Incremental Augmented Naive Bayes Classifiers

Josep Roure Alcobé.[1]

Abstract. We propose two general heuristics to transform a batch Hill-climbing search into an incremental one. Our heuristics, when new data are available, study the search path to determine whether it is worth revising the current structure and if it is, they state which part of the structure must be revised. Then, we apply our heuristics to two Bayesian network structure learning algorithms in order to obtain incremental Augmented Naive Bayes classifiers. We experimentally show that our incremental approach saves a significant amount of computing time while it yields classifiers of similar quality as the ones learned with the batch approach.

1 Introduction

The incremental learning approach was firstly motivated by the human capability for incorporating knowledge from new experiences. However, nowadays there exist other reasons which increase the interest in incremental algorithms. Companies store huge amounts of data every day and batch algorithms are not able to process and incorporate to a knowledge base this great amount of continuously incoming instances in a reasonable amount of time and memory space. In this environment, *incremental learning* becomes particularly relevant since these algorithms are required to use a small constant time per record, to scan datasets only once, to use a fixed amount of memory, and to make a usable model available at any point in time [5].

Classification plays an important role in the field of machine learning, pattern recognition and data mining. The simplest Bayesian Classifier is the Naive Bayes [6, 11]. It assumes that attributes are independent when the class label is known. More recently, a lot of effort has focused on improving the Naive Bayes classifier by relaxing independence assumptions [8, 3]. Mainly, these methods infer 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.

1.1 Hill-Climbing Search

Here we revise the hill-climbing search algorithm, HCS from now on, in order to introduce the notation that we will use in the following sections. The idea of HCS is to generate a model in a step-by-step fashion by making the maximum possible improvement of an objective quality function at each step. So, we need to define the following three elements for a given problem: (i) an objective **function** $S(M, D)$ to measure the quality of a given model M with respect to a dataset D, (ii) a set of **traverse operators** $OP = \{op^1, \ldots, op^k\}$ that given an argument, A and a model M, obtain a new model $M' = op^i(M, A)$, (iii) a **domain** $\mathcal{MD}$ to define the *legal* models. For example, in Bayesian network learning, the domain is the space of DAGs, an operator is adding an arc, and an argument is the arc.

---

[1] Escola Universitària Politècnica de Mataró, Av. Puig i Cadafalch 101-111, 08303 Mataró, Catalonia, Spain. Email: roure@eupmt.es

We remark that a traverse operator op^i will be used only if it produces a model within the domain $\mathcal{MD}$. This fact guides us to define the concept of neighborhood by means of a traverse operator set,

Definition 1 (Neighborhood) *The neighborhood $\mathcal{N}(M)$ of a given model M is the set of all the alternative models that belong to the domain $\mathcal{MD}$ and that can be built by using a single operator,*

$$\mathcal{N}(M) = \{M' | M' = op^i(M, A) \wedge M' \in \mathcal{MD}\}$$

Note that a neighborhood, $\mathcal{N}(M)$, is built using a set of operator and argument pairs, $\mathbf{OpA}_{\mathcal{N}(M)} = \{(op^i, A_i) \,|\, op^i(M, A_i) \in \mathcal{D}\}$.

Now, we are able to describe HCS formally, see Algorithm 1. HCS begins with an initial model M_0 (i.e. the empty model) and iteratively constructs a sequence of models M_i, $i = 0, \ldots, n$, where each model M_i is the one with the highest score of the models in the neighborhood of M_{i-1}, $M_i = arg\, max_{M \in \mathcal{N}(M_{i-1})} S(M, D)$. HCS stops when no further improvement can be achieved, that is, when the current model has the highest score of the ones in its neighborhood.

Algorithm 1 Hill-Climbing Search

Require: a domain $\mathcal{MD}$, a database D, an initial model M_0, a scoring function $S(M, D)$, a set of operators $OP = \{op^1, \ldots, op^k\}$
Ensure: M be a model of high quality within $\mathcal{MD}$

  i=0
  repeat
    oldScore= $S(M_i, D)$
    i = i + 1
    $M_i = op_i^{k_i}(M_{i-1}, A_i)$ where $(op_i^{k_i}, A_i) = arg\, max_{(op^k, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})}} S(op_i^{k_i}(M_{i-1}, A_i), D)$
  until oldScore $\geq S(M_i, D)$

Usually, one is interested in the final model M yielded by HCS and does not bother about the intermediate ones. However, we will study the whole *search path* (i.e. the sequence of intermediate models) because we will use it in our incremental approach.

Definition 2 (Search path) *Let M_0 be an initial model, let M be the final model obtained by a hill-climbing search algorithm as*

$$M = op_n^{k_n}(\ldots (op_2^{k_2}(op_1^{k_1}(M_0, A_1), A_2)\ldots, A_n)$$

where each operator and argument pair $(op_i^{k_i}, A_i)$ yields the model with the highest score of the neighborhood, $(op_i^{k_i}, A_i) = arg\, max_{\mathbf{OpA}_{\mathcal{N}(M_{i-1})}} S(op_i^{k_i}(M_{i-1}, A_i), D)$. The search path is the sequence of operator and argument pairs $\mathcal{O}_{op} = \{(op_1^{k_1}, A_1), (op_2^{k_2}, A_2), \ldots, (op_n^{k_n}, A_n)\}$ used to build M, or equivalently, the sequence $\mathcal{M}_M = \{M_0, M_1, \ldots, M_n\}$ of intermediate models obtained with the sequence of operators and argument pairs, where $M_i = op_n^{k_i}(M_{i-1}, A_i)$.

Note that the models in the *search path* are ordered in increasing quality score order, $S(M_0, D) < S(M_1, D) < \ldots < S(M_n, D)$ and that the final model M_n is a local maximum of the domain $\mathcal{MD}$ of models.

1.2 Bayesian Network Learning

A *Bayesian network* is an annotated directed acyclic graph that encodes a joint probability distribution of a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ each of which has a domain of possible values. Formally, a Bayesian network for $\mathbf{X}$ is a pair $BN = (B_S, B_P)$ where the first component, B_S, is a directed acyclic graph (DAG) whose vertices correspond to the random variables $X_1, \ldots, X_n$, and whose edges represent directed dependencies between variables. The parents of X_i, denoted as $\mathbf{Pa}_i$, is the set of variables with an arc to X_i in the graph. The model structure yields to a factorization of the joint probability distribution for $\mathbf{X}$, $P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}_i)$. The second component, B_P, represents the parameters that quantifies the network. It has a parameter $\theta_{ijk} = P(X_i = x_i^k|\mathbf{Pa}_i = \mathbf{pa}_i^j)$ for each possible state x_i^j of X_i and for each configuration $\mathbf{pa}_i^j$ of $\mathbf{Pa}_i$.

We will revise two learning algorithms, namely, the Chow & Liu [4], (CL algorithm) to learn tree-shaped networks, and one that learns general Bayesian networks [7] that we will call GN. Both are hill-climbing searchers that begin with the arc-less network and perform the operator that most increases the score of the resulting structure and does not introduce a cycle into the network. Algorithms stop when the use of a single operator cannot increase the network's score. The difference between the algorithms is the domain of models and the operators they use.

Algorithm CL obtains a maximum spanning tree that maximizes the mutual information of variables. The neighborhood, $\mathcal{N}_{CL}(B_S)$ of a network structure, $B_S = (\mathbf{X}, E)$, is the set of all trees that can be obtained from B_S by adding a single arc $X_j \to X_i$ to B_S,

$$\mathcal{N}_{CL}(B_S) = \{(\mathbf{X}, E') \mid E' = E \cup \{(X_j, X_i)\}$$
$$\wedge \; \nexists X_k : (X_k, X_i) \in E\}$$

Algorithm GN yields full DAG structures and the neighborhood of a given network structure, B_S, is the set of all networks that can be obtained from B_S by adding, reversing or deleting a single arc $X_i \to X_j$ to B_S such that does not introduce a cycle,

$$\mathcal{N}_H(B_S) = \{(\mathbf{X}, E')|E' = E \cup \{(X_i, X_j)\} \wedge B_S' \text{is a DAG}$$
$$\vee E' = E \cup \{(X_i, X_j)\}\backslash\{(X_j, X_i)\} \vee E' = E\backslash\{(X_i, X_j)\} \}$$

CL uses the mutual information between two variables as quality function, and algorithm GN uses, in our experiments, the Bayesian Dirichlet approach (BDeu), see [2].

1.3 Bayesian Network Classifiers

Bayesian classifiers have proven to be competitive with other approaches like nearest neighbor, decision trees or neural networks [8]. 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.

The **Naive Bayes** as discussed by Duda and Hart [6] 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. The **Tree Augmented Naive Bayes** (TAN) classifier was introduced [8] in order to improve the performance of the Naive Bayes. The TAN classifier relaxes the independence assumptions having a dependence tree among the attributes $x_1, \ldots, x_n$ and maintains the class variable as a parent of each attribute. Finally, a straightforward extension of TAN is the

Bayesian Network Augmented Naive Bayes (BAN) classifier [8]. The BAN classifier further relaxes the independence assumptions having a Bayesian network BN among the attributes and maintains the class variable as a parent of each attribute. The posterior probability is of these classifiers is formulated as

$$P(C_i|x_1, \ldots, x_n) \propto P(C_i) \prod_k P(x_k|\mathbf{Pa}_k, C_i)$$

where $\mathbf{Pa}_k$ is the empty set for the Naive classifier, it is a set with one single parent for TAN, and it is an unlimited parent set for BAN.

Friedman et al. [8] showed that TAN outperforms Naive Bayes while maintaining the computational simplicity on learning and classifying. Furthermore, Cheng et al. [3] showed that BAN generally outperforms TAN and that the former is at most five times slower. Note that to incrementally learn the Naive Bayes classifier we only need to calculate its parameters since the network structure is fixed beforehand (i.e. the class variable is the single parent of each variable) and so, we could learn the parameters incrementally [14]. On the contrary, for the TAN and BAN classifiers we need to update the network structure in addition to the network parameters.

2 Incremental Hill-Climbing Search

In this section we propose two general heuristics in order to transform any batch Hill-Climbing searcher into an incremental one. Both heuristics study the search path obtained in the former learning step in order to decide whether it is worth to update the current structure and, when it is worth, they are able to determine from which point in the search path the structure should be revised.

Our heuristics relies on a desirable property of scoring functions. We usually prefer quality functions that score similarly a model structure when it is measured with respect to similar datasets, that is, sets whose underlying probability distance is close, $D_{KL}(P(D)||P(D'))$ (the Kullback and Leibler measure). For example, we do not want quality functions to be sensitive in the presence of few noisy instances in a dataset. Using this property, we can deduce that when a model has *significantly* different quality scores when it is measured with respect to two datasets, then their underlying probability distributions are also *significantly* different.

2.1 Traversal Operators in Correct Order

We call the first heuristic *Traversal Operators in Correct Order*, TOCO from now on. The TOCO heuristic assumes that it is only worth to update an already learned model when the new data alter the order of the models that are in its search path.

In our incremental approach, we will keep the search path of the former learning step obtained with a given dataset D, $S(M_0, D) < \ldots < S(M_{i-1}, D) < S(M_i, D) < \ldots < S(M_n, D)$. When new data D' are available the score of the models of the search path can be calculated again, $S(M_{\sigma(0)}, D \cup D') < \ldots < S(M_{\sigma(i-1)}, D \cup D') < S(M_{\sigma(i)}, D \cup D') < \ldots < S(M_{\sigma(n)}, D \cup D')$. Where $\sigma(i)$ is a mapping function such that $0 \leq \sigma(i) \leq n$. If the order of models in the path changes, that is, there exists $0 \leq i \leq n$ such that $\sigma(i) \neq i$ we can conclude two facts. First, we know that $D_{KL}(P(D)||P(D \cup D'))$ is bigger enough to significantly change the quality score of models and thus the new data provides additional information. Second, we know that algorithm HCS, see Algorithm 1, would follow a different search path through the space of models and possibly obtain a different one. On the contrary, when the order still holds, $\forall i \in [0, n]\; \sigma(i) = i$, we conclude that $D_{KL}(P(D)||P(D \cup D'))$ is small and thereof, we assume that algorithm HCS would follow again the same path and obtain the same

model. Thus, in the former case we trigger algorithm HCS while in the latter we do not revise the structure.

When the order of models in the path changes, we will assume that the model built by means of the correctly ordered traverse operators is still correct and thus we will use it as the initial model for algorithm HCS. More precisely, if the first model in incorrect order is M_i, the first part of the learning path, $S(M_{\sigma(0)}, D \cup D') < \ldots < S(M_{\sigma(i-1)}, D \cup D')$, is kept and the search is resumed from the last model in correct order, $M_{\sigma(i-1)}$.

So, the benefit of the TOCO heuristic is twofold. First, the model will only be revised when it is *invalidated* by new data, and second, in the case that it must be revised, the learning algorithm will not begin from scratch.

2.2 Reduced Search Space

The second heuristic applies when the current structure needs to be revised. Basically, it reduces the search space by restricting the set of operator and arguments pairs that will be considered during the next learning steps. We call this heuristic Reduced Search Space (RSS).

Given a model M, the HCS algorithm scores operator and argument pairs $\mathbf{OpA}_{\mathcal{N}(\mathcal{M})}$ in order to obtain the best model of the neighborhood $\mathcal{N}(M)$. During this process, the RSS heuristic stores, into a set $\mathcal{B}$, the $nRSS$ pairs with the best scores. So, in the next search steps, when new data are available, the HCS algorithm restricts the search to the models of the neighborhood obtained with these pairs, namely $\mathbf{OpA}_{\mathcal{N}(\mathcal{M})} \cap \mathcal{B}$. The rationale for this is that if the score of a model is very low it will not grow very much with a short chunk of new data instances specially when these instances come from the same underlying probability distribution. Note that when $nRSS$ is close to zero the search space is very much reduced and thus the model obtained may be biased towards the former one and it may be very different to the one obtained using the whole search space.

Algorithm 2 Incremental Hill-Climbing Search
Require: a domain of models $\mathcal{MD}$, a set of operators $OP = \{op^1, \ldots, op^k\}$, a database D, a scoring function $S(M, D)$, the search path of the last learning step $\mathcal{O}_{Op}$ used to obtain the former model M_{for}, the sets $\mathcal{B}_i$ for each variable, two positive integers $nRSS$ and k to state the number of pairs in $\mathcal{B}$ to consider
Ensure: M be a model of high quality in $\mathcal{MD}$
    let $(op^{k_j}_j, A_j)$ be the last ordered pair in $\mathcal{O}_{Op}$
    $M_{ini} = op^{k_j}_j(\ldots(op^{k_2}_2(op^{k_1}_1(M_0, A_1), A_2)\ldots, A_j)$
    $i = 0; M_i = M_{ini}$
    repeat
        oldScore= $S(M_i, D)$
        i = i + 1
        $M_i = op^{k_i}_i(M_{i-1}, A_i)$ where
            $(op^{k_i}_i, A_i) = arg\ max_{(op^k, A) \cap \mathcal{B}_i} S(op^{k_i}_i(M_{i-1}, A_i), D)$
        if $M_{ini} \neq M_{for}$ **then**
            Calculate $\mathcal{B}_i$ for each pair in $\mathcal{O}_{Op}$
        end if
    until oldScore $\geq S(M_i, D)$

2.3 Incremental hill-climbing search

Now we are ready to show the incremental hill-climbing search (iHCS) which incorporates the TOCO and RSS heuristics. Algorithm iHCS, see Algorithm 2, performs two different tasks. The first one corresponds to the TOCO heuristic and consists in checking whether the learning path of the previous learning step are in correct order and to state the initial model, M_{ini}, from which the search will be

resumed. The second task consists in performing the search (i.e. update the model) using the newly available data.

Note that although all the operator and argument pairs are in correct order the incremental algorithm will try to improve the model obtained in the previous learning step. It is convenient to allow iHCS algorithm to use new operators as more data are available because more complex structures (i.e. higher degree relations among variables) can reliably be evaluated. So, there are two different situations (i.e. *pre-conditions*) in which the iHCS algorithm gets to the **repeat** statement. The first one happens when the TOCO heuristic finds an operator and argument pair incorrectly ordered ($M_{ini} \neq M_{for}$) and, in consequence, the iHCS fires the revising process. The second one happens when the TOCO heuristic finds all operators in correct order ($M_{ini} = M_{for}$) and thus the former model is still *valid* and the iHCS algorithm tries to use new operators just in case that the former model can be improved. In the first situation the sets of the RSS heuristic, $\mathcal{B}$, are not updated, while in the second the RSS sets are updated since $D_{KL}(D||D \cup D')$ is big enough to provoke changes in the ranking of operator and parameter pairs.

3 Incremental TAN and BAN

Here, we adapt the TOCO and RSS heuristics to CL and GN algorithms to incrementally learn augmented classifiers. Remember that both algorithms begin with the arc-less network structure and apply the operator that most increases the quality of the current Bayesian network. In our case, to learn Augmented Naive Bayes classifiers, algorithms begin from the Naive structure, that is, a structure where the class variable is the parent of all the other variables. Recall also that algorithm CL only uses the *add* operator while algorithm GN uses *add, reverse* and *delete* operators.

Before adapting our heuristics, we want to show that the quality measures used by the CL and GN algorithms, namely the mutual information and the BDeu, score similarly a Bayesian network when it is measured with respect to two datasets with similar underlying probabilities, $D_{KL}(P(D)||P(D \cup D'))$. This can be shown using some properties of the measures. First, see that the mutual information measure proposed by Chow & Liu [4] is a special case of the MDL measure where there is no penalty term for the network complexity. Second see that the MDL is a decreasing function of the divergence, $D_{KL}(P(D)||P(B_S))$, between the probability distribution of a dataset D and the probability distribution of a Bayesian network structure B_S [10], from where the desired behavior for the mutual information is straightforward. And finally, see that under some conditions $MDL(B_S, D) = BDeu(B_S, D) + \mathcal{O}(1)$ [1].

In order to use our TOCO heuristic we keep the order in which the operators are used (the search path) to built the structure when dataset D is available. In the following learning steps, when new data D' are presented, this order is checked. If the order does not hold at a given point of the path when the operator and argument pairs are measured with respect to the old and new data $D \cup D'$, the search path is resumed from that point.

We say that the order does not hold when there exists one operator in the order that does not maximize the score. More precisely, let $(op^i, (X_m, X_n))_k$ be an operator op^i and an arc (X_m, X_n) in position k within the order $\mathcal{O}$, we say that $(op^i, (X_m, X_n))_k$ is not in correct order when exists another operator $(op^j, (X_p, X_q))_r$ in the order where $k < r$ and such that $S((op^i(B_S, (X_m, X_n)), D \cup D') < S((op^j(B_S, (X_p, X_q)), D \cup D')$.

We experimentally saw that this condition of order among arcs was too strong in algorithm GN. That is, in many cases the order of operators did not hold and when the learning process was fired,

it yielded the same operators in a different order. This provoked to fire the revising process when in was not actually needed and thus to spend unnecessary computing time. For this reason, we relaxed this condition by means of a *window* when checking the order. That is, when new data are available we do not require the best operator to be exactly at the same place in the order but in a place nearby. In this way, the k-th operator in the order $\mathcal{O}$ will be considered correct if the score with respect to the new data, $D \cup D'$, is between the $(k - nTOCO)$-th and the $(k + nTOCO)$-th position, where $nTOCO$ is a parameter. The intuition is that we need more evidence that an operator is incorrect to revise the structure. Note that if the window length is too large, the $TOCO$ heuristic will never detect changes in the order and thus the revising process will not be fired.

In order to introduce the RSS heuristic into algorithms CL and GN, we keep for each variable as many lists of operator candidates as parents the variable has. So, when a revision is performed, the algorithm will only try those operators in the lists of candidates. The number of operators stored into the lists in order to restrict the search is specified by means of a parameter $nRSS$.

4 Experimental Results

In this section we compare the accuracy of the batch classifiers against the corresponding incremental ones. We used the datasets Adult(48.842 instances, 13 variables and 2 classes), Car (1.738 inst., 6 var. and 4 cl.), Letter Recognition (20.000 inst., 16 var. and 26 cl.), Mushroom (8.124 inst., 23 var. and 2 cl.) and Nursery (12.960 inst., 9 var. and 5 cl.) from the UCI machine learning repository [13], We used the discretize utility of MLC++ [9] in order to discretize the attributes of the Adult and Letter Recognition datasets.

We trained both TAN and BAN with 2/3 of the dataset instances and the rest to calculate the accuracy. These last ones were randomly sampled from the datasets. We fed training instances to both algorithms in chunks of 100 and calculated the accuracy of the classifiers obtained along all the learning process. We also presented training instances in different orders to test the behavior of our incremental approach. Incremental algorithms may suffer from *ordering effects* due to the nature of the incremental processing of data combined with the tendency of HCS to get stuck at local maxima. It is known that incremental HCS may output a very skewed model when the first observed samples give a biased view of the domain even in cases when the later samples give a correct view. We fed the instances to the algorithms in four different kind of orders. Namely, an order where similar instances are consecutive, another where dissimilar instances are consecutive, another where instances from the same class are consecutive and finally a random order. We used five different orders of each kind to run both algorithms, and the results shown in this section correspond to the mean and the standard deviation of the accuracy.

The main objective of our incremental algorithms is to reduce the time spent in learning a new network structure when the system already learned one from past data and still produce classifiers with high accuracy. In Table 1 and Table 2 we present the accuracy obtained with both the batch and incremental approaches of TAN and BAN classifiers, respectively. In Table 1, we also present the accuracy obtained with the Naive Bayes classifier to see the improvement obtained with *augmented* classifiers. In Table 2, we present the results obtained with the $nTOCO$ parameter set to 0 and to 3, in order to see the behavior of the TOCO heuristic. In this table, the standard deviation is only shown for the accuracy where the BAN algorithm was run with $nTOCO = 3$ for a lack of space. The standard deviation of the other columns was seen to be slightly lower than the one shown. Columns "**TG%**" and "**CG%**" show the time

Table 1. Results with Naive Bayes and TAN classifiers

| | | NAIVE | TAN | | | |
		Accuracy	Acc. Batch	Acc. Incr.	TG%	CG%
A	Rnd	0.838 (0.01)	0.860 (0.00)	0.859 (0.00)	63.61	80.34
	Sim	0.833 (0.00)	0.859 (0.00)	0.856 (0.01)	65.53	82.37
	Dis	0.864 (0.04)	0.864 (0.01)	0.862 (0.01)	64.93	83.10
	Cl	0.828 (0.05)	0.841 (0.05)	0.840 (0.04)	67.54	81.20
C	Rnd	0.848 (0.03)	0.930 (0.02)	0.925 (0.02)	100	46.91
	Sim	0.830 (0.03)	0.930 (0.02)	0.929 (0.03)	0.00	35.52
	Dis	0.848 (0.03)	0.940 (0.01)	0.932 (0.02)	-200	52.85
	Cl	0.830 (0.05)	0.908 (0.05)	0.907 (0.04)	-140	38.18
L	Rnd	0.746 (0.01)	0.860 (0.01)	0.860 (0.01)	50.65	60.57
	Sim	0.751 (0.00)	0.864 (0.01)	0.864 (0.01)	49.21	59.00
	Dis	0.758 (0.02)	0.865 (0.01)	0.865 (0.01)	56.19	65.96
	Cl	0.740 (0.01)	0.857 (0.01)	0.857 (0.01)	45.53	51.10
M	Rnd	0.981 (0.01)	1.000 (0.00)	1.000 (0.00)	34.13	54.29
	Sim	0.983 (0.00)	1.000 (0.00)	1.000 (0.00)	6.08	12.24
	Dis	0.982 (0.00)	1.000 (0.00)	1.000 (0.00)	30.67	31.94
	Cl	0.984 (0.01)	1.000 (0.00)	1.000 (0.00)	42.25	29.64
N	Rnd	0.894 (0.01)	0.927 (0.01)	0.927 (0.01)	11.44	52.02
	Sim	0.908 (0.00)	0.935 (0.01)	0.937 (0.01)	19.60	49.47
	Dis	0.893 (0.01)	0.925 (0.01)	0.920 (0.02)	47.04	54.62
	Cl	0.894 (0.02)	0.930 (0.01)	0.934 (0.01)	62.96	61.46

Table 2. Results with BAN classifier

| | | nTOCO=0 | | | | nTOCO=3 | | |
		Ac.B	Ac.I	TG%	CG%	Acc. Incr.	TG%	CG%
A	R	0.865	0.861	95.40	83.11	0.862 (0.00)	96.12	83.73
	S	0.863	0.862	89.36	78.59	0.858 (0.00)	96.93	84.56
	D	0.867	0.866	86.42	76.94	0.867 (0.01)	95.84	83.31
	C	0.845	0.842	89.26	78.07	0.842 (0.06)	89.26	80.24
C	R	0.853	0.853	37.50	45.87	0.853 (0.03)	46.15	45.87
	S	0.875	0.871	71.43	43.01	0.872 (0.08)	79.17	47.18
	D	0.860	0.860	43.75	44.75	0.860 (0.04)	39.73	49.85
	C	0.839	0.839	26.32	46.55	0.839 (0.06)	30.30	46.55
L	R	0.822	0.812	66.83	79.19	0.797 (0.04)	67.66	80.79
	S	0.814	0.779	70.04	79.73	0.772 (0.01)	71.54	80.77
	D	0.822	0.794	88.59	82.17	0.792 (0.02)	89.24	82.67
	C	0.810	0.783	80.56	73.53	0.774 (0.03)	93.50	84.02
M	R	1.000	1.000	8.17	19.78	1.000 (0.00)	79.41	71.93
	S	1.000	1.000	2.85	9.08	1.000 (0.00)	47.51	50.62
	D	1.000	1.000	4.30	12.99	1.000 (0.00)	62.48	62.08
	C	1.000	1.000	4.26	13.33	1.000 (0.00)	64.17	63.14
N	R	0.913	0.917	80.46	65.33	0.914 (0.01)	81.11	66.17
	S	0.922	0.917	77.46	63.29	0.915 (0.02)	75.38	68.24
	D	0.908	0.908	79.86	64.60	0.905 (0.01)	78.94	68.78
	C	0.917	0.917	77.74	67.70	0.918 (0.01)	83.99	70.07

gain and the gain in the number of calls to the scoring function performed by the algorithms, respectively. The columns are calculated as $(1 - nI/nB) \times 100$, where nI and nB are the number of clock ticks or calls performed by the incremental and the batch approach respectively. We believe that the number of calls are a good measure of the time spent because it gives an idea of the number of the different network structures being considered by the search algorithms. Note also that for small datasets (i.e. Car) where the computer use few clock ticks the time gain does not seem to correspond to the call gain. Another advantage of using this number is that it is decoupled from the specific architecture of the computer being used.

We can see in the tables that for both algorithms, the incremental approaches save a significantly amount of CPU clock ticks while the accuracy of the final Bayesian classifiers is very close to the ones obtained with the batch approaches. We also note from the tables that the more complex a domain is, that is, the more attributes and the larger the range of attributes, the greater the gain is. Also note that the gain grows with the number of data instances. This last point is due to the fact than when many data instances have already been processed, the new incoming data instances slightly modify the data distribution and therefore the network structure does not have to be updated. Though observe, in Table 1 and in Table 2 the columns that correspond to $nTOCO = 0$, that we obtain the lowest time gain with the Mushroom dataset. Analysing the structures returned by the incremental algorithms we saw that it is due to the fact that there are many arcs that bring similar quality to the network and this provokes that the order of operators in the search path change very often when new data are considered. In order to obtain a higher time gain, we relaxed the TOCO heuristic (setting $nTOCO = 3$) for the BAN

classifier. Table 2 shows that the accuracy for the Mushroom dataset is exactly the same (i.e. 100%) while the time gain grows from about 5% when $nTOCO = 0$ to about 60% when $nTOCO = 3$. Despite of this, we observed that the score quality of the Bayesian network structures returned by the incremental algorithm is a bit worse in the latter case. The accuracy do not decrease because the returned structures still are of high quality and the class with highest posterior is the same. This behavior is observed for all datasets, being the accuracy slightly worse only for the Letter database.

We want to remark that our incremental approaches do not suffer from ordering effects, that is, they are able to recover from biased past Bayesian classifiers yielding final structures of almost the same accuracy as the batch approaches do, independently of the order in which data instances are presented to the learning algorithms. This is true for all along the learning path.
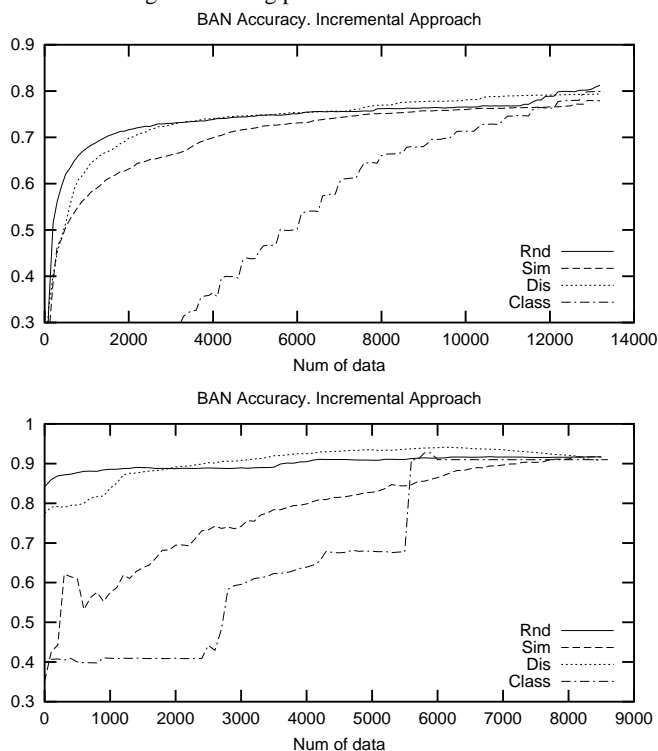
Figure 1. Learning path of Letter (top) and Nursery (bottom) datasets

Figure 1 shows the learning paths of the incremental classifiers for the Letter and the Nursery datasets and for the four data orders. We can see that for the random, similar and dissimilar orders the accuracy grows asymptotically being the curve of the similar order the one that grows slowest among the three. The curve that corresponds to the order where the instances of the same class are consecutive is shaped like a staircase. The parts of the curve that rapidly grow correspond to the moments where the first instances of a class are processed and consequently instances from that class are correctly classified. The plain parts of the curve correspond to the moments where the classes have already been *learned* and new instances from the same class are still being processed. In this figure we can see again that our incremental algorithm is not order dependent since all the learning curves reach almost the same accuracy.

5 Discussion and Final Conclusions

There are few works on incremental Bayesian network learning and here we will briefly comment the most relevant from our viewpoint. Buntine [2] stores in memory the most promising networks and re-

stricts the search among them when new data is available performing a sort of beam search. Friedman et al. [7] use *two-way* operators (i.e. add, delete and reverse arcs) to obtain a new network from the current one. These operators allow them to reconsider the arcs added during the past learning steps and to perform a sort of back-tracking to revise the current structure.

We have experimentally seen that our incremental proposal is robust in front of different instance orders. Also we claim that our proposal is very reactive to data distribution changes and consequently, when new data provides new information, the current classifier is updated obtaining a new classifier with almost the same accuracy than the one obtained with the batch approach. This is done saving a significant amount of computing time.

We also want to comment that in order to avoid processing data multiple times we stored *sufficient statistics* in memory by means of AD-tree structures [12]. However, the full discussion of our approach to AD-trees is beyond the scope of this paper.

As a final conclusion, we would like to remark that TOCO and RSS heuristics are a general method that allows to transform any batch hill climbing searcher into an incremental one, and we believe that our approach can be successfully applied to learn incrementally from data other complex models.

REFERENCES

[1] R.R. Bouckaert, *Bayesian belief networks: from inference to construction*, Ph.D. dissertation, Utrecht University, 1995.

[2] W. Buntine, 'Theory refinement on Bayesian networks', in *Proceedings of the 7th UAI*, (1991).

[3] Jie Cheng and Russell Greiner, 'Learning Bayesian belief network classifiers: Algorithms and systems', in *Proceedings of the Canadian Conference on AI*, pp. 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] Domingos and Hulten, 'A general framework for mining massive data streams', *Journal of Computational and Graphical Statistics*, **12**, (2003).

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Wiley & Sons, New York, 2nd edn., 2001.

[7] N. Friedman and M. Goldszmidt, 'Sequential update of Bayesian network structure', in *Proceedings of the 13th UAI*, (1997).

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

[9] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger, 'MLC++: A Machine Learning library in c++', in *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, pp. 740–743. IEEE Computer Society Press, (1994).

[10] W. Lam and F. Bacchus, 'Learning Bayesian belief networks. an approach based on the MDL principle', *Computational Intelligence*, **10**(4), 269–293, (1994).

[11] Pat Langley and Stephanie Sage, 'Induction of selective Bayesian classifiers', in *Proceedings of the tenth Conference on Uncertainty in Artificial Intelligence (UAI'94)*, eds., R. López de Mantaras and D. Poole, pp. 399–406. San Francisco, CA: Morgan Kaufmann, (1994).

[12] Andrew W. Moore and Mary S. Lee, 'Cached sufficient statistics for efficient machine learning with large datasets', *Journal of A.I. Research*, **8**, 67–91, (1998).

[13] P.M. Murphy and D.W. Aha. UCI repository of Machine Learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html, 1994.

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