International Journal on Artificial Intelligence Tools, Vol. 11, No. 4 (2002) 587–601
© World Scientific Publishing Company

LEARNING THE STRUCTURE OF AUGMENTED BAYESIAN CLASSIFIERS

EAMONN J. KEOGH* and MICHAEL J. PAZZANI†
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92697 USA
**eamonn@cs.ucr.edu*
†pazzani@ics.uci.edu

Received 24 May 2001
Accepted 15 January 2002

The naïve Bayes classifier is built on the assumption of conditional independence between the attributes given the class. The algorithm has been shown to be surprisingly robust to obvious violations of this condition, but it is natural to ask if it is possible to further improve the accuracy by relaxing this assumption. We examine an approach where naïve Bayes is augmented by the addition of correlation arcs between attributes. We explore two methods for finding the set of augmenting arcs, a greedy hill-climbing search, and a novel, more computationally efficient algorithm that we call SuperParent. We compare these methods to TAN; a state-of the-art distribution-based approach to finding the augmenting arcs.

Keywords: Machine learning; Bayesian learning; Heuristic search.

1. Introduction

The Bayesian classifier is a simple classification method, which classifies an instance j by determining the probability of it belonging to class C_i.[1] These probabilities are calculated as:

$$P(C_i \mid A_1 = V_{1_j} \,\&\cdots\&\, A_N = V_{N_j}), \tag{1}$$

where an example is represented as attribute-value pairs of the form $A_i = V_i$. If there are N independent attributes, then the probability is proportional to:

$$P(C_i)\prod_k P(A_k = \mathrm{V}_{\mathrm{k_j}} \mid C_i) \tag{2}$$

When this independence assumption is made, the classifier is called naïve (Simple, Idiots) Bayes.[2] Naïve Bayes has been shown to be competitive with more complex, state-of-the-art classifiers.[3,4,5] This is surprising given the explicit assumption that all attributes are independent given the class. This assumption rarely holds in real world problems. There have been recent attempts to explain its surprisingly good performance and to improve performance by relaxing the independence assumptions.[6,7,8,9]

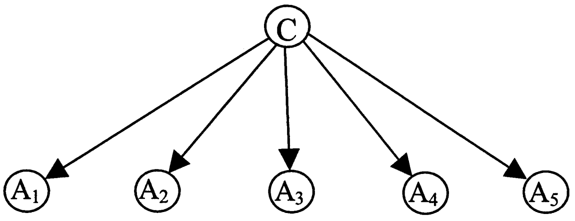

Fig. 1. An example of a Naïve Bayes Network.

The work of Friedman and Goldszmidt is particularly interesting.[8] They compared naïve Bayes to Bayesian networks, a much more powerful and flexible representation of probabilistic dependence.[10] Surprisingly, using unrestricted Bayesian networks did not generally lead to improvements in accuracy and even reduced accuracy in some domains. This prompted them to propose a compromise representation, combining some of Bayesian networks ability to represent dependence, with the simplicity of naïve Bayes. This representation is called *augmented naïve Bayes*, and is defined by the following conditions:

- Each attribute has the class attribute as a parent.
- Attributes may have one other attribute as a parent.

The latter condition means that if there is an arc from A_i to A_j, the two attributes are not independent given the class. Instead the influence of A_j on the class probabilities depends on the value of A_i. Figure 2 shows an example of an augmented Bayes network.

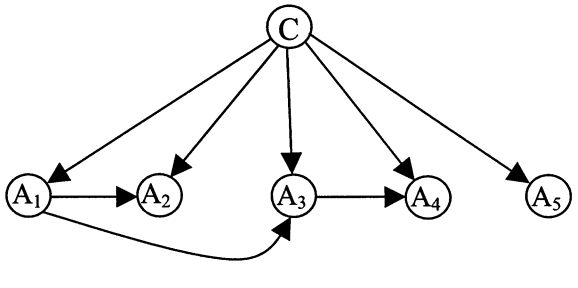

Fig. 2. An example of an Augmented Bayes Network.

Finding the best augmented Bayes network (in the sense of best approximation of an unrestricted Bayes network) is an NP-hard problem. Friedman and Goldszmidt deal with this difficulty by restricting the network to be a tree topology (in addition to the arcs from the parent to each node), and utilizing a result by Chow and Liu to efficiently find the best tree-augmented naïve Bayes (TAN).[11]

In this paper we take a different approach to constructing augmented Bayesian networks. Rather than directly attempting to approximate the underlying probability distribution, we concentrate solely on using the same representation to improve classification accuracy. We show two methods for constructing augmented Bayesian networks, a hill climbing greedy search, and a novel, more efficient search algorithm, which we call SuperParent. We compare these to Friedman and Goldszmidt's approach and show that the best approximation of the underlying probability distribution does not necessarily result in the most accurate classifier.

2. Searching for Augmented Bayes Networks

Following Friedman and Goldszmidt we restrict the allowed topology of an augmented Bayes network in the following manner. Each node may have at most one (non-class) parent. There are two reasons for this restriction. First, it reduces the search space. Second, the probability estimates for a node become more unreliable as additional parents are added, because the size of the conditional probability tables increases exponentially with the number of parents. Restricting the number of parents to two mitigates problems in estimating probabilities from data while allowing some amount of dependence among variables to be represented.

This restriction limits the maximum number of correlation arcs added to $N - 1$, where N is the number of attributes. The minimum number of correlation arcs added is zero, which corresponds to a naïve Bayes classifier, a special case of an augmented Bayes network.

We present the following definition to facilitate discussion of augmented Bayes networks.

Definition 1 (Orphan)
In augmented Bayes network, a node without a parent, other than the class node, is called an *orphan*.

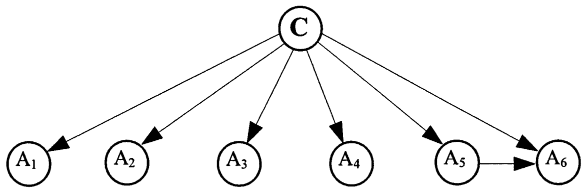

Fig. 3. An augmented Bayes network **B**, all non-class nodes except A6 are *orphans*.

Table 1 shows an outline of the algorithm for building an augmented Bayes network using hill climbing search (HCS). The network is initialized to naïve Bayes, and the set of orphans, O, is initialized to the full set of nodes $A_1, A_2, \ldots, A_N$. Each possible arc from A_i to A_j ($A_i \neq A_j$, $A_j \in O$) is evaluated, using leave-one-out cross validation to estimate the accuracy of the network with that arc added. If no arc produces an improvement in classification accuracy, the current classifier is returned. Otherwise, the arc that gave the most improvement is retained, and the node pointed to by the arc is removed from *O*. This process is repeated until *O* contains just one node, or there are no arcs that can be added to improve classification accuracy. Note that we must stop adding arcs when there is just one orphan, otherwise we will introduce a cycle into the network.

Table 1: An outline of hill climbing search (HCS)

```
0. Initialize network to naive Bayes.
1. Evaluate the current classifier.
2. Consider adding every legal arc to the current classifier.
3. If there is an arc addition, which improves accuracy, then
   add the arc which improves accuracy the most to the current
   network, and go to 2.
    Else: Return current classifier.
```

$O(N^2)$ classifiers are constructed and evaluated for each arc added to the network, and $O(N)$ arcs may be added. So the complexity for HCS is $O(N^3)$. In Section 2.1 we discuss two optimizations that greatly speed up the search process without affecting the returned network.

2.1. *Efficient evaluation of changes to classifiers*

To speed up the process of evaluating many classifiers we have implemented the following two optimizations, the first of which was introduced by Pazzani.[7]

We order the training instances, so that examples misclassified by the previous classifier are tested first. This allows the algorithm to abandon testing of a classifier as soon as the number of misclassified examples is greater than the current *best-so-far* classifier. This technique allows the algorithm to perform a fraction of the tests required by full search, but return the same result. The fraction of examples actually tested on a given dataset is approximately equal to the error rate of naïve Bayes on that dataset.

To prevent the need for completely recomputing equation 1 for each classifier built, we do the following. In the first step of the algorithm, we store the results of equation 2 in a J by I matrix, (J is the number of instances in the training set, I is the number of distinct classes) where each element is the probability that example *j* belongs to class C_i. When testing a new classifier which has an addition arc from node A_b to node A_a, we adjust the matrix by multiplying the element (i, j) by

$$\frac{P(A_a = V_{a_j} | C_i \,\&\, A_b = V_{b_j})}{P(A_a = V_{a_j} | C_i)} \tag{3}$$

This equation simultaneously factors out the effect of the "orphan" node A_a, and factors in the effect of the arc from node A_b to node A_a. Therefore, the time required to evaluate one change to a network is independent of the number of attributes. So the speed up achieved by this optimization is approximately N, the number of attributes.

To test the utility of these two optimizations we performed the following experiment. We ran the optimized and unoptimized versions of the HCS algorithm, 10 times, on several datasets from the UCI repository (Details about the datasets can be found in tables 3 and 6).[13] Table 2 shows the average speedup achieved.

Table 2: The average speedup achieved on various datasets by the two optimizations described in section 2.1

DataSet	Speedup achieved
Iris	41.0
Soybean-Large	312.3
Segment	199.2
Vote	135.7

As explained above, the speedup achieved depends on both the number of attributes and the accuracy achievable by naïve Bayes. Nevertheless the optimizations are clearly useful in general.

2.2. *Experimental results*

Our experimental methodology is closely modeled on that of Friedman and Goldszmidt.[8] We tested 13 data sets from the UCI repository and one artificial data set.[13] The accuracy of each learning method on each domain was determined by running 5-fold cross validation 25 times for each data set. All classification algorithms were trained and tested on exactly the same cross validation folds. Following Friedman and Goldszmidt, instances with missing values were deleted from the database and continuous values were discretized, using only the training data, by Fayyad & Irani's entropy based method.[14] In our comparison, we include both the smoothed and unsmoothed versions of TAN for completeness. In the smoothed version we used $N^0 = 5$ (see Friedman & Goldszmidt for details).[8] In the unsmoothed TAN and in our work, we replace zero probabilities with a small epsilon (.0001). The four algorithms used in this experiment are: **Naive** - naïve Bayes; **TAN** - Friedman and Goldszmidt's unsmoothed Tan network; **TAN^S** - Friedman and Goldszmidt's smoothed Tan network; **HCS** - Augmented Bayes networks built using hill climbing greedy search.

As an extreme example of attribute dependence we create an artificial noisy two-class dataset called *exclusive-or*. This dataset has 500 instances, and 10 attributes. After the introduction of class noise, the exclusive-or of two of the attributes predicts the class with 70% accuracy. The other 8 attributes are completely irrelevant. Table 3 summarizes the datasets.

Table 3: Descriptions of domains used.

Dataset	# Attributes	# Classes	# Instances
Vehicle	18	4	846
Post-op	9	3	90
Lung	56	3	32
Australia	14	2	690
Hepatitis	19	2	270
Vote	16	2	435
Heart	13	2	270
Soybean-Large	35	19	562
Pima	8	2	768
Breast	10	2	683
Iris	4	3	150
Segment	19	7	1540
E.coli	7	8	336
exclusive-or	10	2	500

Table 6 in the Appendix summarizes the accuracy of each algorithm on each dataset. The best accuracy achieved on each dataset is shown in bold. Runners up, which did not differ at the 5% confidence level using a paired two tailed t-test are also shown in bold. We confirmed Friedman and Goldszmidt's ranking of Naïve, TAN and TAN^S classifiers. Using a binomial sign test HCS is significantly better than the other algorithms on this collection of datasets at the .001 level. Figure 4 provides a visual comparison of HCS and TAN. It shows that the HCS approach is usually more accurate than the TAN approach.

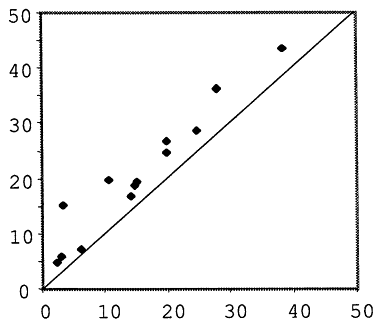

Fig. 4. Scatter plot comparing the error of HCS to TAN on 13 datasets from the UCI repository. Points above the diagonal line correspond to datasets where HCS performs better. Points on the diagonal line indicate no difference in performance.

The data set *exclusive-or* is an excellent example of how augmenting arcs allow a Bayesian classifier to learn a non-linearly separable function. Naïve Bayes performs at chance levels on this data set. TAN and TAN^S do better, each is sometimes able to find the correct arc to connect the two dependent attributes. However, even if one considers just the folds where TAN and TAN^S find the correct arc, they still do slightly worse than HCS. This is because both add 8 additional unnecessary arcs. These arcs are fitted to noise, and do not generalize to the test data. In contrast HCS always finds the correct arc to add in the first iteration, and then it usually halts (occasionally it adds a second arc due to a chance pattern occurring in a cross-validated fold). This allows HCS to achieve the maximum accuracy possible on the data set. This increased representation flexibility of HCS has an additional bonus. If we see an arc in a network returned by HCS it indicates that modeling the relationship between these attributes is important to increase predictive accuracy of the model. Viewing an arc in a network returned by TAN, we have no such assurance, because TAN always returns N - 1 augmenting arcs, even when the attributes are completely independent. This makes our approach more useful when comprehensibility and insight is important. In Section 4, we further explore a comparison of augmenting the naïve Bayesian classifier modeling the probability distribution and our approach.

3. SuperParent

We have demonstrated that the addition of arcs using full greedy search can mitigate the strong independence assumptions of naïve Bayes and improve its classification accuracy. Here, we introduce SuperParent (SP) a more efficient heuristic search that attempts to have the same improvement in accuracy with a less expensive search technique. The general idea is that in hill climbing search as described so far, we attempt to find the best arc to add. Here, we break this up into two steps, first finding a good parent and then finding the best child of that parent.

Definition 2 (SuperParent)

Given an augmented Bayes network, if we extend arcs from node A_x to every orphan, node A_x is said to be a *SuperParent.*

Definition 3 (FavoriteChild)

Given an augmented Bayes network, if we extend an arc from node A_x to each orphan in turn, and test the effect on predictive accuracy, the node pointed to by the best arc is said to be the *FavoriteChild* of A_x.

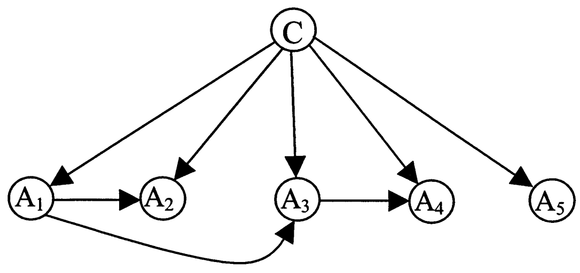

Fig. 5. The augmented Bayes network **B2** = *SuperParent(*A3,**B***)*, note that node A3 has been made parent of all *orphans*.

Table 4 shows an outline of the SuperParent algorithm. The network is initialized to naïve Bayes, and the list of orphans, O, is initialized to the full set of nodes $A_1, A_2, \ldots, A_N$. The effect on classification accuracy, of making each node a SuperParent is assessed. The best such node we designated A_{SP}. Next, the algorithm finds FavoriteChild of A_{SP}, by assessing the effect of adding a single arc from A_{SP} to each orphan. If the addition of the arc from A_{SP} to the FavoriteChild improves the classification accuracy and $|O| > 1$, then the FavoriteChild is removed from O, the arc is added to the current classifier, and the SuperParent cycle begins again. If there was no improvement, or $|O| = 1$, the current classifier is returned.

Table 4: An outline of the SuperParent algorithm (SP).

```
0. Initialize network to naive Bayes.
1. Evaluate the current classifier.
2. Consider making each node a SuperParent. Let $A_{SP}$ be the
    SuperParent which increases accuracy the most.
3. Consider an arc from $A_{SP}$ to each orphan. If the best such arc
    improves accuracy, keep it and go to 2.
4. Else: Return the current classifier.
```

We defer a detailed discussion of our experimental results until section 3.2. However, Figure 6 demonstrates that SuperParent has essentially the same accuracy as HCS, even though it utilizes a more efficient search.

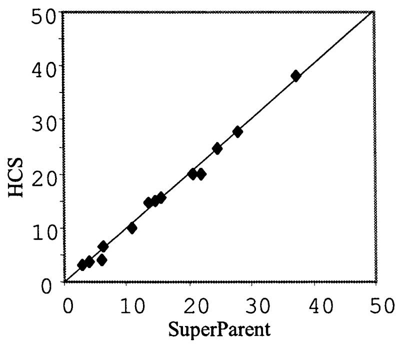

Fig. 6. Scatter plot comparing SuperParent to HCS on 13 datasets from the UCI repository. Points on the diagonal line indicate no difference in performance.

O(N) classifiers are constructed and evaluated for each arc added to the network, and O(N) arcs may be added. So the complexity for SP is O(N^2). We also utilize the two optimizations mentioned for HCS.

3.1. *How SuperParent works*

If an attribute is truly independent of all other attributes, then the following equality is true.

$$P(C_i)\prod_k P(A_k = \mathrm{V}_{\mathrm{k_j}} | C_i) = P(C_i)\prod_k P(A_k = \mathrm{V}_{\mathrm{k_j}} | C_i \,\&\, A_{SP}) \tag{4}$$

Therefore making that attribute the SuperParent should not affect the classification accuracy. Because we are estimating the probabilities with a finite amount of data, we can expect small fluctuations in classification accuracy. If however, the right-hand side of equation 4 differs from the left-hand side by a substantial amount, we can infer that A_{SP} is related to at least one other attribute. By making each attribute a SuperParent in turn, we can detect which attributes make good parents of other (as yet, unknown) attributes. This is what we do in line 2 of Table 4.

Choosing the attribute that, as SuperParent, most increases the classification accuracy, ensures that we have an attribute that is strongly correlated with some other attribute, or attributes. We still need to find the attribute to which it is most strongly correlated. This we do in line 3 of Table 4.

Simply stated, the SuperParent algorithm is a heuristic, which repeatedly finds a good parent node, then that node's best child, until there is no more improvement to be had. The heuristic is not admissible. That is to say it is not guaranteed to produce the same set of augmenting arcs as HCS.

3.2. *Experimental results with SuperParent*

Table 6 in the Appendix also lists the results of running SuperParent under the same conditions as Table 1 (in fact, SuperParent was tested on exactly the same folds). Only on the dataset Segment is HCS significantly more accurate than SuperParent. These results are also summarized in Figure 6. It shows that there is little difference between the two algorithms. This indicates that searching for a good parent in the manner that SuperParent does is a useful heuristic for finding arcs to add.

Table 6 also shows the average time taken by the TAN and SuperParent algorithms to build a classifier. The experiments were run on a Pentium 120 MHz. All algorithms were coded in Matlab™, which is interpreted. An implementation in a complied language like C would run 2 to 3 orders of magnitude faster. Note that the ratios of the time taken by each approach over different datasets are not constant. The time taken by TAN depends only on the size of the dataset, whereas the time taken by SuperParent depends on the accuracy achievable with naïve Bayes, and the amount of improvement possible. In general, SuperParent is approximately ten times slower than TAN but is clearly fast enough to use in practice on the same sorts of problems that TAN has been used. It might be suggested that TAN could use this time difference to somehow search over variations of itself, and match SuperParents' accuracy. In Section 4 we will demonstrate some difficulties with such extensions to TAN.

3.3. *Anytime implementation of SuperParent*

We have shown that a heuristic search is capable of producing augmented Bayes classifiers, which are superior to TAN networks and Naïve Bayes. The price paid for higher classification accuracy is computational time. The two optimizations mentioned in section 2.1 greatly help, but nevertheless SuperParent may be too slow for interactive exploration of high dimensional datasets with many instances. One possible solution is to parallelize the algorithm. This should be trivial to implement, because each processor can be assigned a subset of the search space. Another possibility is to implement the SuperParent as an anytime algorithm.

Anytime algorithms are algorithms that trade execution time of quality of results.[15] Because of the computational complexity of evaluating and building Bayesian networks, there has been much previous work on using anytime algorithms in a Bayesian framework.[16,17]

The SuperParent algorithm is ideally suited to anytime implementation for the following reasons.

1) The accuracy of current *best-so-far* classifier is always at least as good as naïve Bayes.
2) The largest gains in accuracy tend to occur in the early stages of the algorithm's execution.
3) If we interrupt the algorithm before it has exhausted the search space, we can use the *best-so-far* classifier to classify an instance, then continue searching from the point at which it was interrupted.
4) The accuracy of the *best-so-far* classifier provides a measure of quality that can be used for meta reasoning about resources. For example, we can decide to interrupt the algorithm if the accuracy ever climbs above some threshold, or if the percentage change in accuracy during the previous iteration was below some minimum threshold.

We have implemented SuperParent in the following manner. If the algorithm is interrupted in the FavoriteChild cycle (Step 3 of Table 4) we return the *best-so-far* classifier, including the best arc found during the current (incomplete) cycle. If the algorithm is interrupted during the SuperParent cycle (Step 2 of Table 4), we delete the (temporary) SuperParent arcs and return the *best-so-far* classifier from the previous iteration.

We ran the following experiment on Anytime SuperParent. We ran the normal version of SuperParent on the Pima, Heart, Vote and Australia datasets, using all the data as training data. This step was performed simply so that we could measure how long the standard algorithm took. We then ran Anytime SuperParent on the same datasets, using a randomly chosen 80 percent of the data. We interrupted the algorithm at 5, 10, 20, 40, 60 and 80 percent of the time taken for a full search by the standard algorithm, and evaluated the returned classifier with the remaining 20 percent of the data. We repeated this experiment 24 times for each dataset, and averaged the results. Figure 7 shows the performance profile for each dataset.

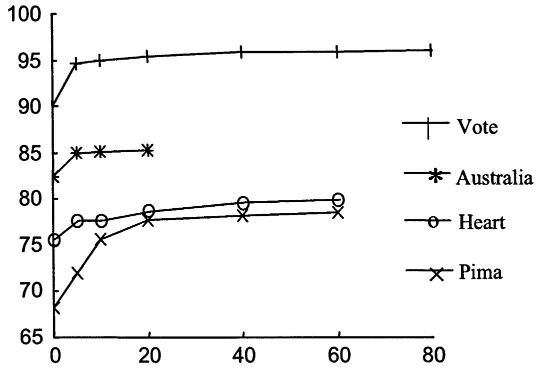

Fig. 7. The performance profile of Anytime SuperParent. The y-axis represents classification accuracy. The x-axis represents the time taken as a percentage of the time taken by the standard SuperParent algorithm.

4. Why is a Classification Approach Better?

There are three differences between Friedman and Goldszmidt's approach and the search approaches we proposed in the previous sections. First, they use a different criteria for deciding which arcs to add to the naïve Bayesian classifier. Second, Friedman and Goldszmidt's approach always adds N-1 arcs while our approach stops adding arcs when no improvement will occur.[8] Third, the direction of an arc is chosen randomly in a TAN classifier, whereas our approach differentiates between an arc and its reversal. In order to gain an understanding of why our algorithm yields more accurate classifiers than the earlier approach, we consider modifications to each method.

1. We modify our search algorithm so that it always adds N-1 arcs (SP_{full}). This is done by always adding the arc that results in the most accurate classifier, but not requiring that an arc improve the accuracy when compared to not adding an arc. Comparing this modified algorithm to SP and TAN will allow us to determine if SP's superior performance is due to its ability to add fewer arcs.

2. We create a variant of Friedman and Goldszmidt's approach that does not necessarily add every arc in the tree to the Bayesian classifier (sTAN). In particular, once the tree is formed, we use a hill climbing search procedure to add arcs to the Bayesian classifier. We select from only those edges that appear in the tree and we consider both possible arc directions for all edges. Like our original hill-climbing search procedure, this stops when no arc addition results in a more accurate classifier. Because this variant of TAN is free from limitations with regard to the number of arcs

that can be added, and can choose the direction of the arcs, we can use it to ascertain how whether the arcs selected by TAN improve accuracy when compared to SP.

3. We create a variant of SuperParent called SP_{ran}. Here we build a normal SP classifier, then reverse the direction of each arc with a probability of .5. We do not reverse an arc where this would violate the rule that each node may have at most one non-class parent. Comparing this modification of SP to the standard SP algorithm will help determine the importance of arc direction in augmented Bayesian classifiers.

We ran an experiment on ten datasets using the three above algorithms together with smoothed TAN and the normal SuperParent algorithm. We used the same methodology as mentioned in section 2.2, in particular, all classification algorithms were trained and tested on exactly the same cross validation folds. Table 5 summarizes the results.

Table 5: Experimental results of comparing various algorithms.

Dataset	**Tan^S**	**sTAN**	**SP_{ran}**	**SP_{full}**	**SP**
Vote	94.6	**96.0**	95.2	**96.0**	**96.3**
Australia	84.2	**85.2**	83.4	83.1	**85.2**
Pima	77.6	77.7	76.9	**78.8**	**78.5**
Breast	96.7	**96.9**	96.7	**97.0**	**97.5**
Heart	76.2	76.7	76.6	77.4	**78.7**
Hepatitis	83.8	83.5	84.3	**89.5**	85.8
Vehicle	66.0	67.8	**69.6**	**70.3**	**69.9**
Soybean	86.6	85.1	86.4	86.9	**88.9**
E.coli	81.5	82.2	83.6	83.3	**84.6**
Post-op	72.9	71.3	70.8	**75.5**	**75.3**
Mean	*82.01*	*82.24*	*82.35*	*83.78*	*84.07*

The last row contains the mean accuracy for each column, although we echo the often-stated caution of its debatable significance. The best accuracy for a given dataset is reported in bold text. Where a runner-up does not differ at the 5% confidence level (using a paired two-tailed t-test), it too is recorded in bold.

SP is clearly the superior classifier, only on the dataset Hepatitis is it relegated to a significant second place. SP_{full} does not do quite as well, but it is still better than TAN^S on 9 of the datasets. The difference between its mean accuracy and SP's is only 14% of the difference between TAN^S and SP. We take this as strong evidence that the (possible) disparity in the *number* of arcs added to the augmented Bayes classifier by the different approaches is not the main reason for SP's superior performance.

SP_{ran} does significantly worse than SP on all but one dataset. Its overall performance is only marginally better than TAN^S. This supports the hypothesis that the *direction* of the arcs added has a significant effect on the quality of the classifier created. With this in mind, is it possible to improve the quality of TAN^S by searching over possible arc reversals? The evidence is less clear here. sTAN performs better that TAN^S on 7 of the datasets, significantly so on three datasets. However some of this improvement may be due to the fact that sTAN is able to add less than N-1 arcs. In any case, the improvement of sTAN over TAN^S is only a fraction of the difference between SP and TAN^S.

Given these results, we surmise that the overall difference in performance between TAN^S and SP is due to the fact that SP generally chooses a different set of augmenting arcs to add, and that SP takes into account the direction of the augmenting arcs.

5. Related work

There has been much recent work on extensions to the naïve Bayes classifier. Kohavi has shown that although irrelevant features should theoretically not hurt the accuracy of naïve Bayes, in practice irrelevant features do degrade performance.[4] He deals with this problem using wrappers for subset selection. Langley and Sage use a similar approach.[18] Although neither approach directly deals with the problem of related attributes they can have the effect of mitigating the problem. If two attributes are related, then naïve Bayes will overweight the evidence from the two attributes. Deleting one of the two attributes may help.

Both Kononenko and Pazzani deal with attribute dependence by merging two (or more) related attributes into a new compound attribute, which replaces the original attributes in the classifier.[9,7] While Kononenko uses a statistical test to decide which attributes to merge,[9] Pazzani achieves much better results by using predictive accuracy as a merging criteria.[7]

6. Conclusions

In this paper, we have shown that it is possible to build classifiers that are superior to naïve Bayes and a previous approach to create augmented Bayesian networks by searching for violations of the independence assumption that reduce the accuracy of the naïve Bayesian classifier. We further demonstrate two useful optimizations that greatly speed up the process of finding correlated attributes. Finally, we showed that our algorithm is ideally suited to implementation as an anytime algorithm permitting efficient interactive exploration of the space of augmented Bayesian classifiers and producing more accurate results with more computation.

Acknowledgments

This research was supported by NSF grant IRI-9310413, AFOSR grant F49620-96-1-0224, and Sun Microsystems. The first author would like to thank Peggy Keogh who inspired the term “SuperParent”.

References

[1] Duda, R. & Hart, P. (1973). *Pattern classification and scene analysis*. New York: John Wiley & Sons.

[2] Ripley, B. (1996). Pattern recognition and neural networks. Cambridge University Press.

[3] Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Proceedings of the 12th International Conference on Machine Learning*. (pp. 194-202). Tahoe City, Ca: Morgan Kaufmann.

[4] Kohavi, R. (1994). Feature subset selection as search with probabilistic estimates. In *AAAI Fall Symposium on Relevance*.

[5] Sahami, M. (1996). Learning limited dependence Bayesian classifiers. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (pp. 335-338). Portland OR: AAAI Press.

[6] Domingos, P. & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29, (pp. 103-130).

[7] Pazzani, M. (1996). Searching for dependencies in Bayesian classifiers. In D. Fisher & H. J. Lenz (Eds.), *Learning from data: Artificial intelligence and statistics* V (pp. 239-248). New York, NY: Springer-Verlag.

[8] Friedman, N. & Goldszmidt, M. (1996). Building classifiers using Bayesian networks. In *Proceedings National Conference on Artificial Intelligence*, (pp. 1277-1284). Menlo Park, CA: AAAI Press.

[9] Kononenko, I. (1991). Semi-naïve Bayesian classifier. In *Proceedings of the 6th European Working Session on Learning*, 206-219. Berlin: Springer- Verlag.

[10] Pearl, J. *1988 Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

[11] Chow, C. & Lui, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Trans. On Info Theory* 14 (pp. 462-467).

[12] Pazzani, M. (1996). Constructive induction of Cartesian product attributes. *Information, Statistics and Induction in Science*. Melbourne, Australia.

[13] Merz, C., Murphy, P. & Aha, D. (1997). UCI repository of machine learning databases. Dept of Information and Computer Science, University of California, Irvine. http://www.ics.uci.edu/mlearn/MLRepository.html.

[14] Fayyad, U. & Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, (pp. 1022-1027). Morgan Kaufmann.

[15] Grass, J. & Zilberstein, S. (1996). Anytime algorithm development tools. *Sigart Artificial Intelligence*. Vol 7, No. 2, April. ACM Press.

[16] Horvitz, E. & Klein, A. (1995). Reasoning, Metareasoning, and Mathematical Truth: Studies of theorem proving under limited resources. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence.*

[17] Wellman, M. & Lui, C. (1994). State-space abstraction for anytime evaluation of probabilistic networks. In *Proceedings of the 10th Conference on Uncertainty in AI*, Seattle WA.

[18] Langley, P. & Sage, S. (1994). Induction of selective Bayesian classifiers. In *Proceedings of the Tenth Conference on Uncertainly in Artificial Intelligence.* (pp 399-406).

Appendix

Table 6: Experimental results of comparing various algorithms. The best result and those not significantly worse than the best at the 5% confidence level are shown in bold. The last two columns contain the average time (in seconds) taken to build a classifier using TAN and SP.

Dataset	Naïve	TAN	TAN^S	HCS	SP	TAN Time	SP Time
Vehicle	63.29 ± 1.58	64.17 ± 2.47	68.38 ± 2.39	**71.74** ± 1.87	**71.74** ± 2.01	333	1390
Post-op	72.01 ± 0.83	71.96 ± 1.53	72.78 ± 1.73	**74.88** ± 1.52	**75.10** ± 2.11	4	41
Lung	51.33 ± 7.34	56.86 ± 8.99	58.38 ± 7.86	**62.01** ± 8.18	**61.28** ± 6.58	156	767
Australia	82.32 ± 0.68	81.64 ± 0.66	84.14 ± 0.42	**84.74** ± 0.75	**85.20** ± 0.55	168	1299
Hepatitis	83.75 ± 1.37	83.75 ± 2.43	83.50 ± 2.00	**85.25** ± 1.87	**86.25** ± 2.13	9	126
Vote	90.26 ± 0.78	94.83 ± 1.48	93.76 ± 0.56	**96.43** ± 0.56	**96.01** ± 0.43	17	84
Heart	74.85 ± 3.30	75.85 ± 2.84	**77.95** ± 2.41	**79.59** ± 2.16	**77.81** ± 1.96	6	93
Soybean-Large	85.03 ± 1.19	80.88 ± 1.72	86.32 ± 1.58	**89.97** ± 1.22	**89.02** ± 1.71	1046	13807
Pima	69.23 ± 1.35	77.21 ± 1.75	77.78 ± 1.56	**78.38** ± 1.31	**79.00** ± 1.28	4	63
Breast	96.74 ± 0.45	95.67 ± 0.72	96.74 ± 1.02	**97.01** ± 0.89	**97.11** ± 0.81	21	172
Iris	**93.33** ± 1.00	**93.60** ± 0.95	**94.00** ± 1.25	**93.93** ± 1.35	**93.60** ± 1.25	3	10
Segment	91.17 ± 1.86	85.32 ± 1.65	**95.61** ± 1.23	**96.30** ± 1.07	94.01 ± 1.36	5491	62410
E.coli	77.33 ± 0.44	81.09 ± 0.69	82.11 ± 0.54	**84.30** ± 0.75	**84.36** ± 0.34	16	91
exclusive-or	53.62 ± 2.30	57.05 ± 2.16	59.34 ± 2.74	**70.12** ± 1.46	**70.32** ± 1.43	12	96