Incremental Learning of Tree Augmented Naive Bayes Classifiers

1 author:

Josep Roure

Fundació TecnoCampus Mataró-Maresme

34 PUBLICATIONS   **276** CITATIONS

SEE PROFILE

Title: Incremental Learning of Tree Augmented Naive Bayes Classifiers

Authors:

Josep Roure Alcobé
Departament d'Informàtica i Gestió
Escola Universitària Politècnica de Mataró,
Avda. Puig i Cadafalch 101-111,
08303 Mataró, Catalonia, Spain
roure@eupmt.es
tel: +34 93 757 44 04

Abstract:

Machine learning has focused a lot of attention at Bayesian classifiers in recent years. It has seen that even Naive Bayes classifier performs well in many cases, it may be improved by introducing some dependency relationships among variables (Augmented Naive Bayes). Naive Bayes is incremental in nature but, up to now, there are no incremental algorithms for learning Augmented classifiers.

When data is presented in short chunks of instances, there is an obvious need for incrementally improving the performance of the classifiers as new data is available. It would be too costly, in computing time and memory space, to use the batch algorithms processing again the old data together with the new one.

We present in this paper an incremental algorithm for learning Tree Augmented Naive classifiers. The algorithm rebuilds the network structure from the branch which is found to be invalidated, in some sense, by data. We will experimentally demonstrate that the heuristic is able to obtain almost optimal trees while saving computing time.

Keywords: Bayesian Classification, Augmented Naive Bayes, Incremental Learning

Topics: Classification, Bayesian Network Learning

Section: Paper Track

Incremental Learning of Tree Augmented Naive Bayes Classifiers

Josep Roure Alcobé

Departament d'Informàtica i Gestió,
Escola Universitària Politècnica de Mataró,
Avda. Puig i Cadafalch 101-111,
08303 Mataró, Catalonia, Spain
`roure@eupmt.es`

Abstract. Machine learning has focused a lot of attention at Bayesian classifiers in recent years. It has seen that even Naive Bayes classifier performs well in many cases, it may be improved by introducing some dependency relationships among variables (Augmented Naive Bayes). Naive Bayes is incremental in nature but, up to now, there are no incremental algorithms for learning Augmented classifiers. When data is presented in short chunks of instances, there is an obvious need for incrementally improving the performance of the classifiers as new data is available. It would be too costly, in computing time and memory space, to use the batch algorithms processing again the old data together with the new one.

We present in this paper an incremental algorithm for learning Tree Augmented Naive classifiers. The algorithm rebuilds the network structure from the branch which is found to be invalidated, in some sense, by data. We will experimentally demonstrate that the heuristic is able to obtain almost optimal trees while saving computing time.

1 Introduction

Classification plays an important role in the field of machine learning, pattern recognition and data mining. Classification is the task to identify the class labels for instances based on a set of features or attributes. The induction of Bayesian classifiers from data have received a lot of attention within the Bayesian Network learning field [7, 3, 8].

The simplest Bayesian Classifier is the Naive Bayes [5, 11]. It assumes that attributes are independent when the class label is known. Even it is a very strong assumption and it does not hold in many real world data sets, the Naive Bayes classifier is seen to outperform more sophisticated classifiers specially over data sets where the features are not strongly correlated [11].

More recently, a lot of effort has focused on improving the Naive Bayes classifier by relaxing independence assumptions [7, 3]. Mainly, these methods infer restricted networks among features from data. In this way, these methods combine some of the Bayesian Networks ability to represent dependencies with the simplicity of Naive Bayes. This sort of classifiers are usually called Augmented Naive Bayes.

Naive Bayes is an incremental classifier. That is, it is able to revise the classifier when new data instances are available, with neither beginning from scratch nor pro-

cessing again old data instances. This sort of learning is useful when data instances are presented in streams while the classifier still must work.

When Naive Bayes classifier is augmented, it looses the incremental property as most of the algorithms for inferring Bayesian Networks are batch [2]. In this paper, we use an incremental algorithm for learning tree-shaped Bayesian Networks to obtain an incremental Tree Augmented Naive Bayes classifier. We will show, in this paper, that the incremental version obtains most of times the same accuracy than the batch classifier while saving computational time.

Incremental learning attempts to update current Bayesian Networks in response to newly observed data instances. Langley [10] stated that an algorithm is incremental if (1) it inputs one training experience at a time, (2) does not reprocess any previous data instances, and (3) retains only one knowledge structure in memory.

Each of these three constraints aims at clear objectives. The first wants incremental algorithms to be able to output a Bayesian Network at any moment of the learning process. The second keeps low and constant the time required to process each data instance over all the data set. And finally, the third constraint wants learning algorithms not to do unreasonable memory demands.

2 Bayesian Network Classifiers

Bayesian classifiers have proven to be competitive with other approaches like nearest neighbor, decision trees or neural networks [7]. Bayesian classifiers learn from pre-classified data instances the probability of each attribute X_i given the class label C, $P(X_i|C)$. Then the Bayes rule is used to compute the probability that an example $e = < x_1, \ldots, x_n >$ belongs to a class C_i, $P(C_i|x_1, \ldots, x_n)$. In this way, the class with highest posterior probability is calculated. The independence assumptions among attributes or variables distinguish the different Bayesian classifiers.

2.1 Naive Bayes

The Naive Bayes as discussed by Duda and Hart [5] assume that all attributes are independent given the class label. This classifier can be represented by a simple Bayesian Network where the class variable is the parent of all attributes.

Given the independence assumptions, the posterior probability is formulated as

$$P(C_i|x_1, \ldots, x_n) \propto P(C_i) \prod_k P(x_k|C_i)$$

This simple expression can very efficiently be calculated and it is only needed to estimate $P(C_i)$ and $P(x_k|C_i)$ from data. To do so, we only need to keep a counter for the number of training instances, a counter for each class label, and a counter for each attribute value and class label pair.

Note that to incrementally learn the Naive Bayes classifier we only need to increase the counters as new instances are precessed. See also that the network structure is not learnt from data but fixed before hand. So, we could use the incremental approach proposed by Spiegelhalter et. al [13] in order to incrementally update the conditional probabilities of the classifier.

2.2 Tree Augmented Naive Bayes

The Tree Augmented Naive Bayes (TAN) classifier was introduced [7] in order to improve the performance of the Naive Bayes. The TAN classifier relaxes the independence assumptions having a dependence tree $\mathcal{T}$ among the attributes $x_1, \ldots, x_n$ and maintaining the class variable as a parent of each attribute.

In order to learn the TAN classifier, first it is learned the tree among the attributes and afterwards an arch is added from the class variable to each attribute. To learn the tree structure, it is used the algorithm proposed by Chow and Liu [4].

Given the independence assumptions in the tree $\mathcal{T}$, the posterior probability is

$$P(C_i|x_1, \ldots, x_n) \propto P(C_i) \prod_k P(x_k|x_{j(k)}, C_i)$$

where $x_{j(k)}$ stands for the parent of variable x_k in the tree $\mathcal{T}$, and x_0 for the null variable.

Friedman et al. [7] showed that TAN outperforms Naive Bayes while maintaining the computational simplicity on learning and classifying. We now need to keep a counter for the number of training instances, a counter for each class label, and a counter for each attribute value, parent value and class label triplet.

Note that Chow and Liu's proposal is a batch learning algorithm. In the next section we will explain the Chow and Liu batch algorithm and our incremental approach.

3 Tree-Shaped Bayesian Network Learning

The Chow and Liu's algorithm , CL algorithm from now on, estimates the underlying n-dimensional discrete probability distribution from a set of samples. The algorithm yields as an estimation a distribution of $n-1$ first order dependence relationships among the n variables, forming a tree dependence structure. It builds a maximal cost tree introducing branches into the tree in decreasing cost order.

Our incremental approach revises an already learnt tree-shaped Bayesian Network without processing the old data instances. Roughly speaking, we state that new data invalidates the old tree-shaped structure when the branches are not anymore in decreasing cost order. Then, the tree is rebuilt from the first branch found in a bad position into the order. In this way, our algorithm, can both detect the need of updating and update the current network. We will call our proposal ACO heuristic (Arches in Correct Order).

3.1 Chow and Liu's batch algorithm

The Chow and Liu's algorithm uses the mutual information as closeness measure between $P(\mathbf{X})$ and $P_\tau(\mathbf{X})$, where $P(\mathbf{X})$ is the probability distribution from a set of samples, and $P_\tau(\mathbf{X})$ is the tree dependence distribution. It is an optimization algorithm that gives the tree distribution closest to the distribution from the samples. Let us give some notation in order to explain the Chow and Liu's measure and algorithm.

Let $(m_1, \cdots, m_n)$ be a permutation of integers $1, 2, \cdots, n$, let $j(i)$ be a mapping with $0 \le j(i) \le i$, let $\mathcal{T} = (\mathbf{X}, E)$ be a tree where $\mathbf{X}(\mathcal{T}) = \{X_{m_i}|1, 2, \cdots, n\}$ is the

set of nodes, $E(\mathcal{T}) = \{(X_{m_i}, X_{m_{j(i)}}) | 1, 2, \cdots, n\}$ is the set of branches, and where X_0 is the *null* node. If we now assign the mutual information between two variables, $I(X_{m_i}; X_{m_{j(i)}})$, as a cost to every dependence tree branch, the maximum-cost dependence tree is defined as the tree $\mathcal{T}$ such that for all $\mathcal{T}'$ in $\mathcal{T}_n$, $\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}}) \geq \sum_{i=1}^{n} I(X_{m_i}; X_{m_{j'(i)}})$. Where $\mathcal{T}_n$ stands for the set of trees with n variables.

Chow and Liu used the Kruskal algorithm for the construction of trees of maximum total cost where $I(X_{m_i}; X_{m_{j(i)}})$ may represent the distance cost from node X_{m_i} to node $X_{m_{j(i)}}$. An undirected graph is formed by starting with a graph without branches and adding a branch between two nodes with the highest mutual information. Next, a branch is added which has maximal mutual information associated and does not introduce a cycle in the graph. This process is repeated until the $(n-1)$ branches with maximum mutual information associated are added as seen in Algorithm 1.

In this paper, we give a direction to all the branches of the tree. We take as the root of the tree one of the nodes of the first branch and the direction of the branches introduced afterwards goes from the node already into the structure to the one recently introduced.

Algorithm 1 CL

Require: a database D on $\mathbf{X} = \{X_{m_1}, \cdots, X_{m_n}\}$ variables
Ensure: $\mathcal{T}$ be a dependence tree structure
    Calculate $SUFF_D(\mathcal{T})$
    $\mathcal{T} = (\mathbf{V}, E)$ the empty tree where $\mathbf{V}(\mathcal{T}) = \{\emptyset\}$ and
    $E(\mathcal{T}) = \{\emptyset\}$
    Calculate costs for every pair $I(X_{m_i}; X_{m_j})$
    Select the maximum cost pair (X_{m_i}, X_{m_j})
    $\mathbf{V}(\mathcal{T}) = \{X_{m_i}, X_{m_j}\}; E(\mathcal{T}) = \{(X_{m_i}, X_{m_j})\}$
    repeat
        $B(\mathcal{T}) = \{(X_{m_i}, X_{m_j}) \mid ((X_{m_i}, X_{m_k}) \in E(\mathcal{T}) \vee (X_{m_k}, X_{m_i}) \in E(\mathcal{T})) \wedge X_{m_j} \notin \mathbf{V}(\mathcal{T})\}$
        Select the max cost pair (X_{m_i}, X_{m_j}) from $B(\mathcal{T})$
        $\mathbf{V}(\mathcal{T}) = \mathbf{V}(\mathcal{T}) \cup \{X_{m_j}\}$
        $E(\mathcal{T}) = E(\mathcal{T}) \cup \{(X_{m_i}, X_{m_j})\}$
    until $(\mathbf{V} = \mathbf{X})$

3.2 Incremental algorithm

We introduce some notation before the explanation of our algorithm. Let $N_X^D(x)$ be the number of instances in D where $X = x$. Let $\widehat{N}_X^D$ be the vector of numbers $N_X^D(x)$ for all values of X. We call the vector $\widehat{N}_X^D$ the *sufficient statistics* of the variable X, $suff_D(X)$. In the same way, the *sufficient statistics* of the tree $\mathcal{T}$, $suff_D(\mathcal{T})$, are defined as the set of vectors $\widehat{N}_{X_{m_i}, X_{m_{j(i)}}} \forall i : 0 \leq i \leq n$.

To find the maximal cost tree we need the vector numbers $\widehat{N}_{X_{m_i}, X_{m_{j(i)}}}^D$ for all the pairs of variables in $\mathbf{X}(\mathcal{T})$, we will call this set of numbers $SUFF_D(\mathcal{T})$. Note that

$SUFF_{D \cup D'}(\mathcal{T})$ can be calculated as $\widehat{N}_{\mathbf{X}}^{D} \oplus \widehat{N}_{\mathbf{X}}^{D'}$, where $\oplus$ stands for the addition of vector components.

We divide our algorithm into two steps. In the first step, the algorithm calculates the *sufficient statistics* for both old D and new D' data instances, and in the second, it revises the tree structure according to the new *sufficient statistics*.

In the first step of the algorithm, we assume that *sufficient statistics* of the old data set D are stored. Thus, in order to recover the *sufficient statistics*, $SUFF_{D \cup D'}(\mathcal{T})$, of the whole set of data instances the algorithm does not need to go through the old ones.

The second step uses a heuristic which decides to update the structure only when the arches are not in correct order. When the tree is built for the very first time using the CL algorithm, arches are introduced into the tree structure in decreasing cost order. This order $\mathcal{O}$ is stored. When new data instances D' are presented, the cost $I(X_{m_i}; X_{m_{j(i)}})$ for each branch is calculated again using the new *sufficient statistics* $SUFF_{D \cup D'}(\mathcal{T})$, and only when the order $\mathcal{O}$ does not hold anymore the structure is updated.

Algorithm 2 ACO heuristic

Require: a database D' on $\mathbf{X} = \{X_{m_1}, \cdots, X_{m_n}\}$ variables a tree structure $\mathcal{T}$, an order $\mathcal{O}$ of branches and $SUFF_D(\mathcal{T})$

Ensure: $\mathcal{T}'$ be a dependence tree structure

    Calculate $SUFF_{D \cup D'}(\mathcal{T})$

    $\mathcal{T}' = (\mathbf{V}, E)$ the empty tree where $\mathbf{V}(\mathcal{T}') = E(\mathcal{T}') = \{\emptyset\}$

    Let X_{m_h} be the root of $\mathcal{T}$

    $B(\mathcal{T}) = \{(X_{m_h}, X_{m_j}) \mid (X_{m_h}, X_{m_j}) \in E(\mathcal{T})\}$

    continue=false; *k=0*

    if $((X_{m_i}, X_{m_j})_{\mathcal{O}(1)} = arg\ max_{(X_{m_r}, X_{m_s}) \in B(\mathcal{T}) \cap E(\mathcal{T})} I(X_{m_r}, X_{m_s}))$ **then**

        $\mathbf{V}(\mathcal{T}') = \{X_{m_h}, X_{m_j}\}$; $E(\mathcal{T}') = \{(X_{m_h}, X_{m_j})\}$

        continue=true; *k=2* be the number of branches added (+ 1)

    end if

    while (*continue*) and $(k \leq |E(\mathcal{T})|)$ **do**

        $B(\mathcal{T}_{\mathcal{O}(k)}) = \{(X_{m_i}, X_{m_j}) \mid ((X_{m_i}, X_{m_k}) \in E(\mathcal{T}_{\mathcal{O}(k)}) \vee (X_{m_k}, X_{m_i}) \in E(\mathcal{T}_{\mathcal{O}(k)})) \wedge$ $X_{m_j} \notin \mathbf{V}(\mathcal{T}_{\mathcal{O}(k)})\}$

        if $((X_{m_i}, X_{m_j})_{\mathcal{O}(k)} = arg\ max_{(X_{m_r}, X_{m_s}) \in B(\mathcal{T}_{\mathcal{O}(k)}) \cap E(\mathcal{T})} I(X_{m_r}, X_{m_s}))$ **then**

            $\mathbf{V}(\mathcal{T}') = \mathbf{V}(\mathcal{T}') \cup \{X_{m_j}\}$

            $E(\mathcal{T}') = E(\mathcal{T}') \cup \{(X_{m_i}, X_{m_j})\}$; *k++*

        else

            continue=false

        end if

    end while

    if $(k \leq |V(\mathbf{X})|)$ **then**

        Continue building $\mathcal{T}'$ using the original CL algorithm

    end if

More precisely our algorithm, see Algorithm 2, inspects the arches in the order $\mathcal{O}$ they were added into the tree. When an arch $(X_{m_i}, X_{m_{j(i)}})_{\mathcal{O}(k)}$ at the k-th position in $\mathcal{O}$ has not the highest cost among all candidate arches present into the former structure, the

tree is rebuilt from that arch using the original CL algorithm. Formally, when the arch at the k-th position $(X_{m_i}, X_{m_{j(i)}})_{\mathcal{O}(k)} \neq arg\, max_{(X_{m_k}, X_{m_l}) \in B(\mathcal{T}_{\mathcal{O}(k)}) \cap E(\mathcal{T})} I(X_{m_k}, X_{m_l})$. Where $\mathcal{T}_{\mathcal{O}(k)}$ stands for the tree built only with the first $k-1$ arches of the order $\mathcal{O}$ and $B(\mathcal{T}_{\mathcal{O}(k)})$ stands for the set of arches that do not introduce any cycle in $\mathcal{T}_{\mathcal{O}(k)}$, $B(\mathcal{T}_{\mathcal{O}(k)}) = \{(X_{m_i}, X_{m_j}) \mid ((X_{m_i}, X_{m_k}) \in E(\mathcal{T}_{\mathcal{O}(k)}) \vee (X_{m_k}, X_{m_i}) \in E(\mathcal{T}_{\mathcal{O}(k)})) \wedge X_j \notin \mathbf{V}(\mathcal{T}_{\mathcal{O}(k)})\}$.

Note, it may happen that (X_{m_i}, X_{m_k}) has the maximum cost among the arches in $B(\mathcal{T}_{\mathcal{O}(k)}) \cap E(\mathcal{T})$ and not among the ones in $B(\mathcal{T}_{\mathcal{O}(k)})$. In such situation, the ACO heuristic and the CL algorithm would not recover the same tree structure.

4   Experiments

We conducted several experiments in order to compare the repeated use of the batch CL algorithm against our incremental ACO heuristic. We presented data instances to both algorithms in chunks of 100. Then we compared the Bayesian Network structures and the classifiers accuracy during all the learning process. We used five well-known datasets from the UCI machine learning repository [12]: Adult (13 attributes, 48.842 instances and 2 classes), Nursery (8 attributes, 12.960 instances and 5 classes), Mushroom (22 attributes, 8.124 instances and 2 classes), DNA (60 attributes, 3.190 instances and 3 classes) and finally Car (6 attributes, 1.738 instances and 4 classes).

We presented the instances to the algorithms in three different kind of orders. Namely, an order where similar instances are consecutive, another where dissimilar instances are consecutive, and finally a random order. We used five different orders of each kind to run both algorithms, and all numbers presented in the tables of this section are the mean and the standard deviation of the quantity being analyzed.

We used these three different kind of orders because it is widely reported in the literature that incremental algorithms may yield different results when the same instances are presented in different orders [10].

4.1   Computational time gain

The main objective of the incremental algorithm proposed was to reduce the time spent in learning a new tree structure when the system already learned one from past data. In Table 1, we compare the operations done by the batch and the incremental algorithms.

At the first two columns, we show the number of $I(X; Y)$ calculations, which is the most time consuming function of the learning algorithm. In our implementation, both batch and incremental algorithms calculate the $I(X; Y)$ amounts once, when it is firstly needed, and store them in an array. At the third and fourth columns, we show the number of times the $I(X; Y)$ is recalled. This gives an idea of the number of comparisons the algorithms perform in order to find the arch with highest $I(X; Y)$.

We can see that the number of $I(X; Y)$ calculations and recalls are much higher for the batch algorithm. Note that the number of $I(X; Y)$ calculations and recalls are the same for all runs of the batch algorithm as it always builds the tree structure from scratch, while they are different for the incremental algorithm as it builds the tree structure from the arch found in an incorrect order.

We also note from Table 1 that the more attributes data sets have the greater the gain is. Compare Adult against Nursery and Car results. And also, we can see that the gain grows with the number of data instances (see Adult results). This last point is due to the fact than when many data instances have already been processed, the new data instances slightly modify the probability distribution of the database and therefore the network structure does not need to be updated.

Another cause which may influence the time spent is the order in which the instances are presented. Usually, when similar instances are presented consecutively, the network structures learned from data are not good models of the probability distribution of the entire database. Thereof, the incremental algorithm spends more time as it must update the network structure. We see, at Table 1, that the number of $I(X;Y)$ calculations and recalls are usually higher when similar instances are ordered together.

Table 1. CPU clock ticks and operations spent in learning

		$I(X;Y)$ Calculations		$I(X;Y)$ Recalls	
		Batch	Incremental	Batch	Incremental
Adult	Rand		4801.00 (445.57)		28830.20 (1234.17)
	Sim	25350	4468.40 (564.36)	114400	24727.20 (2826.10)
	Diss		4243.00 (153.88)		19744.60 (5114.83)
Nursery	Rand		1155.40 (271.08)		3305.60 (903.06)
	Sim	2408	1216.80 (359.76)	6622	3411.20 (798.36)
	Diss		1092.80 (246.21)		3304.60 (411.21)
Mush-room	Rand		5702.20 (3085.10)		43269.40 (18451.79)
	Sim	12474	10947.80 (1383.95)	94500	73106.80 (9536.44)
	Diss		8490.40 (1839.75)		56983.20 (12287.89)
DNA	Rand		35827.60 (3042.28)		692008.60 (70438.44)
	Sim	37170	34847.60 (970.70)	754551	652884.80 (51460.44)
	Diss		33489.80 (6154.39)		647897.00 (144846.32)
Car	Rand		87.60 (49.51)		171.60 (100.51)
	Sim	165	106.40 (16.53)	330	218.80 (53.65)
	Diss		77.80 (24.10)		154.00 (74.14)

4.2 Quality of the recovered structures

In Figure 1, we show the behavior of our heuristic along the learning process where the algorithm is fed with chunks of 100 data instances, and using a random data order. We compare the structures obtained with our incremental proposal against the ones obtained with the CL batch algorithm.

Graphics present three curves. The first, shows the first arch which is not in decreasing cost order. When the number shown coincides with the number of attributes, it means that all arches are in decreasing cost order and consequently the structure is not updated. This curve gives an idea of when ACO detects that the structure must be updated. The second, shows the number of different arches between the structures learnt by the batch algorithm and the ones learnt by our incremental proposal. This curve gives us an idea of how well ACO approximates the best solution. Finally, the third curve, shows the number of arches that are different between the former and the current

tree structure learnt with the batch algorithm. This curve gives an idea of the degree in which the new 100 data instances make the current structure to change.

Looking at the figure, we discover that our incremental algorithm approximates very well the best solution. It is able to detect when the structure should be updated and is updated correctly. The third curve shows that, at the early stages of the learning process, when few data instances have already been processed, the structure changes quite a lot and that the number of changed arches tend to decrease as more data is processed. This is very well seen at the graphic of the DNA dataset. Even in this case the incremental algorithm learns structures very close to the best one. If we look back to Table 1, we can see that the incremental algorithm saves, in this case, little time as it must trigger the CL algorithm very often at firsts arches, building almost the entire tree.
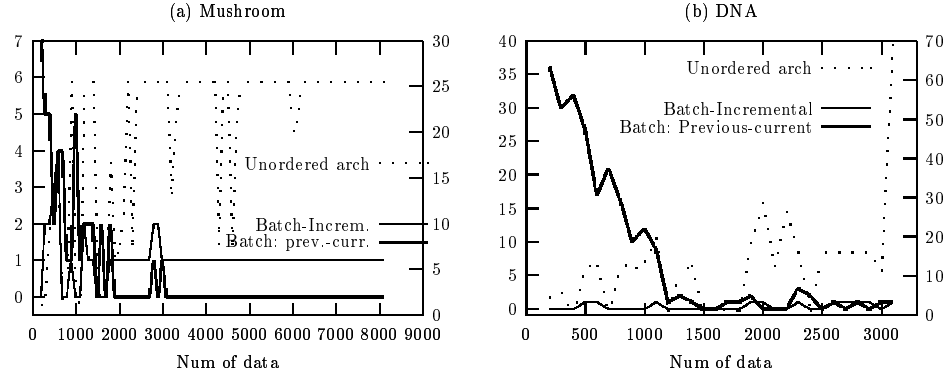

Fig. 1. Quality of recovered structures. Each graphic presents three curves; the first (*Unordered arch*), shows the first arch which is not in decreasing cost order. When the number shown co-incides with the number of attributes, it means that all arches are in decreasing cost order. The second (*Batch-Incremental*) shows the number of different arches between the trees learnt by the batch algorithm and the ones learnt with the ACO heuristic. The third curve (*Batch: Previous-current*), shows the number of different arches between the former and the current trees learnt by the batch algorithm. Note that the y axis of the first curve is on the right while the y axis of the second and third curves is on the left.

4.3   Accuracy curves

In this section, we compare the accuracy curves of the batch and the incremental algorithms when data is presented in the three different orders we explained above. In our experiments, we used two thirds of the data instances to train the classifier and the remainder for testing. In all data orders the test instances were randomly sampled.

In Figure 2, we can see the evolution of the accuracy for the DNA data set. The graphic on the left corresponds to the repeated use of the batch algorithm and the one on the right corresponds to the incremental algorithm.

Note that the shape of both graphics is almost the same. That is, our incremental classifier behaves as well as the batch one. We expected this result as the tree structures yielded by both algorithms are, most of the times, the same as shown in Figure 1.

If we compare the accuracy curves of the three different orders, we can see that the accuracy is best when data is randomly presented, while it is worse when similar instances are presented consecutively. That is due to the fact that when similar instances come together, the classifier is, at the beginning, trained with instances from one single class, and though it is not able to correctly classify instances from other classes. Lately, when new instances from other classes are used to train the classifier its accuracy is improved. Note also that the accuracy of the last learnt classifier is almost the same for the three orders.

We can see at Figure 2 that the classifier accuracy dramatically drops around the 600th instance when similar instances are presented together. That is due to overfitting, that is, the classifier is too specialized to recognize the training instances and it is not able to correctly classify the test ones.

Fig. 2. Accuracy curves. DNA data set.
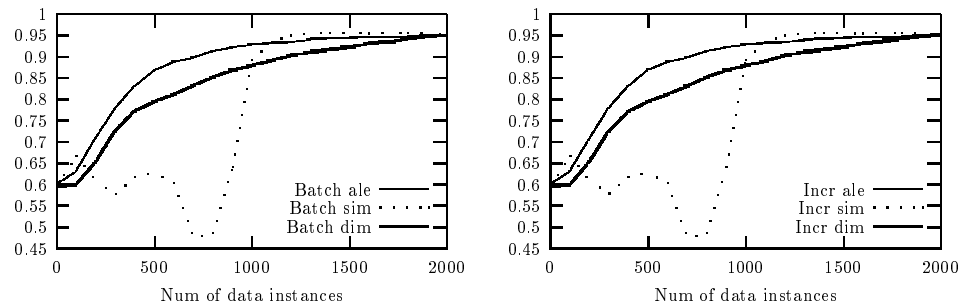
5   Discussion and Final Conclusions

Previous work on incremental learning of Bayesian Networks have focused on learning general network structures, namely, directed acyclic graphs (DAGs) [1, 6, 9]. The authors assume that the size of the *sufficient statistics* necessary to recover any possible DAG is very large and thereof it is not feasible to store them in main memory.

We presented in this paper an extension of the CL algorithm in order to incrementally learn tree-shaped Bayesian Networks. We used our algorithm to incrementally learn Tree Augmented Naive Bayes classifiers. We obtained in this way a TAN classifier which is incremental like Naive Bayes.

We claim that our algorithm is very reactive, that is, it is able to quickly detect changes in new data and to correctly update the structure. In Figure 1, we could see

that the heuristic is sound in the sense that it triggers the updating process only when changes are actually needed. We could also see in Figure 2 that the accuracy of the incremental classifier is as good as the accuracy of the batch one.

<mark>The major benefit of our incremental proposition is that it saves computing time.</mark> Even when the tree must be updated the number of calculations and the number of comparisons required is very much reduced each time a branch is checked as correctly ordered. The number of comparisons the CL algorithm must perform to order the arches is $\binom{n}{2} + \sum_{i=2}^{n} i(n-i)$, while in our proposition when the first branch is checked as correct the number of comparisons is reduced by $\binom{n}{2}$ and the number of calculations of mutual information is reduced from $\binom{2}{n}$ to a maximum of $\binom{n-1}{2}$. And when the k-th branch is checked as correct, being $1 < k < n$, the number of comparisons is reduced by $k(n-k)$ and the number of tests is reduced from $\binom{n-k}{2}$ to a maximum of $\binom{n-k-1}{2}$.

References

[1] W. Buntine. Theory refinement on Bayesian networks. In B.D. D'Ambrosio, P. Smets, and P.P. Bonisone, editors, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

[2] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Trans. On Knowledge And Data Engineering*, 8:195–210, 1996.

[3] Jie Cheng and Russell Greiner. Learning bayesian belief network classifiers: Algorithms and system. In *Proceedings of the Canadian Conference on AI*, pages 141–151, 2001.

[4] C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Teory*, 14:462–467, 1968.

[5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.

[6] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 1997.

[7] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[8] E. Keogh and M. Pazzani. Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches. In FL Ft. Lauderdale, editor, *Proceedings of the seventh International Workshop on Artificial Intelligence and Statistics*, pages 225–230, 1999.

[9] W. Lam and F. Bacchus. Using new data to refine Bayesian networks. In R. López de Mantaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 383–390, 1994.

[10] P. Langley. Order effects in incremental learning. In P. Reimann and H. Spada, editors, *Learning in humans and machines: Towards an Interdisciplinary Learning Science*. Pergamon, 1995.

[11] Pat Langley and Stephanie Sage. Induction of selective bayesian classifiers. In R. López de Mantaras and D. Poole, editors, *Proceedings of the tenth Conference on Uncertainty in Artificial Intelligence (UAI'94)*, pages 399–406. San Francisco, CA: Morgan Kaufmann.

[12] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html., 1994. Irvine, CA: University of California, Department of Information and Computer Science.

[13] D. J. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.