artificial, University of the Basque Country, 2002.

[87] M. Laszlo. *Computational Geometry and Computer Graphics in C++*. Prentice Hall, Upper Saddle River, NJ, 1996.

[88] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[89] P. Lee. *Bayesian Statistics: An Introduction*. A Hodder Arnold Publication, third edition, 2004.

[90] T. Leonard and J. S. Hsu. *Bayesian Methods. An Analysis for Statisticians and Interdisciplinary Researchers*, volume 5 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2001.

[91] J. Lindsey. *Parametric Statistical Inference*. Clarendon Press, Oxford, 1996.

[92] H. Linhart and W. Zucchini. *Model Selection*. John Wiley & Sons, Inc., 1986.

[93] M. G. Madden. The performance of Bayesian network classifiers constructed using different techniques. In *Proceedings of the 14th European Conference on Machine Learning, Workshop on Probabilistic Graphical Models for Classification (Available on-line at http://www.sc.ehu.es/ccwBayes/ecml-pkdd-03-workshop/call.htm)*, pages 59–70, 2003.

[94] M. A. Maloof and R. S. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41(1):27, 2000.

[95] C. Meek, B. Thiesson, and D. Heckerman. The learning-curve method applied to model-based clustering. *Journal of Machine Learning Research*, 2:397–418, 2002.

[96] R. S. Michalski. Knowledge repair mechanisms: Evolution vs. Revolution. In *Proceedings of the 3rd International Machine Learning Workshop*, pages 116–119. Boston, MA: Kluwer, 1985.

[97] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification.* Ellis, New York, 1994.

[98] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[99] D. Montgomery. *Introduction to Statistical Quality Control.* John Willey & Sons, Inc., New York, 3rd edition, 1997.

[100] A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

[101] K. Murphy. A brief introduction to graphical models and Bayesian networks. Available on-line at http://www.cs.ubc.ca/ murphyk/Bayes/bnintro.html, 1998.

[102] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Machine-readable data repository, University of California, Department of Information and Computer Science, Irvine, CA, 1992.

[103] R. E. Neapolitan. *Learning Bayesian Networks.* Pearson Prentice Hall, Upper Saddle River, NJ, 2004.

[104] K. G. Olesen, S. L. Lauritzen, and F. V. Jensen. aHUGIN: A system creating adaptive causal probabilistic networks. In *Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence*, pages 223–229. Morgan Kaufmann Publishers, Inc, 1992.

[105] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, Cambridge, 1992.

[106] M. Pazzani. Searching for attribute dependencies in Bayesian classifiers. In D. Fisher and H. Lenz, editors, *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 424–429, Ft.Lauderdale, FL., 1995.

[107] M. J. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting Web sites. In *Proceedings of the 13th National Conference on Artificial Intelligence*, volume 1, pages 54–61. AAAI Press, 1996.

[108] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc, 1988.

[109] F. Pernkopf and P. O'Leary. Floating search algorithm for structure learning of Bayesian network classifiers. *Pattern Recognition Letters*, 24(15):2839–2848, 2003.

[110] M. Plutowski. Survey: Cross-validation in theory and practice. Available on-line at http://www.emotivate.com/CvSurvey.doc, 1996.

[111] F. Provost, D. Jensen, and T. Oate. Efficient progressive sampling. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 23–32. Springer-Verlag, 1999.

[112] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3(2):131–169, 1999.

[113] S. Ramachandran. *Theory Refinement of Bayesian Networks with Hidden Variables*. PhD thesis, The University of Texas at Austin, 1998.

[114] M. Ramoni and P. Sebastiani. *Intelligent Data Analysis: An Introduction*, chapter Bayesian Methods for Intelligent Data Analysis. Springer Verlag, New York, 1999.

[115] I. Rish, J. Hellerstein, and T. Jayram. An analysis of data characteristics that affect the Naïve Bayes performance. Technical Report TR-RC21993, IBM Research Group, 2001.

[116] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15 of *Series in Computer Science*. World Scientific Publisher Company Inc, 1989.

[117] J. Rissanen. Fisher information and stochastic complexity. *Information Theory*, 42(1):40–47, 1996.

[118] R. W. Robinson. Counting unlabelled acyclic digraphs. In *Lecture Notes in Mathematics: Combinatorial Mathematics V*, volume 622, pages 28–43. Springer-Verlag, 1977.

[119] J. Roure. *Incremental Methods for Bayesian Network Structure Learning.* PhD thesis, Universitat Politecnica de Catalunya, 2004.

[120] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[121] M. Sahami. Learning limited dependence Bayesian classifiers. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 335–338. AAAI Press, 1996.

[122] M. Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, 1997.

[123] R. Sangüesa and U. Cortés. Learning causal networks from data: A survey and a new algorithm for recovering possibilistic causal networks. *AI Communications*, 10(1):31–61, 1997.

[124] J. C. Schlimmer and R. H. Granger. Beyond incremental processing: Tracking concept drift. In *Proceedings of the 5th National Conference of Artificial Intelligence*, pages 502–507. AAAI Press, 1986.

[125] G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 7(2):461–464, 1978.

[126] W. Schwarz. Economic control of quality of manufactured product. D. van Nostrand Company, Inc., Toronto, Canada, 1931.

[127] P. Sen. Estimates of the regression coefficient based on Kendall's tau. *American Statistical Association*, 63:1379–1389, 1968.

[128] C.-C. Shan. Model selection for belief networks when learning with incomplete data. Technical report, Harvard University, 2001.

[129] M. Singh and M. Valtorta. Construction of Bayesian network structures from data: a brief survey and an efficient algorithm. *International Journal of Approximate Reasoning*, 12:111–131, 1995.

[130] D. J. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.

[131] A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California at Berkeley, 1994.

[132] M. Stone. Cross-validation choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B*, 36:111–147, 1974.

[133] J. Suzuki. Learning Bayesian belief networks based on the minimum description length principle: Basic properties. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 462–470. Morgan Kaufmann Publishers, Inc, 1993.

[134] J. Suzuki. Learning Bayesian belief networks based on the minimum description length principle: an efficient algorithm using the B & B technique. In *Proc. 13th International Conference on Machine Learning*, pages 462–470. Morgan Kaufmann Publishers, Inc, 1996.

[135] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, Department of Computer Science, Trinity College Dublin, 2004.

[136] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1999.

[137] D. E. Vaughan. *Simultaneous Generalized Hill Climbing Algorithms for Addressing Sets of Discrete Optimization Problems*. Doctor of philosophy in industrial and systems engineering, Virginia Polytechnic Institute and State University, 2000.

[138] A. Webb. *Statistical Pattern Recognition*. Oxford University Press Inc., 1999.

[139] G. Webb and M. Pazzani. Adjusted probability naïve Bayesian induction. In *Proceedings of 11th Australian Joint Conference on Artificial Intelligence*, volume 1502 of *LNCS*, pages 285–295. Springer Verlag, 1998.

[140] G. Webb, M. Pazzani, and D. Billsus. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):19–29, 2001.

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

[142] G. Widmer and M. Kubat. Effective learning in dynamic environments by explicit context tracking. In *Proceedings of the European Conference on Machine Learning (ECML'93)*, volume 667 of *LNCS*, pages 227–243. Springer Verlag, 1993.

[143] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[144] WikiPedia, a web-based, free-content encyclopedia. Available on-line at http://www.wikipedia.org.

[145] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, Inc, 1999.