Applying General Bayesian Techniques to Improve TAN Induction

Jesús Cerquides
Ubilab
UBS AG
Bahnhofstrasse 45
P.O. Box, CH-8098 Zürich
Jesus.Cerquides@ubs.com

Abstract

Tree Augmented Naive Bayes (TAN) has shown to be competitive with state-of-the-art machine learning algorithms [9]. However, the TAN induction algorithm that appears in [9] can be improved in several ways. In this paper we identify three weak points in it and introduce two ideas to overcome those problems: the *multinomial sampling approach* to learning bayesian networks and *local bayesian model averaging*. These ideas are generic and can thus be reused to improve other learning algorithms. We empirically test the new algorithms, and conclude that in many cases they lead to an improvement in accuracy in the classification and in the quality of the probabilities given as predictions.

Keywords:Tree Augmented Naive Bayes, Bayesian Model Averaging, Multinomial Sampling

1 Introduction

Tree Augmented Naive Bayes (TAN) has shown to be competitive with state-of-the-art machine learning algorithms. In this paper we analyze the TAN induction method proposed in [9]. While performing well in practice, we have identified three weak points where corrections can lead to a more coherent and accurate classifier:

1. The development of the algorithm relies in the decomposition of the log likelihood according to the structure of the network. This decomposition provides us the maximum likelihood parameter setting. In [9] they notice experimentally that ad hoc softening performs better. This is not understandable from a theoretical point of view.

2. The algorithm selects a single model, ignoring model uncertainty.

3. The algorithm tries to find the TAN that maximizes likelihood while in order to have better classification accuracy, conditional likelihood should be maximized.

In the following, we give solutions that try to overcome those problems. We do that by providing a coherent theoretical framework for TAN induction. We start by introducing bayesian networks and proposing a new approach (the *multinomial sampling approach*) to the problem of learning bayesian networks in Section 2. In Section 3 we discuss TAN into detail and use the *multinomial sampling approach* to derive an algorithm for learning maximum likelihood TAN's that uses a unique coherent probability distribution at every step and competes in accuracy with the one proposed in [9]. In order to deal with problems two and three, we consider bayesian model averaging (BMA) in Section 4, introduce *local bayesian model averaging* (LBMA) in 4.2 and we make the application of LBMA to TAN induction in 4.3. We empirically evaluate the results of our improvements in Section 5, to finish up pointing some conclusions and research prospects in Section 6

2 Learning Bayesian Networks

Let $\mathbf{U} = \{X_1, \ldots, X_n\}$ be a set of discrete random variables. A bayesian network is an annotated directed acyclic graph that encodes a joint probability distribution over $\mathbf{U}$. Formally it is a pair $B = \langle G, \Theta \rangle$. G is a DAG whose vertices correspond to the random variables and whose edges represent direct dependencies between the variables. Θ represents the set of parameters that quantify the network. It contains a parameter $\theta_{x_i|\Pi_{x_i}} = P_B(x_i|\Pi_{x_i})$ for each possible value x_i of X_i and Π_{x_i} of Π_{X_i}, where Π_{X_i} denotes the set of parents of X_i in G. A bayesian network B defines a unique joint probability distribution over $\mathbf{U}$ given by

$$P_B(X_1, \ldots, X_n) = \prod_{i=1}^{n} P_B(X_i|\Pi_{X_i}) = \prod_{i=1}^{n} \theta_{X_i|\Pi_{X_i}} \tag{1}$$

The problem of learning a bayesian network has been informally stated as:

Statement 1 (Friedman, Geiger & Goldszmidt, 1997) *Given a training set* $D = \{u_1, \ldots, u_N\}$ *of instances of* $\mathbf{U}$ *find the network* B *that best matches* D.

This statement is possibly a good objective in the case you are trying to describe what is in the data, but for classification purposes, the fact of trying to match the data perfectly usually causes overfitting. That is why, in order to use bayesian networks for classification purposes we prefer the following informal statement:

Statement 2 *Given a sample* $S = \{u_1, \ldots, u_N\}$ *of a probability distribution* P^* *find the network* B *that best matches* P^*.

In fact, we do think that even for descriptive purposes the last statement can turn out very useful.
In order to continue the development we need to make some simple definitions. Let $\#A$ stand for the cardinality of set A, $Count_D(X)$ stand for the number of observations in our sample that fulfill condition X and $Freq_D(X) = \frac{Count_D(X)}{N}$. $Val(X_i)$ is the set of possible states of the random variable X_i and $\#Val(X_i)$ the number of different possible states X_i can be in.
One of the measures used to learn bayesian networks is the *log likelihood*:

$$LL(B|D) = \sum_{i=1}^{N} \log(P_B(u_i)) \tag{2}$$

This measure has the property that can be decomposed according to the structure of B giving:

$$LL(B|D) = N \sum_{i=1}^{n} \sum_{\substack{x_i \in Val(X_i) \\ \Pi_{x_i} \in Val(\Pi_{X_i})}} Freq_D(x_i, \Pi_{x_i}) \log(\theta_{x_i|\Pi_{x_i}}) \tag{3}$$

If $Freq_D(x_i, \Pi_{x_i})$ is strictly positive and defined everywhere (i.e. we have at least one observation for each possible pair x_i, Π_{x_i} in our dataset) it is easy to see that $LL(B|D)$ is maximized when

$$\theta_{x_i|\Pi_{x_i}} = \frac{Freq_D(x_i, \Pi_{x_i})}{Freq_D(\Pi_{x_i})} \tag{4}$$

This results allow us to search separately the space of network structures and the space of network parameters.

2.1 The multinomial sampling approach to learning bayesian networks

Here we extend Statement 2 providing it with a more concrete semantic. The dataset D of Statement 1 can be seen as a sample of a probability distribution P^*. We can assume that P^* is a multinomial distribution with a number of possible states equal to:

$States(P^*) = \prod_{i=1}^{n} \#Val(X_i)$. Each instance in the dataset is equivalent to the observation of a concrete state as an outcome of a multinomial trial. In order to learn a bayesian network we should take two steps:

1. Approximate the distribution P^* from the information we have in the sample S by using multinomial sampling methods. This generates P^*_S.

2. Find the bayesian network which better fits P^*_S.

This means a change in the epistemic approach to bayesian networks. In the commonly used approach, we try to find the network structure that is more likely to have generated the data. We really believe that the data is generated by a bayesian network, and try to find by which one. In this new approach we assume that the data is generated by a huge multinomial probability distribution P^*, far more complicated that what we can understand. In the first instance we calculate P^*_S which should be the best approximation to P^* given the data at hand. Then we try to find a simple bayesian network that fits P^*_S best. This network will give us a more understandable view of P^*_S and will allow us to predict for unseen examples.
We need a way to calculate P^*_S. There is a lot of literature dedicated to the problem of multinomial sampling. A good reference for this is [8]. For the purposes of this paper we adhere to the *principle of indifference*, which says that if we lack better information, we should assign an equal probability to each possible success. Our prior is then a Dirichlet distribution with $States(P^*)$ equiprobable possible states. We still have to fix one more parameter, the relevance we are giving to the prior, namely λ. There is no a theoretically best way to fix λ. The question of the relevance we should give to the prior is still open. Once the prior is fixed, we can give an expression for P^*_S (i.e. the posterior probability after having seen the sample S).

$$P^*_S(x_1, \ldots, x_n) = \frac{Count_S(x_1, \ldots, x_n) + \frac{\lambda}{States(P^*)}}{N + \lambda} \tag{5}$$

For readers familiar with the work of Rudolf Carnap [4], what we have done is just setting a *Carnapian system* as it is described in [8]
In the next section we will see how this approach to learning bayesian networks can be applied to TAN induction.

3 Tree Augmented Naive Bayes

Tree Augmented Naive Bayes (TAN) appears as a natural extension to the *Naive Bayes* classifier. *Naive Bayes* [16, 19, 6] is a very simple classifier that performs very well on small and not-so-small datasets. The assumption made by *Naive Bayes* is that all the attributes in the dataset are conditionally independent given the value of the class. This is a very strong assumption that is very likely not to be fulfilled, but the classifier works well in practice even when strong dependencies hold in the dataset. Furthermore, it has been shown to be optimal under zero-one loss in a larger subspace [6]. Given these facts,

the general idea is that if we somehow relax the assumptions that are made and keep the "way of reasoning", we can get a more accurate classifier. This has been tried in different ways [9, 13, 14, 15, 18, 21]. From our point of view TAN are the more coherent and best performing enhancement to *Naive Bayes* up to now. In this section we discuss the TAN induction algorithm presented at [9]. After that we apply the *multinomial sampling approach* to the TAN induction problem and get a maximum likelihood TAN algorithm that uses the same estimates for probabilities in every step, hence providing a theoretically founded approach to how softening should be done. We propose this algorithm as a good solution to the first weak point identified in the introduction.
To talk of the classification problem we will use the common notation of for distinguishing between the random variable we want to predict (the class, C) and all the rest (the attributes, $A_1, \ldots, A_n$).

3.1 Learning TAN

TAN are a restricted family of bayesian networks in which the class variable has no parents and each attribute has as parents the class variable and at most one other attribute. The interesting property of this family is that we have an efficient procedure for identifying the structure of the network with maximum likelihood. The procedure and the theorem are given below.

```
procedure Construct-TAN (ProbabilityDistribution P)
    var
        WeightMatrix I_P;
        UndirectedGraph UG;
        UndirectedTree UT;
        DirectedTree T;
        DirectedGraph TAN;
    foreach A_i, A_j
        Compute I_P(A_i; A_j|C) =   Σ                P(x,y,z)log( P(x,y|z) / (P(x|z)P(y|z)) )
                                 x ∈ Val(A_i)
                                 y ∈ Val(A_j)
                                 z ∈ Val(C)
    end
    G = ConstructUndirectedGraph(I_P);
    UT = MaximumWeightedSpanningTree(G);
    T = MakeDirected(UT);
    TAN = AddClass(T);
    return TAN;
```

Figure 1: TAN construction procedure

Theorem 1 (Friedman, Geiger & Goldszmidt, 1997) *Let D be a collection of N instances of $C, A_1, \ldots, A_n$. The procedure* `Construct-TAN`$(Freq_D)$ *builds a TAN B_T that maximizes $LL(B_T|D)$ and has time complexity $O(n^2N)$.*

So we have a way to determine the model that best fits the data at hand. To learn the maximum likelihood TAN we should use Theorem 1 to determine the structure and

Equation 4 to determine the weights. Surprisingly, the empirical classification accuracy is improved by softening [9]. Since neither Theorem 1 nor Equation 4 are asymptotic, the explanation given in [9] (summarizing, that the number of observations is not enough to estimate reliably the probabilities) is not theoretically satisfying. A plausible explanation to this phenomenon could be related to the fact (usually disregarded) that the result given in Equation 4 only holds when $Freq_D$ is strictly positive and defined everywhere. From our point of view, the most likely explanation comes from the fact that, by following Statement 1 we are focusing on fitting the data, and not on predicting future events. That is why we propose the usage of the *multinomial sampling approach* to provide a solution to the problem.

3.2 Applying the multinomial sampling approach to TAN induction

We will start by adapting Equation 4 to the *multinomial sampling approach*. In this case, instead of looking for maximum likelihood, we would like to minimize the *cross entropy* or Kullback-Leibler divergence [17] between the P_S^* and the probability distribution generated by the TAN (following the spirit of Statement 2).We can see that *cross entropy* is minimized when:

$$\theta_{x_i|\Pi_{x_i}} = P_S^*(x_i|\Pi_{x_i}) = \frac{P_S^*(x_i, \Pi_{x_i})}{P_S^*(\Pi_{x_i})} = \frac{\sum\limits_{\overline{\Pi_{x_i}} \in Val(\overline{\Pi_{X_i}})} P_S^*(x_i, \Pi_{x_i}, \overline{\Pi_{x_i}})}{\sum\limits_{\substack{x_i \in Val(X_i) \\ \overline{\Pi_{x_i}} \in Val(\overline{\Pi_{X_i}})}} P_S^*(x_i, \Pi_{x_i}, \overline{\Pi_{x_i}})} \tag{6}$$

where $\overline{\Pi_{X_i}}$ stands for the set of variables which are not parents of X_i in the network. $\overline{\Pi_{X_i}}$ does not include X_i. We define $SC(X_i) = \prod\limits_{X_j \in \overline{\Pi_{X_i}}} \#Val(X_j)$. $SC(X_i)$ is the number of different states of the multinomial for which you have to add up the probability in order to calculate $P_S^*(x_i, \Pi_{x_i})$. From this definition and Equation 6 we get:

$$\theta_{x_i|\Pi_{x_i}} = \frac{Count_D(x_i, \Pi_{x_i}) + \lambda \frac{SC(X_i)}{States(P^*)}}{Count_D(\Pi_{x_i}) + \lambda \frac{\#Val(X_i) SC(X_i)}{States(P^*)}} \tag{7}$$

In the case of TAN induction, we have to set three different kind of parameters: $\theta_{a_i|c,a_j}$, $\theta_{a_i|c}$ and θ_c. In these concrete three cases Equation 7 simplifies to:

$$\begin{aligned}
\theta_{a_i|c,a_j} &= \frac{Count_D(a_i,c,a_j) + \frac{\lambda}{\#Val(C)\#Val(A_i)\#Val(A_j)}}{Count_D(c,a_j) + \frac{\lambda}{\#Val(C)\#Val(A_j)}} \\
\theta_{a_i|c} &= \frac{Count_D(a_i,c) + \frac{\lambda}{\#Val(C)\#Val(A_i)}}{Count_D(c) + \frac{\lambda}{\#Val(C)}} \\
\theta_c &= \frac{Count_D(c) + \frac{\lambda}{\#Val(C)}}{N+\lambda}
\end{aligned} \tag{8}$$

In Figure 2 we have the simple algorithmical description of the maximum likelihood TAN induction algorithm proposed. This algorithm uses P_S^* as its probability estimate at every place. Empirical results are given and explained in Section 5.

```
procedure Learn-TAN (Dataset D)
    var
        ProbabilityDistribution P*_S
        DirectedGraph TAN;
    begin
        Calculate P*_S by using Equation 5
        TAN = Construct-TAN(P*_S)
        Set the weights according to Equation 8
```

Figure 2: Complete TAN learning procedure

4 Local Bayesian Model Averaging

The second weak point in the TAN induction algorithm of [9] is that they ignore uncertainty in model selection. *Bayesian model averaging* (BMA) [12] provides a coherent mechanism for accounting for uncertainty in modelling. In this section we review BMA, and introduce *local bayesian model averaging* (LBMA), a practical way of implementing BMA.

4.1 Bayesian Model Averaging

When trying to solve a classification problem, data analysts typically select a model from some class of models and then proceed as if the selected model had generated the data, using the model to predict over the test data. This approach ignores the uncertainty in model selection, hence leading to predictions that can be inaccurate.
A coherent approach to solve the classification problem is calculating the probability of each class given the data. If we assume that the data has been generated from a model that is contained in a class of models $\mathcal{M}$, the probability distribution of the class given data, is:

$$P(C|I,S) = \sum_{M \in \mathcal{M}} P(C|M,I)P(M|S) \tag{9}$$

where $P(C|M,I)$ is the probability distribution of the class when we know the model M that generated the data, and the value of the attributes for this instance I. $P(M|S)$ is the probability that M is the model that generated the data given the sample S.
Equation 9 tells us not to use a single model to classify the data, but instead to use all the models from the class of models, weighting each model prediction by the probability of the model given the sample of data we are analyzing. Using Equation 9 to predict is known as *Bayesian Model Averaging* or BMA for short. BMA produces optimally accurate predictions within the chosen model family from the probability theory point of view.
In order to use BMA in practice, we need to further develop Equation 9. It can be expanded as:

$$P(M|S) = \frac{P(S|M)P(M)}{\sum_{M' \in \mathcal{M}} P(S|M')P(M')} \tag{10}$$

Here $P(M)$ is the prior probability that M is the real model and $P(S|M)$ is the probability that model M generates the data in S.
BMA is a very attractive solution to the problem of accounting for model uncertainty and has been often used in some form or another in the machine learning community. In [16] BMA is applied to Naive Bayes, and it is shown that it improves both classification accuracy and the quality of the probability estimates. In [1, 5], it is applied to rule induction and in [3] to decision tree induction, in both cases leading to good results.

4.2 Local Bayesian Model Averaging

In practice, the usage of BMA presents some problems, coming from:

- The computational cost of calculating Equation 9.
- The difficulty in the specification of $P(M)$, the prior distribution over competing models.

In order to handle the first of these problems, we propose LBMA, an heuristic approach to approximate BMA. The idea is similar in spirit to the Occam's Window method described in [12, 20]. To apply LBMA we should have an heuristic $h(M, S)$ such that

$$h(M, S) \approx P(M|S) \tag{11}$$

In order to approximate the summation in Equation 9, and given that we dispose of $h(M, S)$, we define our set of *interesting models* $\mathcal{M}'$ as:

$$\mathcal{M}' = \{M \in \mathcal{M} | h(M, S) \geq \gamma\} \tag{12}$$

γ represents a compromise between the prediction accuracy and its computational cost. It should be big enough to make $\#\mathcal{M}' \ll \#\mathcal{M}$, but small enough in order for

$$P(M|S) \approx P'(M|S) = \frac{P(S|M)P(M)}{\sum\limits_{M' \in \mathcal{M}'} P(S|M')P(M')} \tag{13}$$

$$P(C|I, S) \approx \sum_{M \in \mathcal{M}'} P(C|M, I)P'(M|S) \tag{14}$$

to be accurate approximations. It is interesting to note that maximum likelihood prediction is a concrete case of LBMA where $h(M, S) = P(S|M)$ and γ is implicitly set in order for $\mathcal{M}'$ to contain only a model. We describe LBMA algorithmically in Figure 3 to ease its understanding. Once we have the resulting weighted set of models calculated, we can use it to classify by calculating Equation 14 for each class and choosing the one with higher probability.

4.3 Local Bayesian Model Averaging for TAN induction

In this section we will show the application of LBMA to the case of TAN induction. For this concrete case, our class of models $\mathcal{M}$ is

$$\mathcal{M} = \{\langle G, \Theta\rangle | G \in TANStructures, \Theta \in Parameters(G)\} \tag{15}$$

We perform a first reduction of $\mathcal{M}$ by using the results in Equation 8, Equation 4 or the *softened* method proposed in [9] depending on whether we decide to use the *multinomial sampling approach*, the log likelihood principle or the ad hoc adjustment proposed in [9]. In any case, we will only average over the structures, fixing the parameters by using the corresponding equation in each case. Our heuristic over structures will be given by the algorithm `Construct-TAN`, just modifying the step where a maximum spanning tree is induced, to generate a set containing the K maximum spanning trees by using Gabow algorithm [11]. In order to calculate $P'(M)$, we set a prior over tree structures that assigns the same probability to each possible tree structure (since they can be considered of a similar complexity). We also have to provide an implementation for `Predict`, that is, we have to know how to calculate $P(C = c|M, u)$. In a TAN:

$$P(C = c|M, u) = P_M(c|u_{i,1}, \ldots, u_{i,n}) = \frac{P_M(c, u_{i,1}, \ldots, u_{i,n})}{\sum_{c' \in C} P_M(c', u_{i,1}, \ldots, u_{i,n})} = \frac{\prod_{j=1}^{n} \theta_{u_{i,j}|c, u_{i,\pi(A_j)}}}{\sum_{c' \in C} \prod_{j=1}^{n} \theta_{u_{i,j}|c', u_{i,\pi(A_j)}}} \tag{16}$$

where $u_{i,\pi(A_j)}$ is adequately set to the value of the parent of A_j in the tree or nothing if A_j does not has parents in it.

The algorithmic description of the complete LBMA TAN induction procedure appears in Figure 4. The version appearing is the one for the *multinomial sampling approach*. To use LBMA with the TAN induction algorithm proposed in [9] only a few small adjustments should be done to the algorithm appearing in Figure 4.

Applying LBMA to TAN induction we are simultaneously providing a solution for the second and third weak points in [9] noticed in the introduction. It is clear that LBMA is approximation to BMA, and hence focuses on the second point, that is, taking into account the uncertainty in model selection. The third point was that what should be maximized is the conditional likelihood instead of the likelihood. This weak point was already noticed in their paper, where they stated that it was an open question whether good heuristic approaches can be found in order to induce TAN models that maximize conditional likelihood. The way LBMA weights the different models is exactly multiplying each model by its conditional likelihood. We are then trusting more those models with a higher conditional likelihood. The problem is far from being solved, but we think this a good first step.

4.4 Computational complexity

The computational complexity of the `Learn-TAN` procedure in Figure 2 is $O(N \cdot n^2)$. For the general `LBMA` procedure appearing in Figure 3 the costs are:

$$\begin{aligned} Cost(\texttt{Predict}) &= O(c \cdot n) \\ Cost(\texttt{CalculateProb}) &= O(N \cdot c \cdot n) \\ Cost(\texttt{LBMA}) &= O(k \cdot N \cdot c \cdot n) \end{aligned} \tag{17}$$

where $k = \#\mathcal{M}'$, c is the number of classes and n the number of attributes.
Knowing that:

$$\begin{aligned} Cost(Probability\ Approximation) = Cost(Counting) &= O(N \cdot n^2) \\ Cost(Model\ Proposal) &= O(n^2 \cdot \log n) \end{aligned} \tag{18}$$

we have that the total computational cost of the TAN induction algorithm that appears in Figure 4 using BMA is

$$\begin{aligned} Cost(\texttt{LBMA-TAN}) &= Cost(Counting) + Cost(Model\ Proposal) + Cost(\texttt{LBMA}) \\ &= O(N \cdot n \cdot (n + k \cdot c)) \end{aligned} \tag{19}$$

This means that as long as we keep $\mathcal{M}'$ small, the computational overhead will not be large. It will grow linearly on the number of models, and can even be disregarded from an asymptotic point of view if $k \cdot c < n$.
After considering the cost of induction of the classifier, we should consider the cost of applying the classifier to new data. The cost of classifying a new instance with a single TAN model this is $O(c \cdot n)$. With a multiple TAN model the cost is $O(k \cdot c \cdot n)$.

5 Experimental results

5.1 Adjusting the algorithm to run

In order to use the algorithm described in Section 4.3, we need to set some parameters. In our experimental setting, we took:

$$\begin{aligned} k &= \min(10, n) \\ \lambda &= 10 \end{aligned} \tag{20}$$

k was set in order to show that you do not need to average over a large set of models in order to improve accuracy. To fix λ we just tried out a few values and selected the one that performed best. At this point we should discuss a little more about this parameters and its usefulness. We would like to point that k (and equivalently γ in the general version of LBMA) can act as an *effort knob*, in the sense of [22], hence providing a useful feature for data mining users that allows them to decide how much computational power they want to spend in the task. λ offers the user a second dimension in which look for parameters. Even when the relevance of λ decreases as the number of observations increases, a good λ adjustment can seriously improve the quality of the prediction.

5.2 Experimental setting

We tested five algorithms over 14 datasets from the Irvine repository [2] plus our own credit screening database. The dataset characteristics are described in Table 1. To discretize continuous attributes we tried *maximum entropy* [7] discretization and equal frequency discretization with 5 intervals. We present the results for equal frequency because it provided better accuracy. For each dataset and algorithm we tested both accuracy and *LogScore*. *LogScore* is calculated by adding the minus logarithm of the probability assigned by the classifier to the correct class and gives an idea of how well the classifier is estimating probabilities (the smaller the score the better the result).

$$LogScore(M, S') = \sum_{i=1}^{N'} -\log(P_M(u'_i)) \tag{21}$$

The focus of most research in machine learning algorithms is on improving accuracy. There are many cases in the real life application of these algorithms, where not only accuracy is interesting, but also the quality of the probability estimations for each class is very important. Two common examples are the selection of people likely to respond to mailing campaigns and when the results of the learning process should lead to decisions being taken by a human who has to have an estimate of the decision risk. That is why we included *LogScore* in our testing. For the evaluation of both error rate and *LogScore* we used 10 fold cross validation.
The error rates appear in Table 2, with the best method for each dataset boldfaced. *LogScore*'s appear in Table 3. The columns of the tables are the induction methods and the rows are the datasets. The meaning of the column headers are:

- NB is the Naive Bayes algorithm (implemented as BIBL in [16]).
- TAN+MS is the maximum likelihood TAN induction using the *multinomial sampling approach*
- TAN+MS+BMA, is the method described in Section 4.3 tuned with the parameters specified in subsection 5.1.
- sTAN is the softened TAN algorithm as described in [9]
- sTAN+BMA is the result of applying LBMA directly to the FGG algorithm.

5.3 Interpretation of the results

In order to make sense of all the numbers in Table 2 and Table 3 we have selected some comparisons and have put them into separate figures. For the different comparisons we took away the datasets in which NB is the best classifier, because in that case NB will be the classifier of choice.

Dataset	Attributes	Instances	Classes	Missing
DCREDITS	5	3781	15	few
ADULT	14	48842	2	some
BREAST	10	699	2	16
CAR	6	1728	4	no
CHESS	36	3196	2	no
CRX	15	690	2	few
FLARE	10	323	4	no
GLASS	10	214	2	none
HEP	19	155	2	some
IRIS	4	150	3	none
MUSHROOM	22	8124	2	some
NURSERY	8	12960	5	no
PIMA	8	768	2	no
SOYBEAN	35	316	19	some
VOTES	16	435	2	few

Table 1: Datasets information

Dataset	NB	sTAN+BMA	TAN+MS+BMA	sTAN	TAN+MS
DCREDITS	17.86 ± 1.47	14.39 ± 1.32	14.16 ± 1.58	14.29 ± 1.27	$\mathbf{13.98 \pm 1.49}$
BREAST	$\mathbf{4.05 \pm 1.35}$	4.46 ± 2.17	5.76 ± 1.40	5.64 ± 1.45	5.05 ± 1.35
FLARE	23.96 ± 4.20	$\mathbf{18.57 \pm 2.78}$	20.02 ± 3.16	18.86 ± 3.11	20.09 ± 3.07
HEP	22.27 ± 10.28	21.39 ± 9.96	20.96 ± 12.56	$\mathbf{20.14 \pm 8.54}$	21.81 ± 13.94
MUSHROOM	4.68 ± 0.97	0.14 ± 0.04	$\mathbf{0.12 \pm 0.00}$	0.14 ± 0.04	$\mathbf{0.12 \pm 0.00}$
VOTES	11.86 ± 4.75	8.26 ± 4.18	$\mathbf{7.60 \pm 3.33}$	8.49 ± 4.40	8.26 ± 4.18
CAR	14.99 ± 2.63	6.30 ± 1.39	$\mathbf{5.85 \pm 1.37}$	6.25 ± 1.36	$\mathbf{5.85 \pm 1.37}$
CRX	$\mathbf{15.36 \pm 4.38}$	17.20 ± 3.62	18.62 ± 4.31	17.64 ± 3.73	18.37 ± 4.12
GLASS	$\mathbf{22.01 \pm 8.00}$	29.69 ± 17.23	27.04 ± 12.12	29.62 ± 15.68	28.90 ± 11.95
IRIS	16.89 ± 7.25	$\mathbf{12.28 \pm 5.61}$	13.39 ± 5.82	12.84 ± 5.18	13.61 ± 4.92
NURSERY	9.86 ± 1.17	$\mathbf{4.85 \pm 0.99}$	4.90 ± 1.15	6.76 ± 1.33	6.90 ± 1.39
PIMA	26.32 ± 4.42	$\mathbf{25.74 \pm 4.59}$	26.16 ± 4.85	26.19 ± 4.99	26.17 ± 5.13
ADULT	18.47 ± 0.60	16.34 ± 0.65	16.43 ± 0.61	$\mathbf{16.34 \pm 0.63}$	16.42 ± 0.62
CHESS	12.37 ± 1.52	7.71 ± 1.25	$\mathbf{7.53 \pm 1.33}$	8.15 ± 1.52	7.63 ± 1.44
SOYBEAN	11.12 ± 3.92	8.13 ± 3.62	$\mathbf{7.31 \pm 4.22}$	8.01 ± 3.58	7.90 ± 3.92

Table 2: Averages and standard deviations of error rates

5.3.1 TAN+MS against sTAN

We are interested in seeing whether the application of the *multinomial sampling approach* provided any benefits. In Figure 5 we have displayed the percentage of improvement between using TAN+MS and sTAN. Even when the graph seems to favor a little bit TAN+MS we can say that both methods have similar error rate. In Figure 6 we display the percentage of improvement in the $LogScore$, where we can notice one of the problems of sTAN and sTAN+BMA: for some datasets its $LogScore$ goes to infinity because they assign a probability of 0 to the class. Since the cost of sTAN and TAN+MS is the same and TAN+MS has shown the same level of accuracy and a better $LogScore$, plus a higher theoretical consistence by the usage of the same probability distribution in all steps of the algorithm, we recommend the usage of TAN+MS.

5.3.2 TAN+MS+BMA against TAN+MS

To evaluate whether it is useful or not to use LBMA, we show in Figure 7 the percentage of improvement between using TAN+MS+BMA and TAN+MS and in Figure 8 the

Dataset	NB	sTAN+BMA	TAN+MS+BMA	sTAN	TAN+MS
DCREDITS	197.60 ± 22.99	**156.17 ± 15.90**	160.10 ± 20.07	158.81 ± 13.21	158.24 ± 18.14
BREAST	18.21 ± 9.88	**8.53 ± 5.22**	11.97 ± 6.37	10.62 ± 5.81	15.70 ± 8.05
FLARE	87.32 ± 20.92	1573.46 ± 4503.51	**84.37 ± 18.43**	1574.05 ± 4503.34	86.35 ± 19.78
HEP	3.18 ± 3.01	1.80 ± 1.26	2.38 ± 2.24	**1.73 ± 1.20**	3.04 ± 2.94
MUSHROOM	111.03 ± 30.00	0.15 ± 0.26	**0.01 ± 0.01**	0.36 ± 0.51	0.02 ± 0.03
VOTES	27.20 ± 15.49	8.47 ± 5.98	8.58 ± 7.12	8.74 ± 6.30	**8.46 ± 7.04**
CAR	58.11 ± 9.23	37.49 ± 6.36	31.98 ± 5.69	37.47 ± 6.35	**31.96 ± 5.68**
CRX	**29.75 ± 9.64**	31.16 ± 9.56	43.36 ± 13.90	31.19 ± 8.43	43.68 ± 14.76
GLASS	**9.59 ± 4.77**	17.16 ± 7.01	20.31 ± 12.43	17.66 ± 7.12	27.76 ± 16.03
IRIS	3.92 ± 2.51	2.57 ± 1.36	**2.34 ± 1.45**	2.58 ± 1.22	2.87 ± 1.65
NURSERY	340.33 ± 23.21	205.79 ± 18.74	**199.97 ± 18.18**	211.23 ± 19.89	206.03 ± 19.40
PIMA	41.61 ± 9.54	39.53 ± 8.53	41.31 ± 9.35	**39.48 ± 8.88**	41.29 ± 10.08
ADULT	1532.36 ± 47.66	**1170.66 ± 32.47**	1186.25 ± 33.48	1170.94 ± 32.20	1186.36 ± 33.17
CHESS	93.42 ± 7.08	59.39 ± 6.15	58.43 ± 5.86	60.46 ± 6.15	**58.38 ± 5.75**
SOYBEAN	54.58 ± 34.78	**12.03 ± 5.20**	16.04 ± 9.78	12.37 ± 5.30	16.69 ± 9.77

Table 3: Averages and standard deviations of *LogScore*

improvement in *LogScore*. We can see that in most of the cases the application of BMA gives the same or better results both in error rate and *LogScore*.

5.3.3 TAN+MS+BMA against sTAN

To end up with the analysis we plot in Figure 9 the percentage of improvement in error rate between using TAN+MS+BMA and sTAN, and in Figure 10 the improvement in *LogScore*. This two figures are useful as a comparison of our proposed TAN induction method TAN+MS+BMA, against the method proposed in [9] and favor TAN+MS+BMA.

6 Conclusions and future research avenues

6.1 Conclusions

We have proposed solutions for correcting the three weak points we noticed in TAN induction as it is done in [9]. We have introduced the *multinomial sampling approach*, a new approach to learning bayesian networks that provides a coherent way of estimating probabilities, and have used it to develop a theoretically coherent maximum likelihood TAN induction algorithm. We have introduced *local bayesian model averaging* and have used it to account for uncertainty in the selection of the model. By using LBMA, we have weighted each model by its conditional likelihood, instead of by its likelihood, thus providing a partial solution to the third point that we mentioned in the introduction, i.e. that conditional likelihood and not likelihood is what should be maximized.
We have provided empirical evidence that shows that in most of the cases the resulting new method provides more accurate predictions and probability estimates. Furthermore, we think that the concepts presented in the paper provide a better understanding of TAN induction in particular and bayesian networks learning in general.

6.2 Further research

Looking into the final weights of the different models suggests that we can take an adaptive approach to BMA, starting with a large number of models and reducing it progressively as we accumulate evidence against them (some of the weights were of the order of 10^{-20} even for not very large datasets). We have selected a standard Dirichlet uniform distribution as prior for the *multinomial sampling approach* estimation. There are some other interesting choices for priors. Concretely it will be interesting to see how a prior designed to deal with a large number of states, as the one developed in [10], affects the performance of the algorithm. It should also be studied how the different values of λ affect the performance and whether one can develop methods to adjust λ automatically.

7 Acknowledgements

I would like to thank Maria Luisa Barja and Ramon López de Mántaras for carefully reviewing the preliminary versions. I would like to specially thank Maria Luisa for accepting to carry with additional work in order to give me the time to write it.

References

[1] K. Ali, C. Brunk, and M. Pazzani. Learning multiple relational rule-based models. In *Preliminary papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, 1995.

[2] C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases, 1998.

[3] Wray Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[4] Rudolf Carnap. *The Continuum of Inductive Methods.* University of Chicago Press, 1952.

[5] Pedro Domingos. Bayesian model averaging in rule induction. In *Preliminary papers of the Sixth International Workshop on Artificial Intelligence and Statistics*, pages 157–164, 1997.

[6] Pedro Domingos and Michael Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29:103–130, 1997.

[7] Usama M. Fayyad and Keki B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *13th International Joint Conference of Artificial Intelligence*, pages 1022–1027, 1993.

[8] Roberto Festa. *Optimum inductive methods: a study in inductive probability, Bayesian statistics and verisimilitude.* Kluwer Academic Publishers, 1993.

[9] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[10] Nir Friedman and Yoram Singer. Efficient bayesian parameter estimation in large discrete domains. In *Neural Information Processing Systems (NIPS 98)*, 1998.

[11] Harold N. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM J. COMPUT.*, 6(1):139–150, March 1977.

[12] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging. Technical Report 9814, Department of Statistics. Colorado State University, 1998.

[13] Eamonn J. Keogh and Michael Pazzani. Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Uncertainty 99: The Seventh International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, 1999.

[14] Ron Kohavi and George H. John. Wrappers for Feature Subset Selection. *AI Journal*, Special Issue on Relevance.

[15] I. Kononenko. Semi-naive bayesian classifier. In Y. Kodratoff, editor, *Proc. Sixth European Working Session on Learning*, pages 206–219. Berlin: Springer-Verlang, 1991.

[16] Petri Kontkanen, Petri Myllymaki, Tomi Silander, and Henry Tirri. Bayes Optimal Instance-Based Learning. In C. Nédellec and C. Rouveirol, editors, *Machine Learning: ECML-98, Proceedings of the 10th European Conference*, volume 1398 of *Lecture Notes in Artificial Intelligence*, pages 77–88. Springer-Verlag, 1998.

[17] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:76–86, 1951.

[18] P. Langley and S. Sage. Induction of selective bayesian classifiers. In R. López de Màntaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 399–406. San Francisco,CA: Morgan Kaufmann, 1994.

[19] Pat Langley, Wayne Iba, and Kevin Thompson. An Analysis of Bayesian Classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 223–228. AAAI Press and MIT Press, 1992.

[20] David Madigan and Adrian. E. Raftery. Model selection and accounting for model uncertainty in graphical models using occam's window. *J. American Statistical Association*, 89:1535–1549, 1994.

[21] Michael Pazzani. Searching for dependencies in bayesian classifiers. In D. Fisher and H. Lenz, editors, *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, 1995.

[22] Kurt Thearling. Some thoughts on the current state of data mining software applications. In *Keys to the Commercial Success of Data Mining, KDD'98 Workshop*, 1998.

```
procedure LBMA-Main (Dataset S,Real γ,Heuristic h)
    var
        WeightedSetOfModels Result;
    begin
        Calculate M' using h and γ;
        return LBMA(S,M');
    end

procedure LBMA (Dataset S,SetOfModels M')
    var
        WeightedSetOfModels Result;
        ProbabilityDistribution P';
    begin
        foreach M ∈ M'
            P'(M) = CalculateProb(S,M) * P(M);
        end
        Normalize P'(M);
        Result = {(M,P'(M))|M ∈ M'};
        return Result;
    end

/* Calculates P(S|M) */
procedure CalculateProb (Dataset S,ProbabilisticModel M)
    var
        Real P_M;
        ClassProbabilityDistribution P_ThisInstance;
    begin
        P_M = 1;
        foreach u ∈ S
            /* Predict(M,u) returns the probability distribution P(C = c|M,u) */
            P_ThisInstance = Predict(M,u);
            P_M = P_M * P_ThisInstance(u_C);
        end
        return P_M;
    end
```

Figure 3: Local Bayesian Model Averaging

```
procedure LBMA-TAN (Dataset D)
    var
       ProbabilityDistribution P*_S
       DirectedGraphSet M';
    begin
        Calculate P*_S by using Equation 5;
        M' = Construct-K-TAN(P*_S,k);
        foreach M ∈ M'
            Set the weights of M according to Equation 8;
        end
        return LBMA(S,M');
    end

procedure Construct-K-TAN (ProbabilityDistribution P, Integer k)
    var
       WeightMatrix I_P;
       UndirectedGraph UG;
       UndirectedTreeSet UTS;
       DirectedTreeSet TS;
       DirectedGraphSet M';
    begin
        foreach A_i,A_j
            Compute I_P(A_i;A_j|C) as in Construct-TAN
        end
        G = ConstructUndirectedGraph(I_P);
        /* Returns the k maximum weighted spanning trees */
        UTS = K-MaximumWeightedSpanningTree(G,k);
        TS = MakeDirected(UTS);
        M' = AddClass(TS);
        return M';
    end
```

Figure 4: LBMA TAN learning procedure

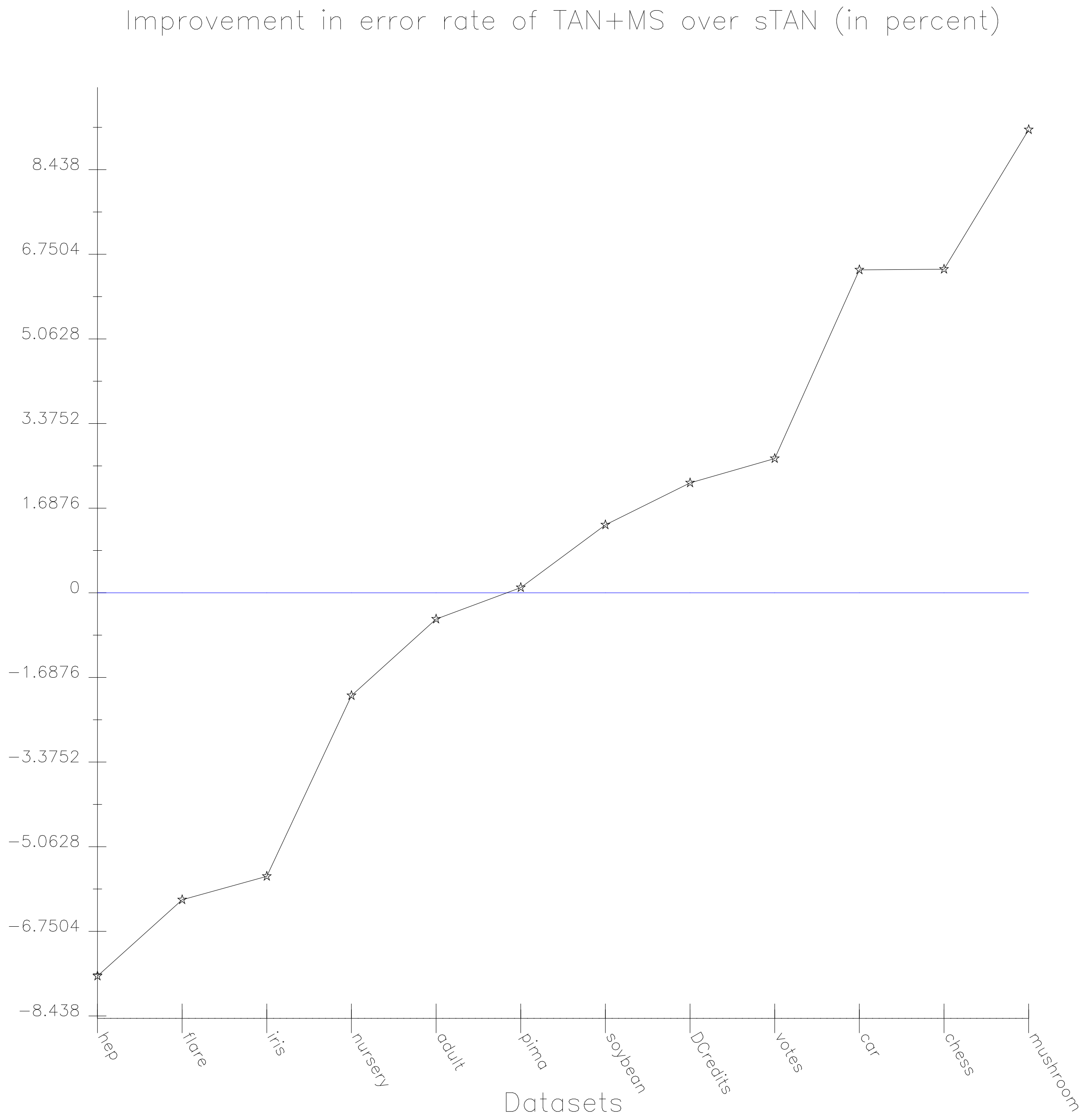

Figure 5: Comparison of the error rate of TAN+MS and sTAN

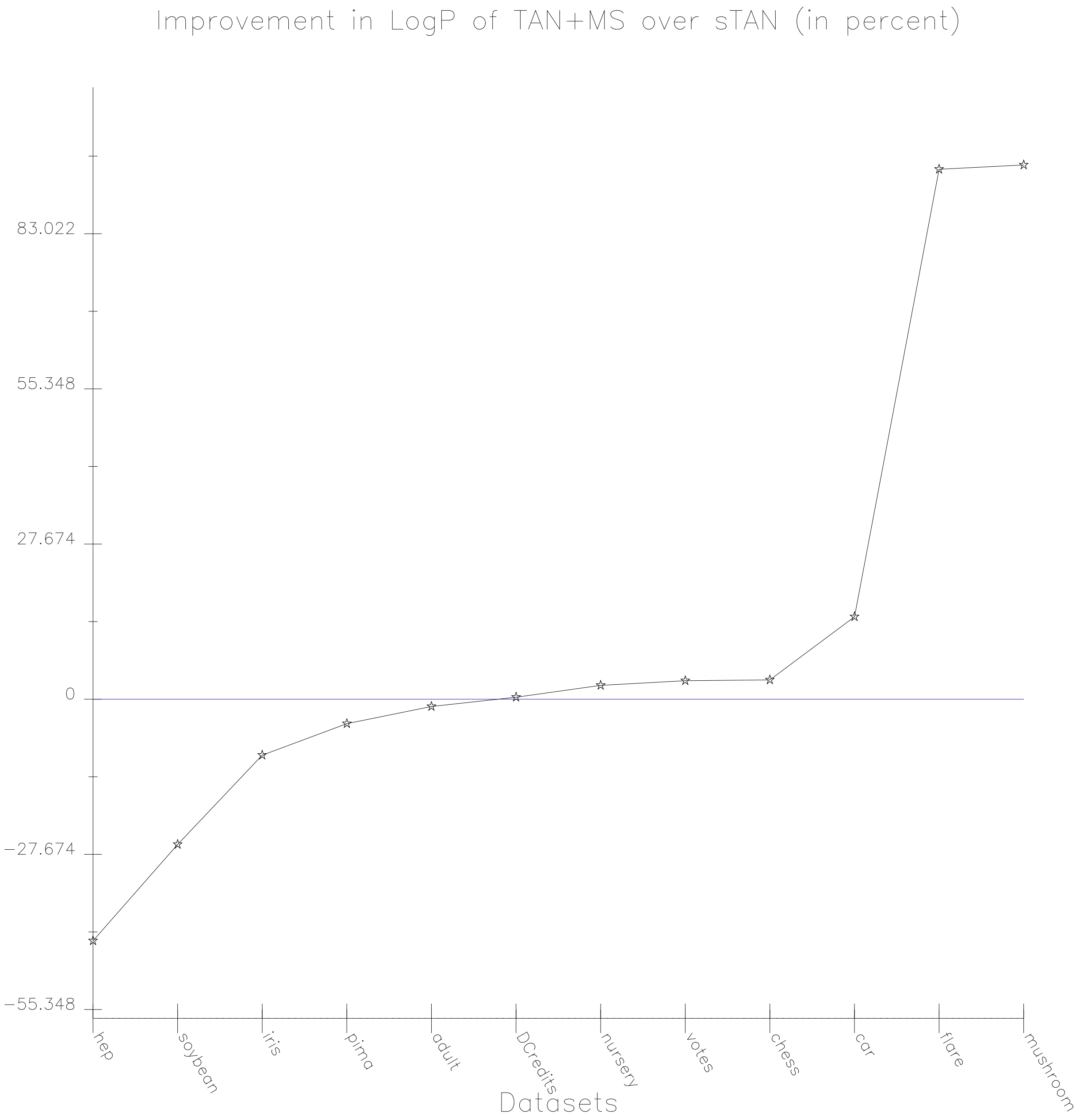

Figure 6: Comparison of the *LogScore* of TAN+MS and sTAN

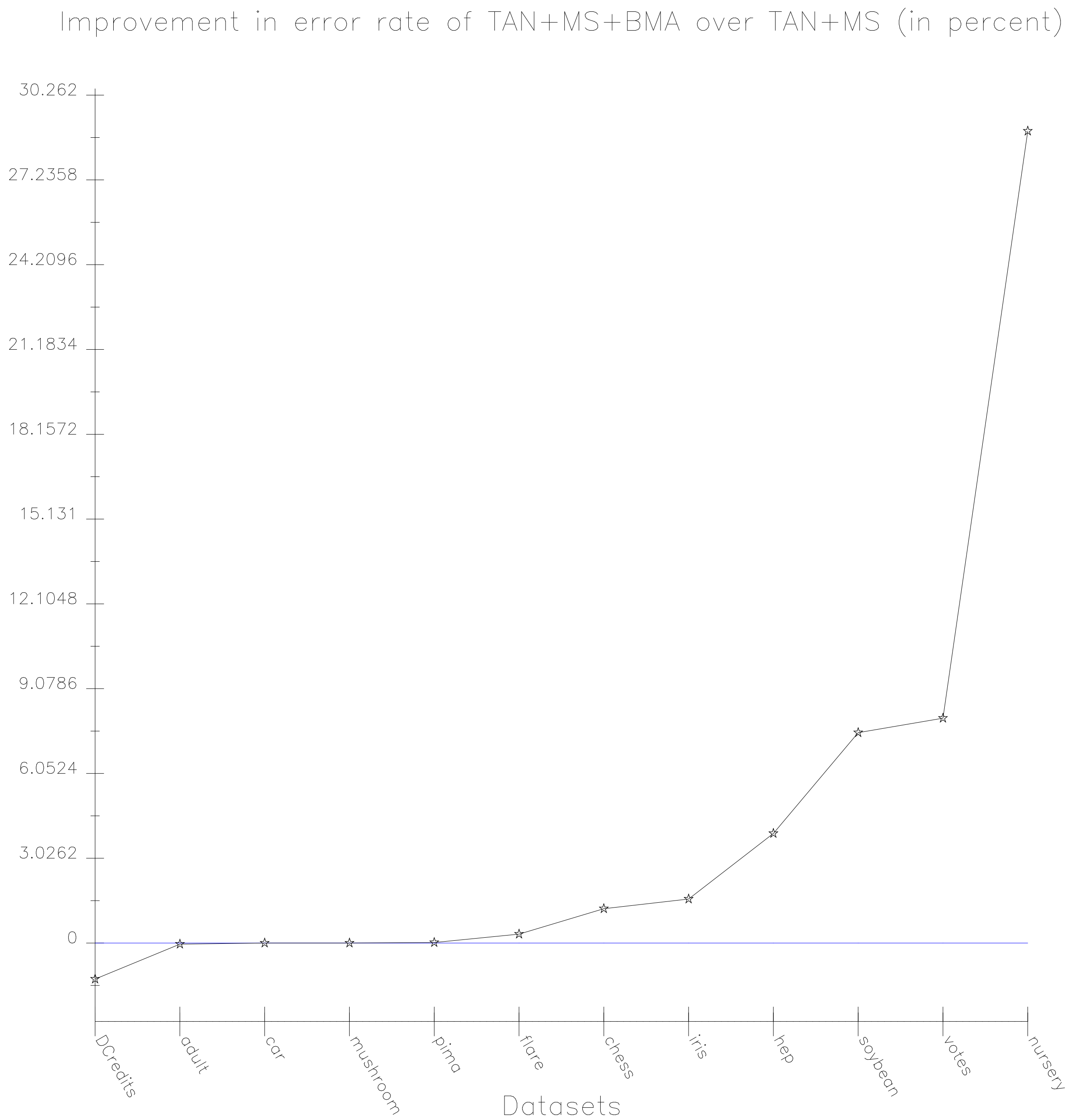

Figure 7: Comparison of the error rate of TAN+MS+BMA and TAN+MS

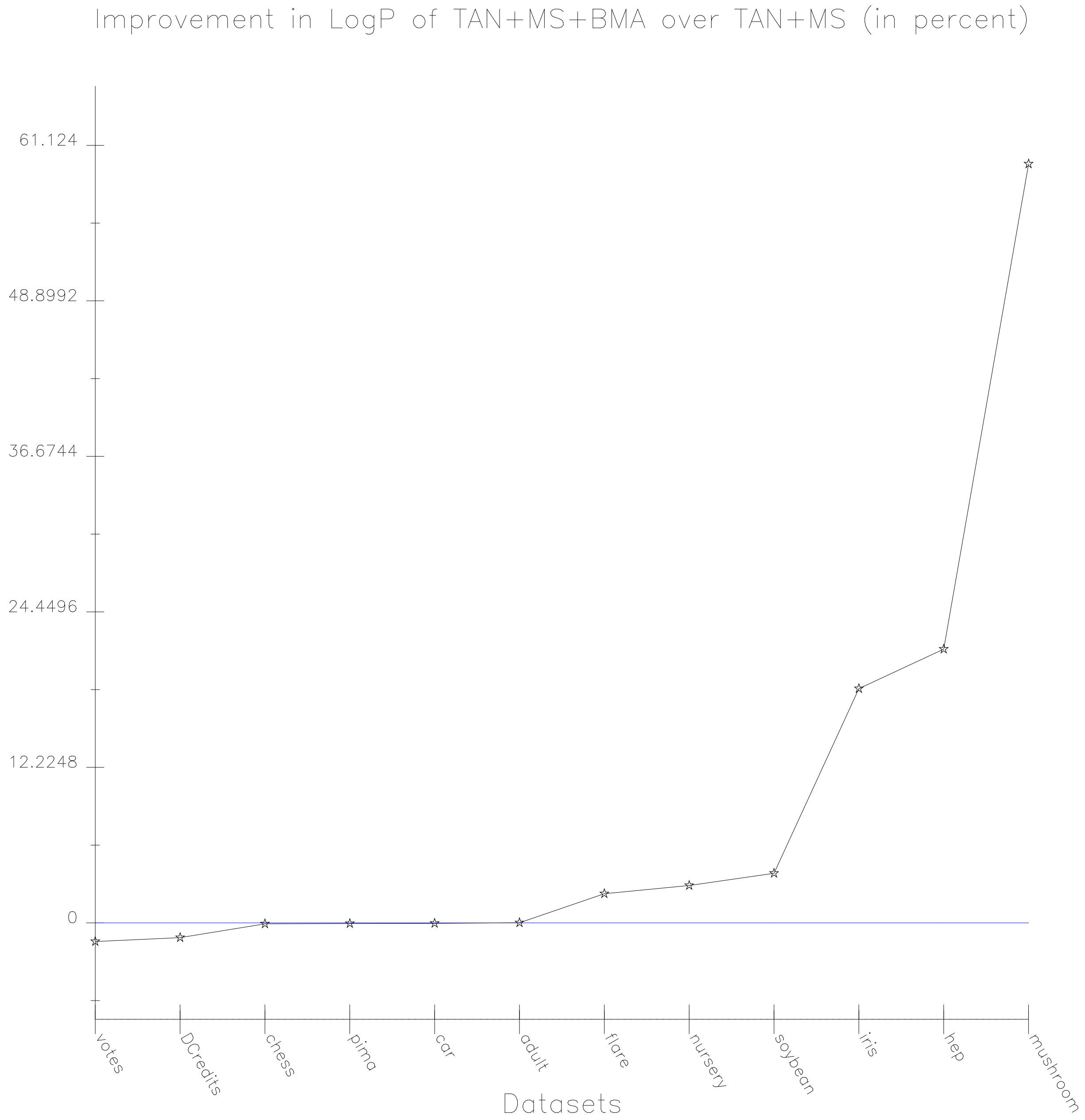

Figure 8: Comparison of the *LogScore* of TAN+MS+BMA and TAN+MS

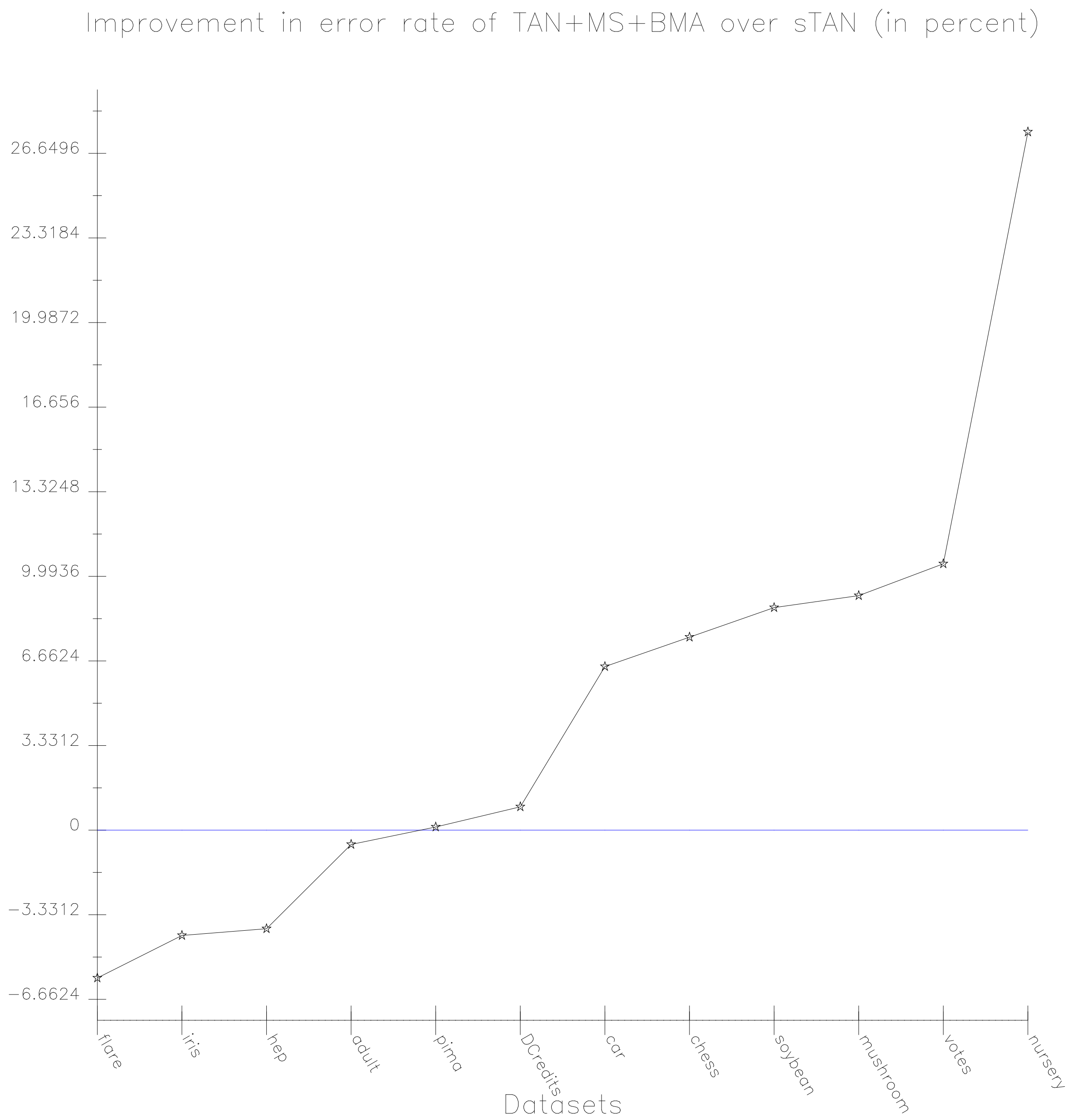

Figure 9: Comparison of the error rate of TAN+MS+BMA and sTAN

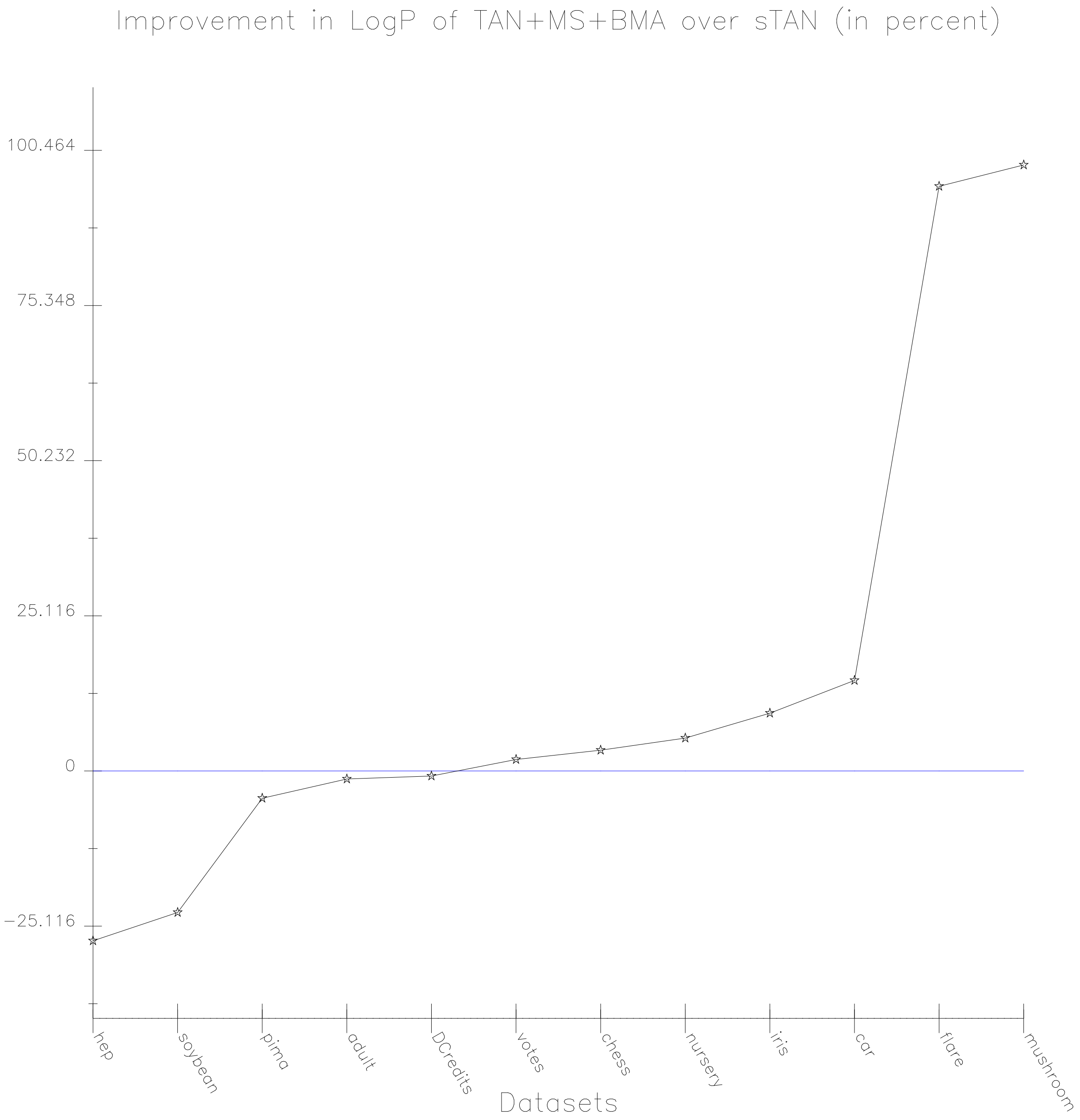

Figure 10: Comparison of the *LogScore* of TAN+MS+BMA and sTAN