

# Voting Massive Collections of Bayesian Network Classifiers for Data Streams

Remco R. Bouckaert

Computer Science Department, University of Waikato, New Zealand  
remco@cs.waikato.ac.nz

**Abstract.** We present a new method for voting exponential (in the number of attributes) size sets of Bayesian classifiers in polynomial time with polynomial memory requirements. Training is linear in the number of instances in the dataset and can be performed incrementally. This allows the collection to learn from massive data streams. The method allows for flexibility in balancing computational complexity, memory requirements and classification performance. Unlike many other incremental Bayesian methods, all statistics kept in memory are directly used in classification. Experimental results show that the classifiers perform well on both small and very large data sets, and that classification performance can be weighed against computational and memory costs.

## 1 Introduction

Bayesian classifiers, such as naive Bayes [5] can deal with large amounts of data with minimal memory requirements and often perform well. The assumption underlying naive Bayes that all attributes are independent given the class value often does not hold and by incorporating more dependencies, its performance can be improved. Tree augmented naive Bayes (TAN) [3], super parent TAN [6] and forest augmented naive Bayes [7] provide such methods. However, these methods are not well suited for incremental learning since the structure of the network cannot be fixed.

Averaged One-Dependence Estimators (AODE) [8] can be considered as voting of various Bayesian classifiers, one for each attribute in the data set. It performs well on both small and large data sets and can be trained incrementally. This raises the expectation that voting more Bayesian classifiers can improve performance. In this article, we analyze how to vote a collection of forest augmented naive Bayes (FAN) under some mild conditions. The number of FANs is exponential in the number of attributes making it seemingly impossible to use in a practical way. However, we show how to calculate the vote in quadratic time instead of exponential. We also consider some other collections. The goal of this paper is to design a classifier that is able to learn and classify from massive amounts of data, is able to incrementally update the classifier, show good classification performance on both massive and small data sets, has a reasonable memory requirement, and can be able to incorporate external information about the model.

The following section introduces Bayesian classifiers formally. In Section 3, we present Bayesian network classifiers for streams and explains how to perform calculation efficiently. Section 4 presents some experimental results and discussion and we conclude with some remarks and directions for further research.

## 2 Bayesian Network Classifiers

Given a set of *attributes*  $\{X_1, \dots, X_n\}$  with values  $\{x_1, \dots, x_n\}$  we want to predict the value of a *class* variable  $Y$  taking one of  $k$  class values  $\mathcal{C} = \{y_1, \dots, y_k\}$ . We assume attributes and classes to be discrete. In order to do so, we train a classifier using a data set  $D$ , which is a set of  $m$  tuples  $\{x_{1j}, \dots, x_{nj}, y_j\}$  ( $1 \leq j \leq m$ ) of values of attributes and a value of the class.

A *Bayesian network*  $B$  over a set of variables  $V$  consists of a network structure  $B_S$ , which is a directed acyclic graph over  $V$  and a set of conditional probability table  $B_P$ , one table  $P(X|pa_X)$  for each variable in  $X \in V$  conditioned on the variables that form the parent set  $pa_X$  in  $B_S$ . A Bayesian network defines a probability distribution over  $V$  by the product of the attributes  $P(V) = \prod_{X \in V} P(X|pa_X)$ . A Bayesian network can be used as a classifier by training the network over the attributes and class variables on the data and use the represented distribution using the following decision rule:  $argmax_{y \in \mathcal{C}} P(Y|X_1, \dots, X_n)$  which, using Bayes rule, is the same as taking  $argmax_{y \in \mathcal{C}} P(Y, X_1, \dots, X_n)$  which equals  $argmax_{y \in \mathcal{C}} \prod_{X \in \{Y, X_1, \dots, X_n\}} P(X|pa_X)$ . So, an instance  $\{x_1, \dots, x_n\}$  is classified by plugging in the values in the Bayesian network and taking the class value with the highest probability represented by the network.

In general, there are two learning tasks when training a Bayesian network: learning the network structure, and learning the conditional probability tables. Network structures can be fixed, as for naive Bayes, or learned based on dependencies in the data, as for tree augmented naive Bayes. The conditional probability tables are typically learned using an estimator based on the counts in the data  $D$ . Throughout, we use the Laplace estimator which estimates the probability of  $X$  taking value  $x$  given its parents taking values  $pa_x$  to be

$$P(x|pa_x) = \frac{\#(x, pa_x) + 1}{\#(pa_x) + |X|}$$

where  $\#(x, pa_x)$  is the count of instances in  $D$  where  $X$  takes value  $x$  and its parents take values  $pa_x$ . Likewise  $\#(pa_x)$  is the count of parents taking value  $pa_x$ . We use  $|X|$  to denote the number of values that  $X$  can take. Note that if the parent set of  $X$  is empty, we have by convention that  $\#(pa_x) = 0$  and  $\#(x, pa_x)$  is the count of  $X$  taking value  $x$ . One of the benefits of this estimator is that we can keep the counts  $\#(x, pa_x)$  and  $\#(pa_x)$  in memory and easily update them with each new incoming instance. So, updating the probability tables incrementally is trivial.

In the remainder of this section, we consider some popular Bayesian classifiers and explain how they relate to the Bayesian network classifier.

*Naive Bayes.* The naive Bayes classifier [5] can be considered a Bayesian network with a fixed network structure where every attribute  $X_i$  has the class as its parent and there are no other edges in the network. Therefore, naive Bayes has the benefit that no effort is required to learn the structure and the probability tables can be updated online by keeping counts in memory.

*Tree Augmented Naive Bayes.* The disadvantage of the naive Bayes classifier is that it assumes that all attributes are conditionally independent given the class, while this often is not a realistic assumption. One way of introducing more dependencies is to allow each (except one) attribute to have an extra parent from among the other attributes. Thus, the graph among the attributes forms a tree, hence the name tree augmented naive Bayes (TAN) [3]. The network structure is typically learned by determining the maximum weight spanning tree where the weight of links is calculated using information gain. This algorithm is quadratic in the number of attributes. Also, it requires gathering of the counts  $\#(x, pa_x)$  and  $\#pa_x$ , where  $pa_x$  takes values for each class and attribute  $Z$  with  $Z \in \{X_1, \dots, X_n\} \setminus X$ . So, we need to keep track of a lot of counts not ultimately used in the classifier. Further, incremental updating requires retraining the network structure, which is  $O(n^2)$  in computational complexity.

*Forest Augmented Naive Bayes.* The TAN cannot distinguish between relevant links in the final tree and irrelevant links. The forest augmented naive Bayes (FAN) [7] is a network structure where each attribute has at most one extra attribute (apart from the class) as parent, but it can as well have no other parent. The benefit is that such a structure can be learned using a greedy approach with a scoring based metric like Bayesian posterior probability, minimum description length, Akaike information criterion, etc which provide a natural stopping criterion for adding edges to the network. Still, since all possible edges need to be considered, like for TANs many counts need to be kept in memory which are not utilized in the probability tables.

*Super Parent Classifier.* The super parent classifier [6] selects one of the attributes and calls this the super parent. Each of the other attributes get this attribute and the class as its parent set while the super parent only has the class as its parent. Determining which parent becomes the super parent requires collecting many counts that are discarded as for TANs.

*Averaged One Dependence Classifiers.* AODE classifiers [8] uses the super parent idea to build a collection of  $n$  Bayesian classifiers by letting each of the attributes be super parent in one of the networks. The class is predicted by taking the sum of the probabilistic votes of each of the networks. Clearly, the benefit is that all counts are utilized and no effort is spent in learning network structures.

### 3 Bayesian Network Classifiers for Data Streams

In the previous section, we shortly listed a number of popular Bayesian network architectures. In this section, we set out to design a Bayesian classifier that can deal with the following requirements;

- be able to learn from massive amounts of data,
- be able to classify massive amounts of data,
- be able to incrementally update the classifier when more data becomes available,
- show good classification performance,
- have a reasonable memory requirement,
- be able to incorporate external information about the model.

Naive Bayes is a fast learner and classifier, but lacks in performance. TANs, FANs and super parents require quadratic time in learning a network structure, which makes incremental updating cumbersome. AODEs perform well in most areas though classification time is quadratic in the number of attributes and it lacks the ability of incorporating prior information. An algorithm that performs well in terms of the first four requirements is the Hoeffding tree algorithm [2]. However, because it grows trees incrementally, it can require large amounts of memory. Also, incorporating knowledge obtained from experts or prior experience is awkward. The Hoeffding criterion can be applied to Bayesian network learning [4] thereby incrementally learning the network structure. Still, many statistics need to be kept in memory thereby slowing down the number of instances that can be processed per second. Also, single Bayesian networks tend to be less accurate classifiers than ensembles like AODE.

This last general observation leads us to try to use a collection of a large number of Bayesian networks. Consider the collections of FANs where the order of attributes  $\{X_1, \dots, X_n\}$  is fixed in such a way that an edge between two attributes always goes from the lower ordered to the higher ordered, that is the network can only contain edges  $X_i \rightarrow X_j$  if  $i < j$ . Now consider the classifier that sums over all such FANs

$$P(Y, X_1, \dots, X_n) = C \sum_{FAN} P_{FAN}(Y, X_1, \dots, X_n) \quad (1)$$

where  $C$  is a normalizing constant. An attribute  $X_i$  can have  $i$  different parent sets, namely only the class variable  $\{Y\}$  or the class with one of the previous attributes  $\{Y, X_1\}, \dots, \{Y, X_{i-1}\}$ . So, in total there are  $n!$  such FANs to sum over. Obviously, this is not a practical classifier for even a moderate number of attributes. However, we show using induction that the sum (1) can be rewritten as

$$CP(Y) \prod_{i=1}^n \left( P(X_i|Y) + \sum_{j=1}^{i-1} P(X_i|C, X_j) \right) \quad (2)$$

The induction is by the number of attributes. For a single attribute  $n = 1$ , there is only one network structure and trivially Equations (1) and (2) are equal. For  $n-1 \geq 1$  attributes, suppose that (1) and (2) are equal and we have  $n$  attributes. Now observe that

$$\sum_{FAN} P_{FAN}(Y, X_1, \dots, X_n) = \sum_{FAN} P(Y) \prod_{i=1}^n P(X_i|pa_i^{FAN})$$

where  $pa_i^{FAN}$  is the parent set of  $X_i$  in the FAN.  $X_n$  can be taken outside the product, giving

$$\sum_{FAN} P(Y) \prod_{i=1}^{n-1} P(X_i|pa_i^{FAN})P(X_n|pa_n^{FAN}) \quad (3)$$

Now, observe that for every sub-forest over the first  $n - 1$  attributes there is one network in the set  $FAN$  where  $X_n$  has no parents but  $Y$ , one where  $X_n$  has  $Y$  and  $X_1$  as parent, one where  $Y$  and  $X_2$  are parents, etc, and one where  $Y$  and  $X_n$  are parents. Let  $FAN'$  represent the set of all sub-forests over the first  $n - 1$  attributes. Then, we can rewrite Equation 3 as

$$\left( \sum_{FAN'} P(Y) \prod_{i=1}^{n-1} P(X_i|pa_i^{FAN'}) \right) \cdot \left( P(X_n|Y) + \sum_{j=1}^{n-1} P(X_n|C, X_j) \right)$$

and by our induction hypothesis  $\sum_{FAN'} P(Y) \prod_{i=1}^{n-1} P(X_i|pa_i^{FAN'})$  is equal to  $P(Y) \prod_{i=1}^{n-1} \left( P(X_i|Y) + \sum_{j=1}^{i-1} P(X_i|C, X_j) \right)$ , so the above equals

$$P(Y) \prod_{i=1}^{n-1} \left( P(X_i|Y) + \sum_{j=1}^{i-1} P(X_i|C, X_j) \right) \left( P(X_n|Y) + \sum_{j=1}^{n-1} P(X_n|C, X_j) \right)$$

which by taking the last part inside the product equals Equation (2).

Note that the calculation of (2) is only quadratic in the number of attributes. So, we can sum over  $n!$  FANs in  $O(n^2)$  time, which means we can perform classification efficiently. No network structure learning is required, just maintaining counts  $\#(X_i, X_j, C)$  and  $\#(X_j, C)$  to calculate the probabilities in (2) is needed. So, only a fixed amount of memory is quadratic in  $n$  required.

### 3.1 Collection Architectures

So far, we only considered the collection of all FANs for a given node ordering. However, the same derivation as above can be done for any collection of parent sets, giving rise to a choice of collection architectures. Let  $PA_i$  be a set of  $i_n$  parent sets  $PA_i = \{pa_{i,1}, \dots, pa_{i,i_n}\}$  for node  $X_i$  such that all nodes in any parent set  $X_j \in pa \in PA_i$  is lower ordered than  $X_i$  (that is,  $j < i$ ). Then, if we have such a parent set for each attribute, we can sum over the collection of all Bayesian networks with any combination of these parent sets. The distribution (corresponding to Equation 2) is

$$P(Y, X_1, \dots, X_n) = CP(Y) \prod_{i=1}^n \sum_{pa_i \in PA_i} P(X_i|C, pa_i)$$

Note that inserting parent sets that do not contain the class node is not useful, since for the purpose of classification the terms  $P(X_i|pa_i \setminus \{C\})$  are constant, so these do not impact the class probabilities.

Which collection architectures can be expected to result in well performing classifiers? The collection of all FANs we considered is essentially the set of all networks where each attribute has the class as parent and at most one other (lower ordered) attribute. We abbreviate such FAN collection as FANC. A natural extension is to consider all networks with the class plus *at most* two other lower ordered attributes, and call this a two-parent collection or TC for short. The number of parent sets grows quadratically with the number of attributes, and the memory requirements for storing the conditional probability tables for  $P(X_i|C, X_j, X_k)$  grows cubic in the number of attributes. So, it may be quite costly when the number of attributes is large. A reasonable alternative is to identify one attribute as super parent and consider for every attribute of the class, possibly the super parent and possibly one other lower ordered attribute. We call this the super parent collection, or SPC for short.

### 3.2 Practical Miscellany

We shortly discuss some pragmatic issues.

*Missing values.* When there are attributes with missing values we can easily take these attributes out of the network by just ignoring the terms containing the attributes. So, terms  $P(X_i|Y, X_j)$  and  $P(X_i|Y)$  where  $X_i$  or  $X_j$  are missing in the instance are simply removed from Equation 2.

*Non-discrete attributes.* Though the above technique works for non-discrete attributes as well, efficient representation of such attributes requires some care. For the sake of simplicity, we concentrate on discrete variables.

*Prior knowledge.* In the FANC implementation of Equation 2 each term has the same weight. When there is strong evidence that some attributes, say  $X_i$  and  $X_j$  ( $j > i$ ) are highly related given  $Y$ , the term  $P(X_j|Y, X_i)$  can be given a higher weight in (2) than the other terms. Alternatively, the parent set collection can be tuned such that all parent sets of  $X_i$  contains  $X_j$ .

*Attribute ordering.* In preliminary experiments, we found that the order of attributes had moderate impact on the performance of the FANC classifier. This issue is open for further research.

*Super parent selection.* In our experiments we found that the choice of the super parent has considerable impact on the performance. Selecting the super parent based on classification accuracy on the training set would give many draws. In our experiments, taking the sum of class probabilities (i.e.  $\sum_{i=1}^m P(y_i|x_{1i}, \dots, x_{ni})$ ) as selecting criteria led to few draws and well performing classifiers. We call this SPC variant SPCr. This appears to work well for small datasets in batch learning, but is harder to implement incrementally for large datasets.

The following table gives an overview of the memory requirements, training time on a data set and test time of an instance with  $a$  representing the maximum number of values of attributes.

	NB	TAN	AODE	FANC	SPC	SPCr	TC
Memory	$O(ma)$	$O(ma^2)$	$O(m^2a^2)$	$O(m^2a^2)$	$O(m^2a^3)$	$O(m^2a^3)$	$O(m^3a^3)$
Train time	$O(nm)$	$O(nm^2)$	$O(nm^2)$	$O(nm^2)$	$O(nm^2)$	$O(nm^3)$	$O(nm^3)$
Test time	$O(m)$	$O(m)$	$O(m^2)$	$O(m^2)$	$O(m^2)$	$O(m^2)$	$O(m^3)$

## 4 Empirical Results

We evaluate the Bayesian classifiers both for small and large datasets to get an impression of its performance in a range of circumstances, both in batch mode and in incremental mode. All experiments were performed using Weka [9] on a PC with a 2.8GHz CPU and 1 GB memory.

### 4.1 Small data sets

Batch mode experiments were performed with naive Bayes, TAN, AODE, FANC, SPC, SPCr and TC using ten times repeated ten fold cross validation on 28 UCI data sets<sup>1</sup>. All datasets are discretized using three bins, but missing values were not filled in, except for TAN, which uses the most common value instead of missing values. We judged the various algorithms on classification accuracy, train time, test time and memory requirements. Due to space limitations, we cannot present the results of these experiments extensively, but give a summary of our findings.

We confirmed results from [3] and [8] that classification performance of TAN and AODE is better than naive Bayes and that AODE tends to perform similar to TAN but with lower train time. Further, of the collections (FANC, SPC, SPCr and TC) classified all consistently better than naive Bayes, except for the labor dataset, which is due to the small number of instances (just 57 in total). Better performance comes from being able to capture more dependencies, while diminished performance for very small datasets is caused by the increased variance in parameter estimation since for the collections a much larger number of parameters need to be established. Training and testing times as well as memory requirements are consistently larger than naive Bayes. Compared with TAN and AODE, FANC classification performs is about the same, SPC and TC slightly better and SPCr considerably better overall. This is illustrated in Table 1 on the right, which shows the classification performance of AODE (x-axis) versus SPCr (y-axis). Points over the x=y-line favor SPCr and below AODE. The significant outlier under the line is labor, but the most other sets show a difference in favor of SPCr. Table 1 on the left gives a summary of our findings and allows for selecting a Bayesian classifier that fits the requirements of a particular learning situation.

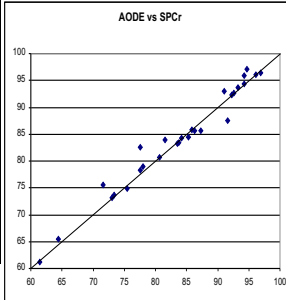
In short, though SPCr classification performance is the best overall, this comes at a cost of training time. If memory is not a restriction, TC performs best; but if it is, SPC tends to perform quite well.

---

<sup>1</sup> anneal, balance scale, breast cancer, wisconsin breast cancer, horse colic, credit-rating, german credit, pima diabetes, glass, cleveland heart-disease, hungarian heart-disease, heart-statlog, hepatitis, hypothyroid, ionosphere, iris, kr-vs-kp, labor, lymphography, primary-tumor, segment, sonar, soybean, vehicle, vote, vowel, waveform, zoo.

**Table 1.** A quick rough overview of some relevant properties of the various algorithms giving a guideline on which algorithm is appropriate for a particular situation.

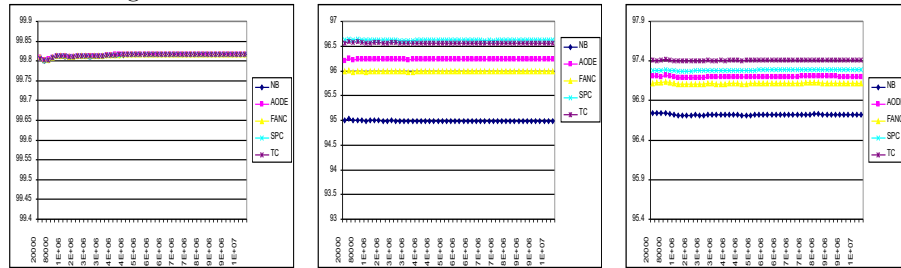
	ranking
Memory reqmnt	NB < TAN < AODE = FANC < SPC = SPCr < TC
Train time	NB < AODE = FANC < TAN < SPC < TC ≈ SPCr
Test time	NB < TAN < FANC < AODE < SPC = SPCr < TC
Accuracy	NB < TAN ≈ AODE ≈ FANC < SPC < TC < SPCr



## 4.2 Large data sets

We randomly generated models over 20 binary variables (a class and 19 binary attributes), a naive Bayes (19 edges), a TAN (37 edges) and Bayesian net augmented naive Bayes (BAN) model (50 edges) representing increasingly complex concepts. From each model a dataset of 10 million instances was generated. Figure 1 shows the learning curve for each of the models for various learning algorithms. Since the curve is rather flat fairly quickly, Figure 2 shows the curve for the first hundred thousand instances as well.

**Figure 1.** Mean accuracy of various variants on streams of synthetic data with 10 million instances. The x-axis shows the number of instances used for training, the y-axis is accuracy. Left, sourced from naive Bayes, in the middle sourced from a TAN and right sourced from a BAN.



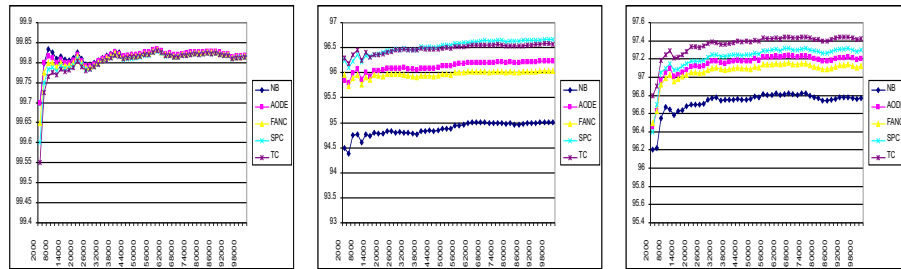
For the naive Bayes data source, all algorithms perform very similar. For the TAN data source, naive Bayes is consistently outperformed by the other algorithms and SPC and TC outperforms the other algorithms. For the BAN data source results are similar to the TAN data source, except that TC outperforms SPC. This leads to two conclusions; the collections perform well on large data sets and with increasing complexity of the concept to be learned increasingly complex collections help in performance.

We investigated sensitivity to noise by adding 10 binary attributes with no dependencies to any of the other attributes. Figure 3 shows the degradation in

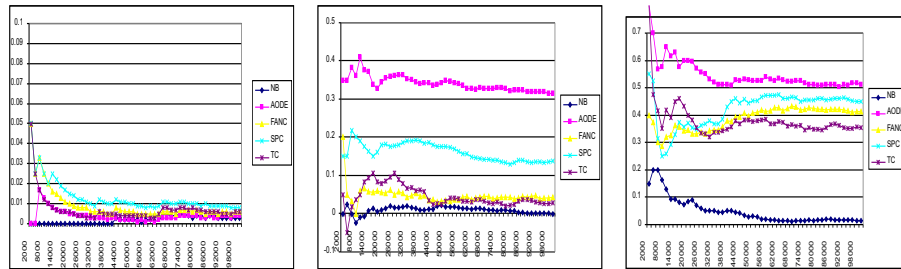


performance for the various algorithms. Naive Bayes is least affected, but the other do not degrade that much leaving naive Bayes still the worst performer. AODE is most affected while FANC, TC and SPC are in between. It appears that working with large collections of classifiers decreases sensitivity to irrelevant attributes. We also investigated the performance degradation due to adding 5% noise to the attributes and inserting 5% missing values (not shown due to space limitations) but found that all algorithms were affected similarly.

**Figure 2** Mean accuracy of various variants on streams of synthetic data. The x-axis shows the number of instances used for training, the y-axis is accuracy. Left, sourced from naive Bayes, sourced from TAN in the middle and right, sourced from BAN.



**Figure 3** Difference of accuracy when ten noisy attributes are added.



## 5 Conclusions

Voting with massive collections of Bayesian networks was shown to be feasible in polynomial time and these collections perform well both on small and large data sets. Since the training time is linear in the data set size and learning of its parameters is naturally suited to incremental updating, these collections can be applied to data streams with millions of instances. With increasing complexity of the concept to be learned increasingly complex collections were found to help in performance but cost in memory and classification time. This allows for balancing between collection performance and computational complexity.

Preliminary experimental results on FAN collections indicate that improvement of performance is possible by ordering attributes. For each attribute the order was based on the sum over all other attributes of the information gain given the class and other attribute. Also, some preliminary results of selection of a proper super parent in a super parent collection (for instance through training set performance) can improve the classification performance of a collection. However, in both these situations, it is not clear how to adapt the algorithm without sacrificing much memory or computational cost. Both these are issues we hope to address more comprehensively in the future.

Specification of alternative collection architectures and identification of their properties are interesting areas to explore further. For example, limiting the number of potential parents to the first  $k$  attributes in a FAN collection. Preliminary results showed that for the UCI data sets performance did not increase much after limiting the number of parents to the the first six attributes in the data set opening the way for computationally more efficient collection architectures.

### Acknowledgements

I thank all Waikato University Machine Learning group members for stimulating discussions on this topic, especially Richard Kirkby for making his Moa system for experimenting on large data streams available.

### References

1. C.L. Blake and C.J. Merz. UCI Repository of machine learning databases. Irvine, CA: University of California, 1998.
2. P. Domingos and G. Hulten. Mining High-Speed Data Streams. SIGKDD 71–80, 2000.
3. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. In *Machine Learning* 29:131–163, 1997
4. G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. SIGKDD 525–531, 2002.
5. G.H. John and Pat Langley. Estimating Continuous Distributions in Bayesian Classifiers. *Uncertainty in Artificial Intelligence*, 338–345, 1995.
6. E. Keogh and M. Pazzani. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. *AISTats* 225–230, 1999.
7. J.P. Sacha. New synthesis of Bayesian network classifiers and interpretation of cardiac SPECT images, Ph.D. Dissertation, University of Toledo, 1999.
8. Geoffrey I. Webb, Janice R. Boughton, Zhihai Wang. Not so naive Bayes: aggregating one-dependence estimators, *Machine Learning*, 58(1) 5–24, 2005.
9. I.H. Witten and E. Frank. *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, San Francisco, 2000.