Extended Tree Augmented Naive Classifier

Cassio P. de Campos[1], Marco Cuccu[2], Giorgio Corani[1], and Marco Zaffalon[1]

[1] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Switzerland
[2] Università della Svizzera italiana (USI), Switzerland
{cassio,giorgio,zaffalon}@idsia.ch, marco.cuccu@usi.ch

Abstract. This work proposes an extended version of the well-known tree-augmented naive Bayes (TAN) classifier where the structure learning step is performed without requiring features to be connected to the class. Based on a modification of Edmonds' algorithm, our structure learning procedure explores a superset of the structures that are considered by TAN, yet achieves global optimality of the learning score function in a very efficient way (quadratic in the number of features, the same complexity as learning TANs). A range of experiments show that we obtain models with better accuracy than TAN and comparable to the accuracy of the state-of-the-art classifier averaged one-dependence estimator.

1 Introduction

Classification is the problem of predicting the *class* of a given object on the basis of some attributes (*features*) of it. A classical example is the iris problem by Fisher: the goal is to correctly predict the *class*, that is, the species of iris on the basis of four features (sepal and petal length and width). In the Bayesian framework, classification is accomplished by updating a prior density (representing the beliefs before analyzing the data) with the likelihood (modeling the evidence coming from the data), in order to compute a posterior density, which is then used to select the most probable class.

The naive Bayes classifier [1] is based on the assumption of stochastic independence of the features given the class; since the real data generation mechanism usually does not satisfy such a condition, this introduces a bias in the estimated probabilities. Yet, at least under the zero-one accuracy, the naive Bayes classifier performs surprisingly well [1, 2]. Reasons for this phenomenon have been provided, among others, by Friedman [3], who proposed an approach to decompose the misclassification error into bias error and variance error; the bias error represents how closely the classifier approximates the target function, while the variance error reflects the sensitivity of the parameters of the classifier to the training sample. Low bias and low variance are two conflicting objectives; for instance, the naive Bayes classifier has high bias (because of the unrealistic independence assumption) but low variance, since it requires to estimate only a few parameters. A way to reduce the naive Bayes bias is to relax the independence assumption using a more complex graph, like a tree-augmented naive Bayes (TAN) [4]. In particular, TAN can be seen as a Bayesian network where

each feature has the class as parent, and possibly also a feature as second parent. In fact, TAN is a compromise between general Bayesian networks, whose structure is learned without constraints, and the naive Bayes, whose structure is determined in advance to be naive (that is, each feature has the class as the only parent). TAN has been shown to outperform both general Bayesian networks and naive Bayes in a range of experiments [5, 4, 6].

In this paper we develop an extension of TAN that allows it to have (i) features without the class as parent, (ii) multiple features with only the class as parent (that is, building a forest), (iii) features completely disconnected (that is, automatic feature selection). The most common usage of this model is traditional classification. However it can also be used as a component of a graphical model suitable for multi-label classification [7].

Extended TAN (or simply ETAN) can be learned in quadratic time in the number of features, which is essentially the same computational complexity as that of TAN. The goodness of each (E)TAN structure is assessed through the Bayesian Dirichlet likelihood equivalent uniform (BDeu) score [8, 9, 10]. Because ETAN's search space of structures includes that of TAN, the BDeu score of the best ETAN is equal or superior to that of the best TAN. ETAN than provides a better fit: a higher score means that the model better fits the joint probability distribution of the variables. However, it is well known that this fit does not necessarily imply higher classification accuracy [11]. To inspect that, we perform extensive experiments with these classifiers. We empirically show that ETAN yields in general better zero-one accuracy and log loss than TAN and naive Bayes (where log loss is computed from the posterior distribution of the class given features). Log loss is relevant in cases of cost-sensitive classification [12, 13]. We also study the possibility of optimizing the equivalent sample size of TAN, which makes its accuracy become closer to that of ETAN (although still slightly inferior).

This paper is divided as follows. Section 2 introduces notation and defines the problem of learning Bayesian networks and the classification problem. Section 3 presents our new classifier and an efficient algorithm to learn it from data. Section 4 describes our experimental setting and discusses on empirical results. Finally, Section 5 concludes the paper and suggests possible future work.

2 Classification and Learning TANs

The classifiers that we discuss in this paper are all subcases of a Bayesian network. A Bayesian network represents a joint probability distribution over a collection of categorical random variables. It can be defined as a triple $(\mathcal{G}, \mathcal{X}, \mathcal{P})$, where $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$ is a directed acyclic graph (DAG) with $V_{\mathcal{G}}$ a collection of nodes associated to random variables $\mathcal{X}$ (a node per variable), and $E_{\mathcal{G}}$ a collection of arcs; $\mathcal{P}$ is a collection of conditional mass functions $p(X_i|\Pi_i)$ (one for each instantiation of Π_i), where Π_i denotes the parents of X_i in the graph (Π_i may be empty), respecting the relations of $E_{\mathcal{G}}$. In a Bayesian network every variable is conditionally independent of its non-descendant non-parents

given its parents (Markov condition). Because of the Markov condition, the Bayesian network represents a joint probability distribution by the expression $p(\mathbf{x}) = p(x_0, \ldots, x_n) = \prod_i p(x_i|\boldsymbol{\pi}_i)$, for every $\mathbf{x} \in \Omega_{\mathcal{X}}$ (space of joint configurations of variables), where every x_i and $\boldsymbol{\pi}_i$ are consistent with $\mathbf{x}$.

In the particular case of classification, the *class* variable X_0 has a special importance, as we are interested in its posterior probability which is used to predict unseen values; there are then several feature variables $\mathcal{Y} = \mathcal{X} \setminus \{X_0\}$. The supervised classification problem using probabilistic models is based on the computation of the posterior density, which can then be used to take decisions. The goal is to compute $p(X_0|\mathbf{y})$, that is, the posterior probability of the classes given the values $\mathbf{y}$ of the features in a *test* instance. In this computation, p is defined by the model that has been learned from labeled data, that is, past data where class and features are all observed have been used to infer p. In order to do that, we are given a complete data set $D = \{D_1, \ldots, D_N\}$ with N instances, where $D_u = \mathbf{x}_u \in \Omega_{\mathcal{X}}$ is an instantiation of all the variables, the first learning task is to find a DAG $\mathcal{G}$ that maximizes a given score function, that is, we look for $\mathcal{G}^* = \text{argmax}_{\mathcal{G} \in \boldsymbol{\mathcal{G}}} s_D(\mathcal{G})$, with $\boldsymbol{\mathcal{G}}$ the set of all DAGs with nodes $\mathcal{X}$, for a given score function s_D (the dependency on data is indicated by the subscript D).[3]

In this work we only need to assume that the employed score is decomposable and respects likelihood equivalence. Decomposable means it can be written in terms of the local nodes of the graph, that is, $s_D(\mathcal{G}) = \sum_{i=0}^{n} s_D(X_i, \Pi_i)$. Likelihood equivalence means that if $\mathcal{G}_1 \neq \mathcal{G}_2$ are two arbitrary graphs over $\mathcal{X}$ such that both encode the very same conditional independences among variables, then s_D is likelihood equivalent if and only if $s_D(\mathcal{G}_1) = s_D(\mathcal{G}_2)$.

The naive Bayes structure is defined as the network where the class variable X_0 has no parents and every feature (the other variables) has X_0 as sole parent. Figure 1(b) illustrates the situation. In this case, there is nothing to be learned, as the structure is fully defined by the restrictions of naive Bayes. Nevertheless, we can define $\mathcal{G}^*_{\text{naive}}$ as being its (fixed) optimal graph.

The class X_0 has also no parents in a TAN structure, and every feature must have the class as parent (as in the naive Bayes). However, they are allowed to have at most one other feature as parent too. Figure 1(c) illustrates a TAN structure, where X_1 has only X_0 as parent, while both X_2 and X_3 have X_0 and X_1 as parents. By ignoring X_0 and its connections, we have a tree structure, and that is the reason for the name TAN. Based on the BDeu score function, an efficient algorithm for TAN can be devised. Because of the likelihood equivalence of BDeu and the fact that every feature has X_0 as parent, the same score is obtained whether a feature X_i has X_0 and X_j as parent (with $i \neq j$), or X_j has X_0 and X_i, that is,

$$s_D(X_i, \{X_0, X_j\}) + s_D(X_j, \{X_0\}) = s_D(X_j, \{X_0, X_i\}) + s_D(X_i, \{X_0\}) \ . \quad (1)$$

This symmetry allows for a very simple and efficient algorithm [14] that is proven to find the TAN structure which maximizes any score that respects likelihood

[3] In case of many optimal DAGs, then we assume to have no preference and argmax returns one of them.

equivalence, that is, to find

$$\mathcal{G}^*_{\text{TAN}} = \operatorname*{argmax}_{\mathcal{G}\in\boldsymbol{\mathcal{G}}_{\text{TAN}}} s_D(\mathcal{G}) \ , \tag{2}$$

where $\boldsymbol{\mathcal{G}}_{\text{TAN}}$ is the set of all TAN structures with nodes $\mathcal{X}$. The idea is to find the minimum spanning tree in an undirected graph defined over $\mathcal{Y}$ such that the weight of each edge (X_i, X_j) is defined by $w(X_i, X_j) = -(s_D(X_i, \{X_0, X_j\}) - s_D(X_i, \{X_0\}))$. Note that $w(X_i, X_j) = w(X_j, X_i)$. Without loss of generality, let X_1 be the only node without a feature as parent (one could rename the nodes and apply the same reasoning). Now,

$$\begin{aligned}
\max_{\mathcal{G}\in\boldsymbol{\mathcal{G}}_{\text{TAN}}} s_D(\mathcal{G}) &= \max_{\Pi'_i:\forall i>1} \left(\sum_{i=2}^{n} s_D(X_i, \{X_0, X_{\Pi'_i}\}) + s_D(X_1, \{X_0\}) \right) \\
&= s_D(X_1, \{X_0\}) - \min_{\Pi'_i:\forall i>1} \left(- \sum_{i=2}^{n} s_D(X_i, \{X_0, X_{\Pi'_i}\}) \right) \\
&= \sum_{i=1}^{n} s_D(X_i, \{X_0\}) - \min_{\Pi'_i:\forall i>1} \sum_{i=2}^{n} w(X_i, X_{\Pi'_i}) \ . \qquad (3)
\end{aligned}$$

This last minimization is exactly the minimum spanning tree problem, and the argument that minimizes it is the same as the argument that maximizes (2). Because this algorithm has to initialize the $\Theta(n^2)$ edges between every pair of features and then to solve the minimum spanning tree (e.g. using Prim's algorithm), its overall complexity time is $O(n^2)$, if one assumes that the score function is given as an oracle whose queries take time $O(1)$. In fact, because we only consider at most one or two parents for each node (two only if we include the class), the computation of the whole score function can be done in time $O(Nn^2)$ and stored for later use. As a comparison, naive Bayes can be implemented in time $O(Nn)$, while the averaged one-dependence estimator (AODE) [15] needs $\Theta(Nn^2)$, just as TAN does.

2.1 Improving Learning of TANs

A simple extension of this algorithm can already learn a forest of tree-augmented naive Bayes structures. One can simply define the edges of the graph over $\mathcal{Y}$ as in the algorithm for TAN, and then remove those edges (X_i, X_j) such that $s_D(X_i, \{X_0, X_j\}) \leq s_D(X_i, \{X_0\})$, that is, when $w(X_i, X_j) \geq 0$, and then run the minimum spanning tree algorithm over this reduced graph. The optimality of such an idea can be easily proven by the following lemma, which guarantees that we should use only X_0 as parent of X_i every time such choice is better than using $\{X_0, X_j\}$. It is a straightforward generalization of Lemma 1 in [16].

Lemma 1. *Let X_i be a node of $\mathcal{G}$, a candidate DAG where the parent set of X_i is Π'_i. Suppose $\Pi_i \subset \Pi'_i$ is such that $s_D(X_i, \Pi_i) \geq s_D(X_i, \Pi'_i)$, where s_D is a decomposable score function. If Π'_i is the parent set of X_i in an optimal DAG, then the same DAG but having Π_i as parent of X_i is also optimal.*

Using a forest as structure of the classifier is not new, but to the best of our knowledge previous attempts to learn a forest (in this context) did not globally optimize the structure, they instead selected a priori the number of arcs to include in the forest [17].

We want to go even further and allow situations as in Figs. 1(a) and 1(d). The former would automatically disconnect a feature if such feature is not important to predict X_0, that is, if $s_D(X_i, \emptyset) \geq s_D(X_i, \Pi_i)$ for every Π_i. The latter case allows some features to have another feature as parent without the need of having also the class. For this purpose, we define the set of structures named Extended TAN (or ETAN for short), as DAGs such that X_0 has no parents and X_i ($i \neq 0$) is allowed to have the class and at most one feature as parent (but it is not obliged to having any of them), that is, the parent set Π_i is such that $|\Pi_i| \leq 1$, or $|\Pi_i| = 2$ and $\Pi_i \supseteq \{X_0\}$.

$$\mathcal{G}^*_{\text{ETAN}} = \underset{\mathcal{G} \in \boldsymbol{\mathcal{G}}_{\text{ETAN}}}{\text{argmax}}\ s_D(\mathcal{G}) \ . \tag{4}$$

This is clearly a generalization of TAN, of the forest of TANs, and of naive Bayes in the sense that they are all subcases of ETAN. Note that TAN is not a generalization of naive Bayes in this sense, as TAN forces arcs among features even if these arcs were not useful. Because of that, we have the following result. The next section discusses how to efficiently learn ETANs.

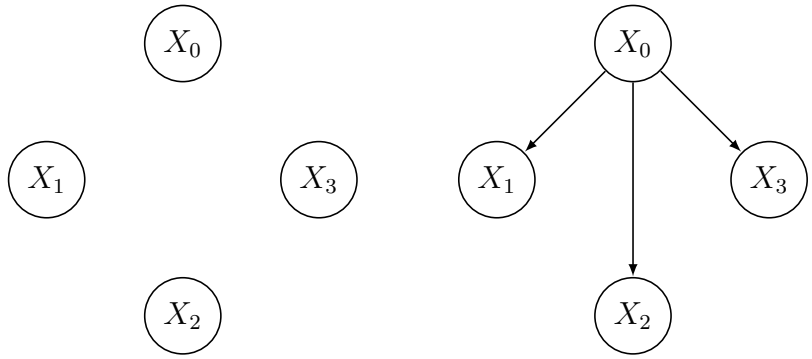
Lemma 2. *The following relations among subsets of DAGs hold.*

$$s_D(\mathcal{G}^*_{ETAN}) \geq s_D(\mathcal{G}^*_{TAN}) \quad \textit{and} \quad s_D(\mathcal{G}^*_{ETAN}) \geq s_D(\mathcal{G}^*_{naive}) \ .$$

3 Learning Extended TANs

The goal of this section is to present an efficient algorithm to find the DAG defined in (4). Unfortunately the undirected version of the minimum spanning tree problem is not enough, because (1) does not hold anymore. To see that, take the example in Fig. 1(d). The arc from X_1 to X_2 cannot be reversed without changing the overall score (unless we connect X_0 to X_2). In other words, every node in a TAN has the class as parent, which makes possible to use the minimum spanning tree algorithm for undirected graphs by realizing that any orientation of the arcs between features will produce the same overall score (as long as the weights of the edges are defined as in the previous section).

Edmonds' algorithm [18] (also attributed to Chu and Liu [19]) for finding minimum spanning arborescence in directed graphs comes to our rescue. Its application is however not immediate, and its implementation is not as simple as the minimum spanning tree algorithm for TAN. Our algorithm to learn ETANs is presented in Algorithm 1. It is composed of a preprocessing of the data to create the arcs of the graph that will be given to Edmonds' algorithm for directed



(a) Possible with ETAN.

(b) Possible with naive or ETAN.

(c) Possible with TAN or ETAN.

(d) Possible only with ETAN.

Fig. 1. Some examples of structures allowed by the different classifiers. The labels indicate which classifier allows them as part of their whole structure.

minimum spanning tree (in fact, we assume that Edmonds' algorithm computes the directed maximum spanning tree, which can be done trivially by negating all weights). `EdmondsContract` and `EdmondsExpand` are the two main steps of that algorithm, and we refer the reader to the description in Zwick's lecture notes [20] or to the work of Tarjan [21] and Camerini et al. [22] or Gabow et al. [23] for further details on the implementation of Edmonds' idea. In fact, we have not been able to find a stable and reliable implementation of such algorithm, so our own implementation of Edmonds' algorithm has been developed based on the description in [20], even though some fixes had to be applied. Because Edmonds' algorithm finds the best spanning tree for a given "root" node (that is, a node that is constrained not to have features as parents), Algorithm 1 loops over the possible roots and extract from Edmonds' the best parent for each node given that fixed root node (line 6), and then stores the best solution over all such possible root nodes. At each loop, Algorithm 3 is called and builds a graph using the information from the result of Edmonds'. Algorithm 1 also loops over a set of

score functions that are given to it. This is used later on to optimize the value of the equivalent sample size in each of the learning steps by giving a list of scores with different prior strengths to the algorithm.

Algorithm 1 ETAN($\mathcal{X}, S$): $\mathcal{X}$ are variables and S is a set of score functions

```
 1: s* ← −∞
 2: for all s_D ∈ S do
 3:     (arcs, classAsParent) ← ArcsCreation(X, s_D)
 4:     EdmondsContract(arcs)
 5:     for all root ∈ X \ {X_0} do
 6:         in ← EdmondsExpand(root)
 7:         G ← buildGraph(X, root, in, classAsParent)
 8:         if s_D(G) > s* then
 9:             G* ← G
10:             s* ← s_D(G)
11: return G*
```

The particular differences with respect to a standard call of Edmonds' algorithm are defined by the methods `ArcsCreation` and `buildGraph`. The method `ArcsCreation` is the algorithm that creates the directed graph that is given as input to Edmonds'. The overall idea is that we must decide whether the class should be a parent of a node or not, and whether it is worth having a feature as a parent. The core argument is again given by Lemma 1. If $s_D(X_i, \{X_0\}) \leq s_D(X_i, \emptyset)$, then we know that no parent is preferable to having the class as a parent for X_i. We store this information in a matrix called `classAsParent` (line 2 of Algorithm 2). Because this information is kept for later reference, we can use from that point onwards the value $\max(s_D(X_i, \emptyset), s_D(X_i, \{X_0\}))$ as the weight of having X_i with only the class as parent (having or not the class as parent cannot create a cycle in the graph, so we can safely use this max value). After that, we loop over every possible arc $X_j \to X_i$ between features, and define its weight as the maximum between having X_0 also as parent of X_i or not, minus the value that we would achieve for X_i if we did not include X_j as its parent (line 8). This is essentially the same idea as done in the algorithm of TAN, but here we must consider both $X_j \to X_i$ and $X_i \to X_j$, as they are not necessarily equivalent (this happens for instance if for one of the two features the class is included in its parent set and for the other it is not, depending on the maximization, so scores defining the weight of each arc direction might be different). After that, we also keep track of whether the class was included in the definition of the weight of the arc or not, storing the information in `classAsParent` for later recall. In case the weight is not positive (line 9), we do not even include this arc in the graph that will be given to Edmonds' (recall we are using the maximization version of Edmonds'), because at this early stage we already know that either no parents for X_i or only the class as parent of X_i (which one of the two is the best can be recalled in `classAsParent`) are better than the score obtained by including

X_j as parent, and using once more the arguments of Lemma 1 and the fact that the class as parent never creates a cycle, we can safely disregard X_j as parent of X_i. All these cases can be seen in Fig. 1 by considering that the variable X_2 shown in the figure is our X_i. There are four options for X_i: no parents (a), only X_0 as parent (b), only X_j as parent (d), and both X_j and X_0 (c). The trick is that Lemma 1 allows us to reduce these four options to two: best between (a) and (b), and best between (c) and (d). After the arcs with positive weight are inserted in a list of arcs that will be given to Edmonds' and `classAsParent` is built, the algorithm ends returning both of them.

Algorithm 2 ArcsCreation($\mathcal{X}, s_D$)

for all $X_i \in \mathcal{X} \setminus \{X_0\}$ **do**
    classAsParent$[X_i] \leftarrow s_D(X_i, \{X_0\}) > s_D(X_i, \emptyset)$
arcs $\leftarrow \emptyset$
for all $X_i \in \mathcal{Y}$ **do**
    for all $X_j \in \mathcal{Y}$ **do**
        twoParents $\leftarrow s_D(X_i, \{X_0, X_j\})$
        onlyFeature $\leftarrow s_D(X_i, \{X_j\})$
        $w \leftarrow \max(\text{twoParents}, \text{onlyFeature}) - \max(s_D(X_i, \emptyset), s_D(X_i, \{X_0\}))$
        if $w > 0$ **then**
            Add $X_j \rightarrow X_i$ with weight w into arcs
            classAsParent$[X_j \rightarrow X_i] \leftarrow$ twoParents $>$ onlyFeature
        else
            classAsParent$[X_j \rightarrow X_i] \leftarrow$ classAsParent$[X_i]$
return (arcs, classAsParent)

Finally, Algorithm 3 is responsible for building back the best graph from the result obtained by Edmonds'. Inside `in` is stored the best parent for each node, and `root` indicates a node that shall have no other feature as parent. The goal is to recover whether the class shall be included as parent of each node, and for that we use the information in `classAsParent`. The algorithm is quite straightforward: for each node that is not the root and has a parent chosen by Edmonds', include it as parent each check if that arc was associated to having or not the class (if it had, include also the class); for each node that has no parent as given by Edmonds' (including the root node), simply check whether it is better to have the class as parent.

Somewhat surprisingly, learning ETANs can be accomplish in time $O(n^2)$ (assuming that the score function is given as an oracle, as discussed before), the same complexity for learning TANs. Algorithm 2 takes $O(n^2)$, because it loops over every pair of nodes and only performs constant time operations inside the loop. `EdmondsContract` can be implemented in time $O(n^2)$ and `EdmondsExpand` in time $O(n)$ [21, 22]. Finally, `buildGraph` takes time $O(n)$ because of its loop over nodes, and the comparison between scores of two ETANs as well as the copy of the structure of an ETANs takes time $O(n)$. So the overall time of the loop

Algorithm 3 buildGraph($\mathcal{X}$, root, in, classAsParent)

```
1: G ← (X, ∅)
2: for all node ∈ X \ {X_0} do
3:     Π_node ← ∅
4:     if node ≠ root and in[node] ≠ null then
5:         Π_node ← Π_node ∪ {in[node]}
6:         if classAsParent[in[node] → node] then
7:             Π_node ← Π_node ∪ {X_0}
8:     else if classAsParent[node] then
9:         Π_node ← Π_node ∪ {X_0}
10: return G
```

in Algorithm 1 takes time $O(n^2)$. Our current implementation can be found at `http://ipg.idsia.ch/software`.

4 Experiments

This section presents results with naive Bayes, TAN and ETAN using 49 data sets from the UCI machine learning repository [24]. Data sets with many different characteristics have been used. Data sets containing continuous variables have been discretized in two bins, using the median as cut-off. Our empirical results are obtained out of 20 runs of 5-fold cross-validation (each run splits the data into folds randomly and in a stratified way), so the learning procedure of each classifier is called 100 times per data set. For learning the classifiers we use the Bayesian Dirichlet equivalent uniform (BDeu) and assume parameter independence and modularity [10]. The BDeu score computes a function based on the posterior probability of the structure $p(\mathcal{G}|D)$. For that purpose, the following function is used:

$$s_D(\mathcal{G}) = \log\left(p(\mathcal{G}) \cdot \int p(D|\mathcal{G},\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}|\mathcal{G}) d\boldsymbol{\theta}\right) \ ,$$

where the logarithm is used to simplify computations, $p(\boldsymbol{\theta}|\mathcal{G})$ is the prior of $\boldsymbol{\theta}$ (vector of parameters of the Bayesian network) for a given graph $\mathcal{G}$, assumed to be a symmetric Dirichlet. BDeu respects likelihood equivalence and its function is decomposable. The only free parameter is the prior strength α (assuming $p(\mathcal{G})$ is uniform), also known as the equivalent sample size (ESS). We make comparisons using different values of α. We implemented ETAN such that $\alpha \in \{1, 2, 10, 20, 50\}$ is chosen according to the value that achieves the highest BDeu for each learning call, that is, we give to ETAN five BDeu score functions with different values of α. Whenever omitted, the default value for α is two.

As previously demonstrated, ETAN always obtains better BDeu score than its competitors. TAN is usually better than naive Bayes, but there is no theoretical guarantee it will always be the case. Table 1 shows the comparisons of BDeu

Table 1. Median value of the BDeu difference between ETAN and competitor (positive means ETAN is better), followed by number of wins, ties and losses of ETAN over competitors, and p-values using the Wilcoxon signed rank test on 49 data sets (one-sided in the direction of the median difference). Names of competitors indicate their equivalent sample size (ESS), and *All* means that ESS has been optimized at each learning call.

Competitor vs. ETAN	BDeu Median	BDeu W/T/L	BDeu p-value
Naive(1)	454	49/0/0	2e-15
Naive(2)	340	48/0/1	3e-15
Naive(10)	276	48/0/1	4e-15
TAN(1)	182	49/0/0	1e-15
TAN(2)	129	46/0/3	8e-13
TAN(10)	23.6	40/1/8	3e-9
TAN(All)	128	46/0/3	2e-8

scores achieved by different classifiers. It presents the median value of the difference between averaged BDeu of the classifiers on the 49 datasets, followed by the number of wins, ties and losses of ETAN against the competitors, and finally the p-value from the Wilcoxon signed rank test (one-sided in the direction of the median value). We note that naive Bayes and TAN might win against ETAN (as it does happen in Tab. 1) because the values of α used by the classifiers in different learning problems are not necessarily the same (the learning method is called 100 for each dataset over different data folds). Nevertheless, the statistical test indicates that the score achieved by ETAN is significantly superior than scores of the other methods.

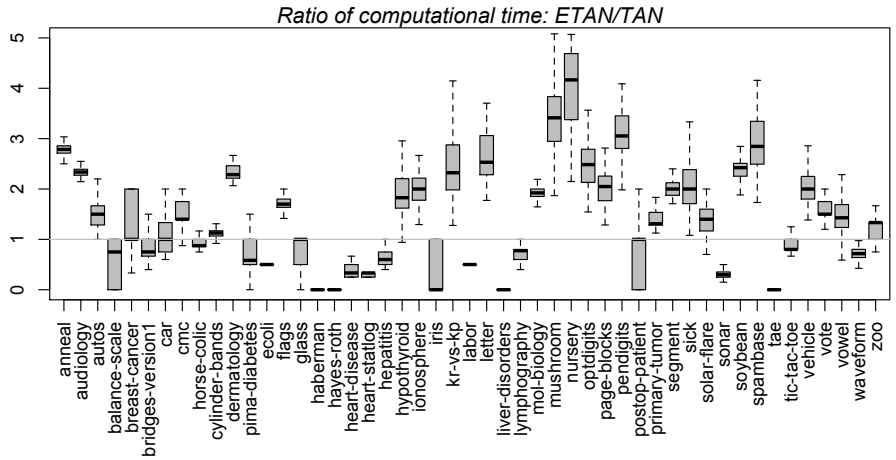


Fig. 2. Computational time to learn the classifier as a ratio ETAN time divided by TAN time, so higher values mean ETAN is slower.

Figure 2 shows the computational time cost to run the learning in the 100 executions per data set for ETAN and TAN (both optimizing α as described before), that is, each of the 20 times 5-fold cross-validation executions. We can see in the graph that learning ETAN has been less than five times slower than learning TAN in all situations (usually less than three times) and has performed better than TAN in a reasonable amount of instances. We recall that both classifiers can be run in quadratic time in the number of features and linear in the sample size, which is asymptotically as efficient as other state-of-the-art classifiers, such as the averaged one-dependence estimator (AODE) [15].

We measure the accuracy of classifiers using zero-one accuracy and log loss. Zero-one accuracy is the number of correctly classified instances divided by the total number of instances, while log loss equals minus the sum (over the testing instances) of the log-probability of the class given the instance's features.

Table 2. Median value of the difference between ETAN and competitor (positive means ETAN is better), followed by number of wins, ties and losses of ETAN over competitors, and p-values using the Wilcoxon signed rank test on 49 data sets (one sided in the direction of the median difference). Names of competitors indicate their equivalent sample size.

Competitor	Zero-one accuracy			Log loss		
vs. ETAN	Median	W/T/L	p-value	Median	W/T/L	p-value
Naive(1)	0.74%	35/3/21	1e-5	0.17	45/0/4	3e-12
Naive(2)	0.64%	36/1/12	4e-5	0.13	46/0/3	5e-12
Naive(10)	1.35%	38/1/10	8e-6	0.12	38/0/11	8e-8
TAN(1)	0.13%	29/1/19	0.022	0.05	43/0/6	2e-8
TAN(2)	0.01%	27/2/20	0.087	0.03	38/1/10	3e-6
TAN(10)	0.01%	28/3/18	0.261	0.01	29/1/19	0.047
TAN(All)	0.06%	29/1/19	0.096	0.0004	26/0/23	0.418
AODE	-0.07%	21/1/27	0.192	-0.005	24/0/25	0.437

Table 2 presents the results of ETAN versus other classifiers. Number of wins, ties and losses of ETAN, as well as p-values from the Wilcoxon signed rank test are displayed, computed over the point results obtained for each of the 49 datasets using cross-validation. We note that ETAN is superior to the other classifiers, except for AODE, in which case the medians are slightly against ETAN and the difference is not significant (pvalues of 0.192 for zero-one accuracy and 0.437 for log loss, in both cases testing whether AODE is superior to ETAN). Median zero-one accuracy of ETAN is superior to TAN(All), although the signed rank test does not show that results are significant at 5% confidence level. The same is true for log loss. In fact, we must emphasize that TAN with optimized choice of α could also be considered as a novel classifier (even if it is only a minor variation of TAN, we are not aware of implementations of TAN that optimize the equivalent sample size).

Figures 3 and 4 show the performance of ETAN versus AODE in terms of zero-one accuracy and log loss, respectively. Each boxplot regards one data set and considers 100 points defined by the runs of cross-validation. In Fig. 3, the values are the zero-one accuracy of ETAN divided by the zero-one accuracy of its competitor in each of the 100 executions. In Fig. 4, it is presented the difference in log loss between AODE and ETAN. In both figures we can see cases where ETAN performed better, as well as cases where AODE did.

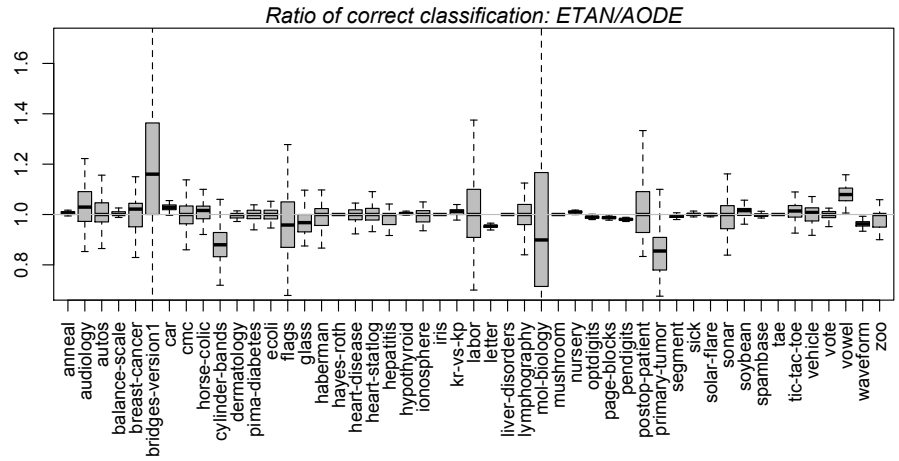


Fig. 3. Comparison of zero-one loss with AODE. Values are ratios of the accuracy of ETAN divided by the competitor, so higher values mean ETAN is better.

5 Conclusions

We presented an extended version of the well-known tree-augmented naive Bayes (TAN) classifier, namely the extended TAN (or ETAN). ETAN does not demand features to be connected to the class, so it has properties of feature selection (when a feature ends up disconnected) and allows features that are important to other features but are not directly depending on the class. We also extend TAN and ETAN to optimize their equivalent sample size at each learning of the structure. We describe a globally optimal algorithm to learn ETANs that is quadratic in the number of variables, that is, it is asymptotically as efficient as the algorithm for TANs, as well as other state-of-the-art classifiers, such as the averaged one-dependence estimator. The class of ETANs can be seen as the (currently) most sophisticated Bayesian networks for which there is a polynomial-time algorithm for learning its structure, as it has been proven that learning with two parents per node (besides the class) is an NP-hard task [25].

Experiments demonstrate that the time complexity of our implementation of ETAN is asymptotically equal to that of TAN, and show that ETAN pro-

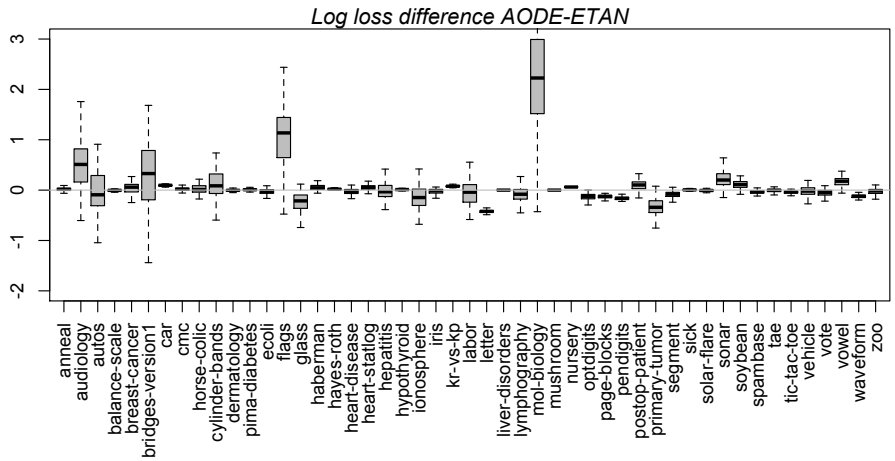


Fig. 4. Comparison of log loss with AODE. Values are differences in the log loss of the competitor minus ETAN, so higher values mean ETAN is better.

vides equal or better fit than TAN and naive Bayes. In our experiments, ETAN achieves better performance in terms of zero-one accuracy and log loss than TAN and naive Bayes under fixed values of the equivalent sample size. If one optimizes the equivalent sample size of TAN, then ETAN has performed in general slightly better than TAN (even though not significant in a statistical test). Future work will investigate further the relation between BDeu and classification accuracy, as well as scenarios where ETAN might be preferable, and will study additional structures beyond ETAN that could be useful in building classifiers.

Acknowledgments

Work partially supported by the Swiss NSF grants Nos. 200021_146606 / 1 and 200020_137680 / 1, and by the Swiss CTI project with Hoosh Technology.

References

1. Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning **29**(2/3) (1997) 103–130
2. Hand, D., Yu, K.: Idiot's Bayes-Not So Stupid After All? International Statistical Review **69**(3) (2001) 385–398
3. Friedman, J.: On bias, variance, 0/1 - loss, and the curse-of-dimensionality. Data Mining and Knowledge Discovery **1** (1997) 55–77
4. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
5. Corani, G., de Campos, C.P.: A tree augmented classifier based on extreme imprecise Dirichlet model. Int. J. Approx. Reasoning **51** (2010) 1053–1068

6. Madden, M.: On the classification performance of TAN and general Bayesian networks. Knowledge-Based Systems **22(7)** (2009) 489–495
7. Corani, G., Antonucci, A., Maua, D., Gabaglio, S.: Trading off speed and accuracy in multilabel classification. In: Proceedings of the 7th European Workshop on Probabilistic Graphical Models. (2014)
8. Buntine, W.: Theory refinement on Bayesian networks. In: Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence. UAI'92, San Francisco, CA, Morgan Kaufmann (1991) 52–60
9. Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning **9** (1992) 309–347
10. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. Machine Learning **20** (1995) 197–243
11. Kontkanen, P., Myllymäki, P., Silander, T., Tirri, H.: On supervised selection of Bayesian networks. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc. (1999) 334–342
12. Elkan, C.: The foundations of cost-sensitive learning. In: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01), Morgan Kaufmann (2001) 973–978
13. Turney, P.D.: Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. Journal of Artificial Intelligence Research **2** (1995) 369–409
14. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory **IT-14 (3)** (1968) 462–467
15. Webb, G.I., Boughton, J.R., Wang, Z.: Not so naive Bayes: Aggregating one-dependence estimators. Machine Learning **58**(1) (2005) 5–24
16. de Campos, C.P., Ji, Q.: Efficient structure learning of Bayesian networks using constraints. Journal of Machine Learning Research **12** (2011) 663–689
17. Lucas, P.J.F.: Restricted Bayesian network structure learning. In: Advances in Bayesian Networks, Studies in Fuzziness and Soft Computing. Volume 146., Berlin, Springer-Verlag (2004) 217–232
18. Edmonds, J.: Optimum branchings. Journal of Research of the National Bureau of Standards B **71B(4)** (1967) 233–240
19. Chu, Y.J., Liu, T.H.: On the shortest arborescence of a directed graph. Science Sinica **14** (1965) 1396–1400
20. Zwick, U.: Lecture notes on Analysis of Algorithms: Directed Minimum Spanning Trees (April 22, 2013)
21. Tarjan, R.E.: Finding optimum branchings. Networks **7** (1977) 25–35
22. Camerini, P.M., Fratta, L., Maffioli, F.: A note on finding optimum branchings. Networks **9** (1979) 309–312
23. Gabow, H.N., Galil, Z., Spencer, T., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**(2) (1986) 109–122
24. Asuncion, A., Newman, D.: UCI machine learning repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html (2007)
25. Dasgupta, S.: Learning polytrees. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence, San Francisco, Morgan Kaufmann (1999) 134–141