bnclassify: Learning Bayesian Network Classifiers

by Bojan Mihaljević, Concha Bielza, and Pedro Larrañaga

Abstract The **bnclassify** package provides state-of-the art algorithms for learning Bayesian network classifiers from data. For structure learning it provides variants of the greedy hill-climbing search, a well-known adaptation of the Chow-Liu algorithm and averaged one-dependence estimators. It provides Bayesian and maximum likelihood parameter estimation, as well as three naive-Bayes-specific methods based on discriminative score optimization and Bayesian model averaging. The implementation is efficient enough to allow for time-consuming discriminative scores on medium-sized data sets. The **bnclassify** package provides utilities for model evaluation, such as cross-validated accuracy and penalized log-likelihood scores, and analysis of the underlying networks, including network plotting via the **Rgraphviz** package. It is extensively tested, with over 200 automated tests that give a code coverage of 94%. Here we present the main functionalities, illustrate them with a number of data sets, and comment on related software.

Introduction

Bayesian network classifiers (Bielza and Larrañaga, 2014; Friedman et al., 1997) are competitive performance classifiers (e.g., Zaidi et al., 2013) with the added benefit of interpretability. Their simplest member, the naive Bayes (NB) (Minsky, 1961), is well-known (Hand and Yu, 2001). More elaborate models exist, taking advantage of the Bayesian network (Pearl, 1988; Koller and Friedman, 2009) formalism for representing complex probability distributions. The tree augmented naive Bayes (Friedman et al., 1997) and the averaged one-dependence estimators (AODE) (Webb et al., 2005) are among the most prominent.

A Bayesian network classifier is simply a Bayesian network applied to classification, that is, to the prediction of the probability $P(c \mid \mathbf{x})$ of some discrete (class) variable C given some features $\mathbf{X}$. The **bnlearn** (Scutari and Ness, 2018; Scutari, 2010) package already provides state-of-the art algorithms for learning Bayesian networks from data. Yet, learning classifiers is specific, as the implicit goal is to estimate $P(c \mid \mathbf{x})$ rather than the joint probability $P(\mathbf{x}, c)$. Thus, specific search algorithms, network scores, parameter estimation, and inference methods have been devised for this setting. In particular, many search algorithms consider a restricted space of structures, such as that of augmented naive Bayes (Friedman et al., 1997) models. Unlike with general Bayesian networks, it makes sense to omit a feature X_i from the model as long as the estimation of $P(c \mid \mathbf{x})$ is no better than that of $P(c \mid \mathbf{x} \setminus x_i)$. Discriminative scores, related to the estimation of $P(c \mid \mathbf{x})$ rather than $P(c, \mathbf{x})$, are used to learn both structure (Keogh and Pazzani, 2002; Grossman and Domingos, 2004; Pernkopf and Bilmes, 2010; Carvalho et al., 2011) and parameters (Zaidi et al., 2013, 2017). Some of the prominent classifiers (Webb et al., 2005) are ensembles of networks, and there are even heuristics applied at inference time, such as the lazy elimination technique (Zheng and Webb, 2006). Many of these methods (e.g., Dash and Cooper, 2002; Zaidi et al., 2013; Keogh and Pazzani, 2002; Pazzani, 1996) are, at best, just available in standalone implementations published alongside the original papers.

The **bnclassify** package implements state-of-the-art algorithms for learning structure and parameters. The implementation is efficient enough to allow for time-consuming discriminative scores on relatively large data sets. It provides utility functions for prediction and inference, model evaluation with network scores and cross-validated estimation of predictive performance, and model analysis, such as querying structure type or graph plotting via the **Rgraphviz** package (Hansen et al., 2017). It integrates with the **caret** (Kuhn et al., 2017; Kuhn, 2008) and **mlr** (Bischl et al., 2017) packages for straightforward use in machine learning pipelines. Currently it supports only discrete variables. The functionalities are illustrated in an introductory vignette, while an additional vignette provides details on the implemented methods. It includes over 200 unit and integration tests that give a code coverage of 94 percent (see `https://codecov.io/github/bmihaljevic/bnclassify?branch=master`).

The rest of this paper is structured as follows. We begin by providing background on Bayesian network classifiers (Section Background) and describing the implemented functionalities (Functionalities). We then illustrate usage with a synthetic data set (Solving a conic linear optimization problem with sdpt3r) and compare the methods' running time, predictive performance and complexity over several data sets (Properties). Finally, we discuss implementation (Implementation), briefly survey related software (Related software), and conclude by outlining future work (Conclusion).

Background

Bayesian network classifiers

A Bayesian network classifier is a Bayesian network used for predicting a discrete class variable C. It assigns $\mathbf{x}$, an observation of n predictor variables (features) $\mathbf{X} = (X_1, \ldots, X_n)$, to the most probable class:

$$c^* = \operatorname*{argmax}_c P(c \mid \mathbf{x}) = \operatorname*{argmax}_c P(\mathbf{x}, c).$$

The classifier factorizes $P(\mathbf{x}, c)$ according to a Bayesian network $\mathcal{B} = \langle \mathcal{G}, \boldsymbol{\theta} \rangle$. $\mathcal{G}$ is a directed acyclic graph with a node for each variable in $(\mathbf{X}, C)$, encoding conditional independencies: a variable X is independent of its nondescendants in $\mathcal{G}$ given the values $\mathbf{pa}(x)$ of its parents. $\mathcal{G}$ thus factorizes the joint into local (conditional) distributions over subsets of variables:

$$P(\mathbf{x}, c) = P(c \mid \mathbf{pa}(c)) \prod_{i=1}^{n} P(x_i \mid \mathbf{pa}(x_i)).$$

Local distributions $P(C \mid \mathbf{pa}(c))$ and $P(X_i \mid \mathbf{pa}(x_i))$ are specified by parameters $\boldsymbol{\theta}_{(C,\mathbf{pa}(c))}$ and $\boldsymbol{\theta}_{(X_i,\mathbf{pa}(x_i))}$, with $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{(C,\mathbf{pa}(c))}, \boldsymbol{\theta}_{(X_1,\mathbf{pa}(x_1))}, \ldots, \boldsymbol{\theta}_{(X_n,\mathbf{pa}(x_n))}\}$. It is common to assume each local distribution has a parametric form, such as the multinomial, for discrete variables, and the Gaussian for real-valued variables.

Learning structure

We learn $\mathcal{B}$ from a data set $\mathcal{D} = \{(\mathbf{x}^1, c^1), \ldots, (\mathbf{x}^N, c^N)\}$ of N observations of $\mathbf{X}$ and C. There are two main approaches to learning the structure $\mathcal{G}$ from $\mathcal{D}$: (a) testing for conditional independence among triplets of sets of variables and (b) searching a space of possible structures in order to optimize a network quality score. Under assumptions such as a limited number of parents per variable, approach (a) can produce the correct network in polynomial time (Cheng et al., 2002; Tsamardinos et al., 2003). On the other hand, finding the optimal structure–even with at most two parents per variable–is NP-hard (Chickering et al., 2004). Thus, heuristic search algorithms, such as greedy hill-climbing, are commonly used (see e.g., Koller and Friedman, 2009). Ways to reduce model complexity, in order to avoid overfitting the training data $\mathcal{D}$, include searching in restricted structure spaces and penalizing dense structures with appropriate scores.

Common scores in structure learning are the penalized log-likelihood scores, such as the Akaike information criterion (AIC) (Akaike, 1974) and Bayesian information criterion (BIC) (Schwarz, 1978). They measure the model's fitting of the empirical distribution $\widehat{P}(c, \mathbf{x})$ adding a penalty term that is a function of structure complexity. They are decomposable with respect to $\mathcal{G}$, allowing for efficient search algorithms. Yet, with limited N and a large n, discriminative scores based on $P(c \mid \mathbf{x})$, such as conditional log-likelihood and classification accuracy, are more suitable to the classification task (Friedman et al., 1997). These, however, are not decomposable according to $\mathcal{G}$. While one can add a complexity penalty to discriminative scores (e.g., Grossman and Domingos, 2004), they are instead often cross-validated to induce preference towards structures that generalize better, making their computation even more time demanding.

For Bayesian network classifiers, a common (see Bielza and Larrañaga, 2014) structure space is that of augmented naive Bayes (Friedman et al., 1997) models (see Figure 1), factorizing $P(\mathbf{X}, C)$ as

$$P(\mathbf{X}, C) = P(C) \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa}(X_i)), \tag{1}$$

with $C \in \mathbf{Pa}(X_i)$ for all X_i and $\mathbf{Pa}(C) = \varnothing$. Models of different complexity arise by extending or shrinking the parent sets $\mathbf{Pa}(X_i)$, ranging from the NB (Minsky, 1961) with $\mathbf{Pa}(X_i) = \{C\}$ for all X_i, to those with a limited-size $\mathbf{Pa}(X_i)$ (Friedman et al., 1997; Sahami, 1996), to those with unbounded $\mathbf{Pa}(X_i)$ (Pernkopf and O'Leary, 2003). While the NB can only represent linearly separable classes (Jaeger, 2003), more complex models are more expressive (Varando et al., 2015). Simpler models, with sparser $\mathbf{Pa}(X_i)$, may perform better with less training data, due to their lower variance, yet worse with more data as the bias due to wrong independence assumptions will tend to dominate the error.

The algorithms that produce the above structures are generally instances of greedy hill-climbing (Keogh and Pazzani, 2002; Sahami, 1996), with arc inclusion and removal as their search operators. Some (e.g., Pazzani, 1996) add node inclusion or removal, thus embedding feature selection (Guyon and Elisseeff, 2003) within structure learning. Alternatives include the adaptation (Friedman et al.,

1997) of the Chow-Liu (Chow and Liu, 1968) algorithm to find the optimal one-dependence estimator (ODE) with respect to decomposable penalized log-likelihood scores in time quadratic in n. Some structures, such as NB or AODE, are fixed and thus require no search.

Learning parameters

Given $\mathcal{G}$, learning $\boldsymbol{\theta}$ in order to best approximate the underlying $P(C, \mathbf{X})$ is straightforward. For discrete variables X_i and $\mathbf{Pa}(X_i)$, Bayesian estimation can be obtained in closed form by assuming a Dirichlet prior over $\boldsymbol{\theta}$. With all Dirichlet hyper-parameters equal to α,

$$\theta_{ijk} = \frac{N_{ijk} + \alpha}{N_{\cdot j \cdot} + r_i \alpha}, \tag{2}$$

where N_{ijk} is the number of instances in $\mathcal{D}$ such that $X_i = k$ and $\mathbf{pa}(x_i) = j$, corresponding to the j-th possible instantiation of $\mathbf{pa}(x_i)$, $N_{\cdot j \cdot}$ is the number of instances in which $\mathbf{pa}(x_i) = j$, while r_i is the cardinality of X_i. $\alpha = 0$ in Equation 2 yields the maximum likelihood estimate of θ_{ijk}. With incomplete data, the parameters of local distributions are no longer independent and we cannot separately maximize the likelihood for each X_i as in Equation 2. Optimizing the likelihood requires a time-consuming algorithm like expectation maximization (Dempster et al., 1977) which only guarantees convergence to a local optimum.

While the NB can separate any two linearly separable classes given the appropriate $\boldsymbol{\theta}$, learning by approximating $P(C, \mathbf{X})$ cannot recover the optimal $\boldsymbol{\theta}$ in some cases (Jaeger, 2003). Several methods (Hall, 2007; Zaidi et al., 2013, 2017) learn a weight $w_i \in [0, 1]$ for each feature and then update $\boldsymbol{\theta}$ as

$$\theta_{ijk}^{weighted} = \frac{(\theta_{ijk})^{w_i}}{\sum_{k=1}^{r_i} (\theta_{ijk})^{w_i}}.$$

A $w_i < 1$ reduces the effect of X_i on the class posterior, with $w_i = 0$ omitting X_i from the model, making weighting more general than feature selection. The weights can be found by maximizing a discriminative score (Zaidi et al., 2013) or computing the usefulness of a feature in a decision tree (Hall, 2007). Mainly applied to naive Bayes models, a generalization for augmented naive Bayes classifiers has been recently developed (Zaidi et al., 2017).

Another parameter estimation method for the naive Bayes is by means of Bayesian model averaging over the 2^n possible naive Bayes structures with up to n features (Dash and Cooper, 2002). It is computed in time linear in n and provides the posterior probability of an arc from C to X_i.

Inference

Computing $P(c \mid \mathbf{x})$ for a fully observed $\mathbf{x}$ means multiplying the corresponding $\boldsymbol{\theta}$. With an incomplete $\mathbf{x}$, however, exact inference requires summing over parameters of the local distributions and is NP-hard in the general case (Cooper, 1990), yet can be tractable with limited-complexity structures. The AODE ensemble computes $P(c \mid \mathbf{x})$ as the average of the $P_i(c \mid \mathbf{x})$ of the n base models. A special case is the lazy elimination (Zheng and Webb, 2006) heuristic which omits x_i from Equation 1 if $P(x_i \mid x_j) = 1$ for some x_j.

Functionalities

The package has four groups of functionalities:

1. Learning network structure and parameters
2. Analyzing the model
3. Evaluating the model
4. Predicting with the model

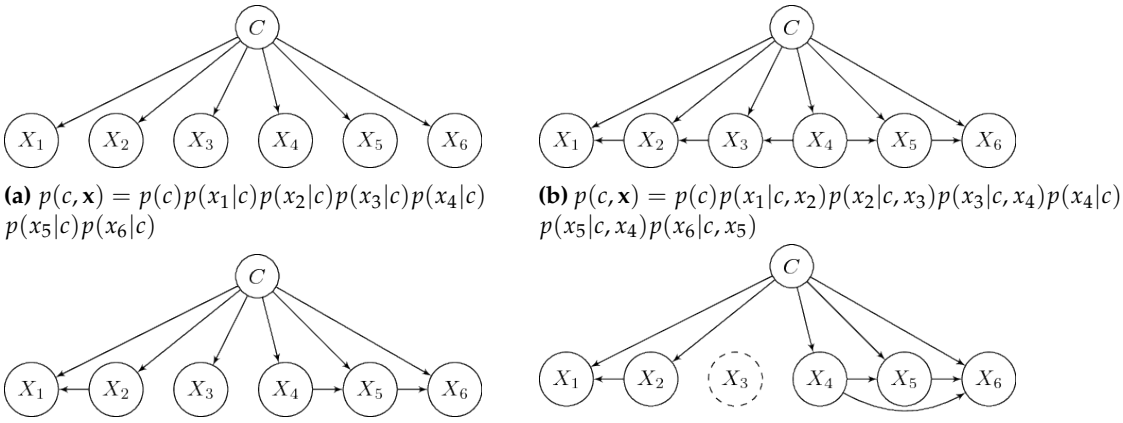
Learning is split into two separate steps, the first step is structure learning and the second, optional, step is parameter learning. The obtained models can be evaluated, used for prediction, or analyzed. The following provides a brief overview of this workflow. For details on some of the underlying methods please see the "methods" vignette.

Structures

The learning algorithms produce the following network structures:

- Naive Bayes (NB) (Figure 1a) (Minsky, 1961)
- One-dependence estimators (ODE)
  - Tree-augmented naive Bayes (TAN) (Figure 1b) (Friedman et al., 1997)
  - Forest-augmented naive Bayes (FAN) (Figure 1c)
- k-dependence Bayesian classifier (k-DB) (Sahami, 1996; Pernkopf and Bilmes, 2010)
- Semi-naive Bayes (SNB)(Figure 1d) (Pazzani, 1996)
- Averaged one-dependence estimators (AODE) (Webb et al., 2005)

Figure 1 shows some of these structures and their factorizations of $P(c, \mathbf{x})$. We use k-DB in the sense meant by Pernkopf and Bilmes (2010) rather than that by Sahami (1996), as we impose no minimum on the number of augmenting arcs. SNB is the only structure whose complexity is not *a priori* bounded: the feature subgraph might be complete in the extreme case.



(a) $p(c, \mathbf{x}) = p(c)p(x_1|c)p(x_2|c)p(x_3|c)p(x_4|c)$ $p(x_5|c)p(x_6|c)$

(b) $p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c, x_3)p(x_3|c, x_4)p(x_4|c)$ $p(x_5|c, x_4)p(x_6|c, x_5)$

(c) $p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c)p(x_3|c)p(x_4|c)$ $p(x_5|c, x_4)p(x_6|c, x_5)$

(d) $p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c)p(x_4|c)$ $p(x_5|c, x_4)p(x_6|c, x_4, x_5)$

Figure 1: Augmented naive Bayes models produced by the **bnclassify** package. (a) NB; (b) TAN (c) FAN (d) SNB. k-DB and AODE not shown. The NB assumes that the features are independent given the class. ODE allows each predictor to depend on at most one other predictor: the TAN is a special case with exactly $n - 1$ augmenting arcs (i.e., inter-feature arcs) while a FAN may have less than $n - 1$. The k-DB allows for up to k parent features per feature X_i, with NB and ODE as its special cases with $k = 0$ and $k = 1$, respectively. The SNB does not restrict the number of parents but requires that connected feature subgraphs be complete (connected, after removing C, subgraphs in (d): $\{X_1, X_2\}$, and $\{X_4, X_5, X_6\}$), also allowing the removal of features (X_3 omitted in (d)). The AODE is not a single structure but an ensemble of n ODE models in which one feature is the parent of all others (a super-parent).

Algorithms

Each structure learning algorithm is implemented by a single R function. Table 1 lists these algorithms along with the corresponding structures that they produce, the scores they can be combined with, and their R functions. Below we provide their abbreviations, references, brief comments, and illustrate function calls.

Fixed structure

We implement two algorithms:

- NB
- AODE

The NB and AODE structures are fixed given the number of variables, and thus no search is required to estimate them from data. For example, we can get a NB structure with

```
n <- nb('class', dataset = car)
```

where `class` is the name of the class variable C and `car` the dataset containing observations of C and $\mathbf{X}$.

Optimal ODEs with decomposable scores

We implement one algorithm:

- Chow-Liu for ODEs (CL-ODE; Friedman et al. (1997))

Maximizing log-likelihood will always produce a TAN while maximizing penalized log-likelihood may produce a FAN since including some arcs can degrade such a score. With incomplete data our implementation does not guarantee the optimal ODE as that would require computing maximum likelihood parameters. The arguments of the `tan_cl()` function are the network score to use and, optionally, the root for features' subgraph:

```
n <- tan_cl('class', car, score = 'AIC', root = 'buying')
```

Greedy hill-climbing with global scores

The **bnclassify** package implements five algorithms:

- Hill-climbing tree augmented naive Bayes (HC-TAN) (Keogh and Pazzani, 2002)
- Hill-climbing super-parent tree augmented naive Bayes (HC-SP-TAN) (Keogh and Pazzani, 2002)
- Backward sequential elimination and joining (BSEJ) (Pazzani, 1996)
- Forward sequential selection and joining (FSSJ) (Pazzani, 1996)
- Hill-climbing k-dependence Bayesian classifier (k-DB)

These algorithms use the cross-validated estimate of predictive accuracy as a score. Only the FSSJ and BSEJ perform feature selection. The arguments of the corresponding functions include the number of cross-validation folds, `k`, and the minimal absolute score improvement, `epsilon`, required for continuing the search:

```
fssj <- fssj('class', car, k = 5, epsilon = 0)
```

Structure	Search algorithm	Score	Feature selection	Function
NB	-	-	-	`nb`
TAN/FAN	CL-ODE	log-lik, AIC, BIC	-	`tan_cl`
TAN	TAN-HC	accuracy	-	`tan_hc`
TAN	TAN-HCSP	accuracy	-	`tan_hcsp`
SNB	FSSJ	accuracy	forward	`fssj`
SNB	BSEJ	accuracy	backward	`bsej`
AODE	-	-	-	`aode`
kDB	kDB	accuracy	-	`kdb`

Table 1: Implemented structure learning algorithms.

Parameters

The **bnclassify** package only handles discrete features. With fully observed data, it estimates the parameters with maximum likelihood or Bayesian estimation, according to Equation 2, with a single α for all local distributions. With incomplete data it uses *available case analysis* and substitutes $N_{\cdot j \cdot}$ in Equation 2 with $N_{ij\cdot} = \sum_{k=1}^{r_i} N_{ijk}$, i.e., with the count of instances in which $\mathbf{Pa}(X_i) = j$ and X_i is observed.

We implement two methods for weighted naive Bayes parameter estimation:

- Weighting attributes to alleviate naive Bayes' independence assumption (WANBIA) (Zaidi et al., 2013)
- Attribute-weighted naive Bayes (AWNB) (Hall, 2007)

We implement one method for estimation by means of Bayesian model averaging over all NB structures with up to n features:

- Model averaged naive Bayes (MANB) (Dash and Cooper, 2002)

It makes little sense to apply WANBIA, MANB, and AWNB to structures other than NB. WANBIA, for example, learns the weights by optimizing the conditional log-likelihood of the NB. Parameter learning is done with the `lp()` function. For example,

```
a <- lp(n, smooth = 1, manb_prior = 0.5)
```

computes Bayesian parameter estimates with $\alpha = 1$ (the `smooth` argument) for all local distributions, and updates them with the MANB estimation obtained with a 0.5 prior probability for each class-to-feature arc.

Utilities

Single-structure-learning functions, as opposed to those that learn an ensemble of structures, return an S3 object of class `"bnc_dag"`. The following functions can be invoked on such objects:

- Plot the network: `plot()`
- Query model type: `is_tan()`, `is_ode()`, `is_nb()`, `is_aode()`, ...
- Query model properties: `narcs()`, `families()`, `features()`, ...
- Convert to a **gRain** object: `as_grain()`

Ensembles are of type `"bnc_aode"` and only `print()` and model type queries can be applied to such objects. Fitting the parameters (by calling `lp()`) of a `"bnc_dag"` produces a `"bnc_bn"` object. In addition to all `"bnc_dag"` functions, the following are meaningful:

- Predict class labels and class posterior probability: `predict()`
- Predict joint distribution: `compute_joint()`
- Network scores: `AIC()`,`BIC()`,`logLik()`,`clogLik()`
- Cross-validated accuracy: `cv()`
- Query model properties: `nparams()`
- Parameter weights: `manb_arc_posterior()`, `weights()`

The above functions for `"bnc_bn"` can also be applied to an ensemble with fitted parameters.

Documentation

This vignette provides an overview of the package and background on the implemented methods. Calling `?bnclassify` gives a more concise overview of the functionalities, with pointers to relevant functions and their documentation. The "usage" vignette presents more detailed usage examples and shows how to combine the functions. The "methods" vignette provides details on the underlying methods and documents implementation specifics, especially where they differ from or are undocumented in the original paper.

Usage example

The available functionalities can be split into four groups:

1. Learning network structure and parameters
2. Analyzing the model
3. Evaluating the model
4. Predicting with the model

We illustrate these functionalities with the synthetic `car` data set with six features. We begin with a simple example for each functionality group and then elaborate on the options in the following sections. We first load the package and the dataset:

```
library(bnclassify)
data(car)
```

Then we learn a naive Bayes structure and its parameters:

```
nb <- nb('class', car)
nb <- lp(nb, car, smooth = 0.01)
```

Then we get the number of arcs in the network:

```
narcs(nb)
```

```
[1] 6
```

Then we get the 10-fold cross-validation estimate of accuracy:

```
cv(nb, car, k = 10)
```

```
[1] 0.8628258
```

Finally, we classify the entire data set:

```
p <- predict(nb, car)
head(p)
```

```
[1] unacc unacc unacc unacc unacc unacc
Levels: unacc acc good vgood
```

Learning

The functions for structure learning, shown in Table 1, correspond to the different algorithms. They all receive the name of the class variable and the data set as their first two arguments, which are then followed by optional arguments. The following runs the CL-ODE algorithm with the AIC score, followed by the FSSJ algorithm to learn another model:

```
ode_cl_aic <- tan_cl('class', car, score = 'aic')
set.seed(3)
fssj <- fssj('class', car, k = 5, epsilon = 0)
```

The `bnc()` function is a shorthand for learning structure and parameters in a single step,

```
ode_cl_aic <- bnc('tan_cl', 'class', car, smooth = 1, dag_args = list(score = 'aic'))
```

where the first argument is the name of the structure learning function while and optional arguments go in `dag_args`.

Analyzing

Printing the model, such as the above `ode_cl_aic` object, provides basic information about it.

```
ode_cl_aic
```

```
  Bayesian network classifier

  class variable:        class
  num. features:   6
  num. arcs:   9
  free parameters:   131
  learning algorithm:    tan_cl
```

While plotting the network is especially useful for small networks, printing the structure in the **deal** (Bottcher and Dethlefsen, 2013) and **bnlearn** format may be more useful for larger ones:

```
ms <- modelstring(ode_cl_aic)
strwrap(ms, width = 60)
```

```
[1] "[class] [buying|class] [doors|class] [persons|class]"
[2] "[maint|buying:class] [safety|persons:class]"
[3] "[lug_boot|safety:class]"
```

We can query the type of structure–`params()` lets us access the conditional probability tables (CPTs), while `features()` lists the features:

```
is_ode(ode_cl_aic)
```

```
[1] TRUE
```

```
params(nb)$buying
```

```
       class
buying         unacc          acc         good        vgood
  low   0.2132243562 0.2317727320 0.6664252607 0.5997847478
  med   0.2214885458 0.2994740131 0.3332850521 0.3999077491
  high  0.2677680077 0.2812467451 0.0001448436 0.0001537515
  vhigh 0.2975190903 0.1875065097 0.0001448436 0.0001537515
```

```
length(features(fssj))
```

```
[1] 5
```

For example, `fssj()` has selected five out of six features.

The `manb_arc_posterior()` function provides the MANB posterior probabilities for arcs from the class to each of the features:

```
manb <- lp(nb, car, smooth = 0.01, manb_prior = 0.5)
round(manb_arc_posterior(manb))
```

```
  buying    maint    doors  persons lug_boot   safety
       1        1        0        1        1        1
```

With the posterior probability of 0% for the arc from `class` to `doors`, and 100% for all others, MANB renders `doors` independent from the class while leaving the other features' parameters unaltered. We can see this by printing out the CPTs:

```
params(manb)$doors
```

```
       class
doors   unacc  acc good vgood
  2      0.25 0.25 0.25  0.25
  3      0.25 0.25 0.25  0.25
  4      0.25 0.25 0.25  0.25
  5more  0.25 0.25 0.25  0.25
```

```
all.equal(params(manb)$buying, params(nb)$buying)
```

```
[1] TRUE
```

For more functions for querying a structure with parameters (`"bnc_bn"`) see `?inspect_bnc_bn`. For a structure without parameters (`"bnc_dag"`), see `?inspect_bnc_dag`.

Evaluating

Several scores can be computed:

```
logLik(ode_cl_aic, car)
```

```
'log Lik.' -13307.59 (df=131)
```

```
AIC(ode_cl_aic, car)
```

```
[1] -13438.59
```

The `cv()` function estimates the predictive accuracy of one or more models with a single run of stratified cross-validation. In the following we assess the above models produced by NB and CL-ODE algorithms:

```
set.seed(0)
cv(list(nb = nb, ode_cl_aic = ode_cl_aic), car, k = 5, dag = TRUE)
```

```
        nb ode_cl_aic
 0.8582303  0.9345913
```

Above, `k` is the desired number of folds, and `dag = TRUE` evaluates structure and parameter learning, while `dag = FALSE` keeps the structure fixed and evaluates just the parameter learning. The output gives 86% and 93% accuracy estimates for NB and CL-ODE, respectively. The **mlr** and **caret** packages provide additional options for evaluating predictive performance, such as different metrics, and **bnclassify** is integrated with both (see the "usage" vignette).

Predicting

As shown above, we can predict class labels with `predict()`. We can also get the class posterior probabilities:

```
pp <- predict(nb, car, prob = TRUE)
# Show class posterior distributions for the first six instances of car
head(pp)
```

```
         unacc          acc         good        vgood
[1,] 1.0000000 2.171346e-10 8.267214e-16 3.536615e-19
[2,] 0.9999937 6.306269e-06 5.203338e-12 5.706038e-19
[3,] 0.9999908 9.211090e-06 5.158884e-12 4.780777e-15
[4,] 1.0000000 3.204714e-10 1.084552e-15 1.015375e-15
[5,] 0.9999907 9.307467e-06 6.826088e-12 1.638219e-15
[6,] 0.9999864 1.359469e-05 6.767760e-12 1.372573e-11
```

Properties

We illustrate the algorithms' running times, resulting structure complexity and predictive performance on the datasets listed in Table 2. We only used complete data sets as time-consuming inference with incomplete data makes cross-validated scores costly for medium-sized or large data sets. The structure and parameter learning methods are listed in the legends of Figure 2, Figure 3, and Figure 4.

N	n	r_c	Dataset
1728	7	4	car
958	10	2	tic-tac-toe
435	17	2	voting
351	35	2	ionosphere
562	36	19	soybean
3196	37	2	kr-vs-kr
3190	61	3	splice

Table 2: Data sets used, from the UCI repository (Lichman, 2013). Incomplete rows have been removed. The number of classes (i.e., distinct class labels) is r_c.

Figure 2 shows that the algorithms with cross-validated scores, followed by WANBIA, are the most time-consuming. Running time is still not prohibitive: TAN-HC ran for 139 seconds on kr-vs-kp and 282 seconds on splice, adding 27 augmenting arcs on the former and 7 on the latter (a added arcs mean a iterations of the search algorithm). Note that their running time is linear in the number of cross-validation folds k; using $\mathsf{k} = 10$ instead of $\mathsf{k} = 5$ would have roughly doubled the time.

CL-ODE tended to produce the most complex structures (see Figure 3), with FSSJ learning complex models on car, soybean and splice, yet simple ones, due to feature selection, on voting and tic-tac-toe. The NB models with alternative parameters, WANBIA and MANB, have as many parameters as the NB, because we are not counting the length-n weights vector, rather just the parameters $\boldsymbol{\theta}$ of the resulting NB (the weights simply produce an alternative parameterization of the NB).

In terms of accuracy, NB and MANB performed comparatively poorly on car, voting, tic-tac-toe, and kr-vs-kp, possibly because of many wrong independence assumptions (see Figure 4). WANBIA may have accounted for some of these violations on voting and kr-vs-kp, as it outperformed NB and MANB on these datasets, showing that a simple model can perform well on them when adequately parameterized. More complex models, such as CL-ODE and AODE, performed better on `car`.

Implementation

With complete data, **bnclassify** implements prediction for augmented naive Bayes models as well as for ensembles of such models. It multiplies the corresponding $\boldsymbol{\theta}$ in logarithmic space, applying the *log-sum-exp* trick before normalizing, to reduce the chance of underflow. On instances with missing entries, it uses the **gRain** package (Højsgaard, 2016, 2012) to perform exact inference, which is noticeably slower. Network plotting is implemented by the **Rgraphviz** package. Some functions are implemented in C++ with **Rcpp** for efficiency. The package is extensively tested, with over 200 unit and integrated tests that give a 94% code coverage.

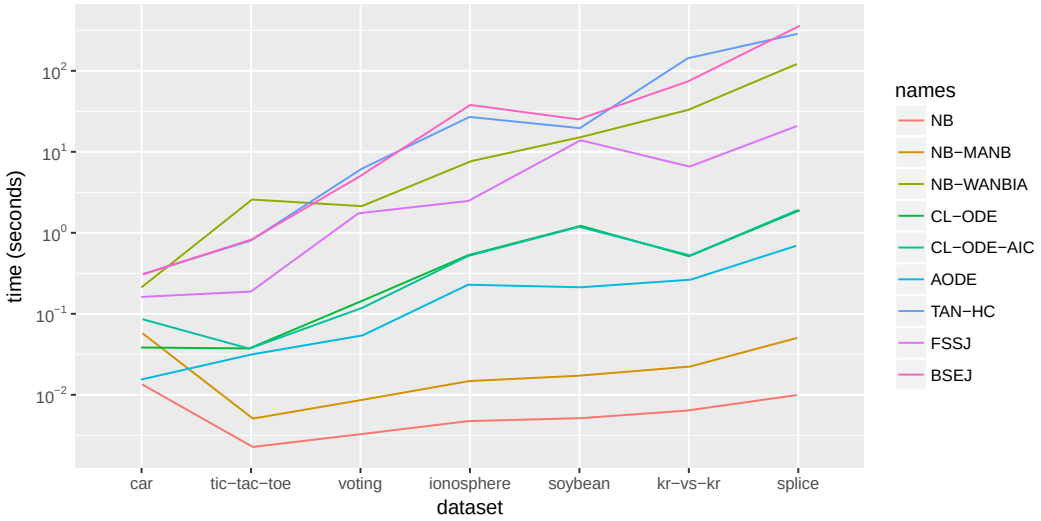


Figure 2: Running times of the algorithms on a Ubuntu 16.04 machine with 8 GB of RAM and a 2.5 GHz processor, on a $\log_{10}$ scale. We used the default options for all algorithms and `k = 5` and `epsilon = 0` for the wrappers. CL-ODE-AIC is CL-ODE with the AIC rather than the log-likelihood score. The lines have been horizontally and vertically jittered to avoid overlap where identical.

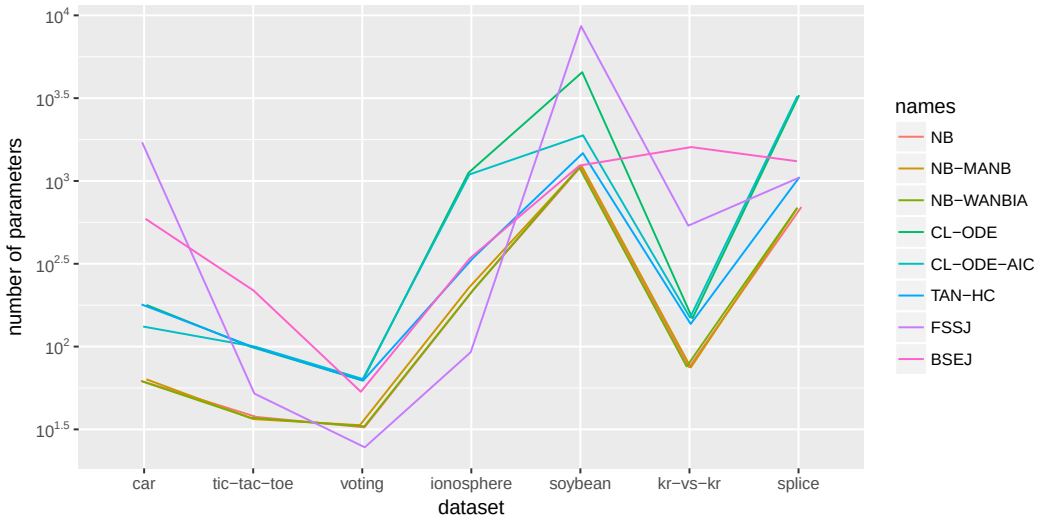


Figure 3: The number of Bayesian network parameters θ of the resulting structures, on a $\log_{10}$ scale. The lines have been horizontally and vertically jittered to avoid overlap where identical.

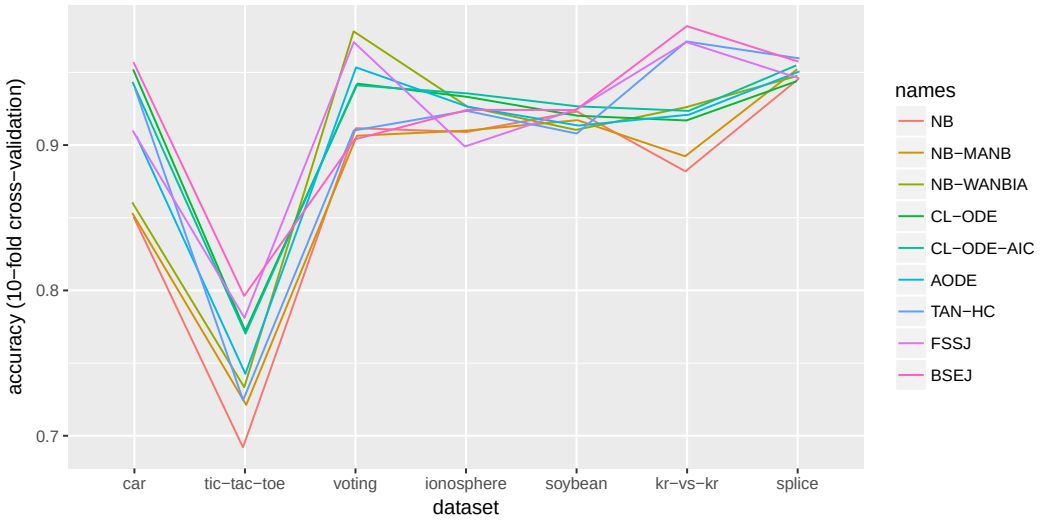


Figure 4: Accuracy of the algorithms estimated with stratified 10-fold cross-validation. The lines have been horizontally and vertically jittered to avoid overlap where identical.

Related software

NB, TAN, and AODE are available in general-purpose tools such as **bnlearn** and Weka. WANBIA (`https://sourceforge.net/projects/rawnaivebayes`) and MANB (`http://www.dbmi.pitt.edu/content/manb`) are only available in stand-alone software, published along with the original publications. We are not aware of available implementations of the remaining methods implemented in **bnclassify**.

The **bnlearn** package implements algorithms for learning general purpose Bayesian networks. Among them, algorithms for Markov blanket learning by testing for independencies, such as IAMB (Tsamardinos and Aliferis, 2003) and GS (Margaritis and Thrun, 2000), can be very useful for classification as they can look for the Markov blanket of the class variable. The **bnlearn** package combines the search algorithms, such as greedy hill-climbing and tabu search (Glover and Laguna, 2013), only with generative scores such as penalized log-likelihood. Among classification models, it implements the discrete NB and CL-ODE. It does not handle incomplete data and provides cross-validation and prediction only for the NB and TAN models, but not for the unrestricted Bayesian networks.

Version 3.8 of Weka (Hall et al., 2009; Bouckaert, 2004) provides variants of the AODE (Webb et al., 2005) as well as the CL-ODE and NB. It implements five additional search algorithms, such as K2 (Cooper and Herskovits, 1992), tabu search, and simulated annealing (Kirkpatrick et al., 1983), combining them only with generative scores. Except for the NB, Weka only handles discrete data and uses simple imputation (replacing with the mode or mean) to handle incomplete data. It provides two constraint-based algorithms, but performs conditional independence tests in an ad-hoc way (Bouckaert, 2004). Weka provides Bayesian model averaging for parameter estimation (Bouckaert, 1995).

The Java library jBNC (`http://jbnc.sourceforge.net/`, version 1.2.2) learns ODE classifiers from Sacha et al. (2002) by optimizing penalized log-likelihood or the cross-validated estimate of accuracy. The CGBayes (version 7.14.14) package (McGeachie et al., 2014) for MATLAB implements conditional Gaussian networks (Lauritzen and Wermuth, 1989). It provides four structure learning algorithms, including a variant of K2 and a greedy hill-climber, all optimizing the marginal likelihood of the data given the network.

Conclusion

The **bnclassify** package implements several state-of-the art algorithms for learning Bayesian network classifiers. It also provides features such as model analysis and evaluation. It is reasonably efficient and can handle large data sets. We hope that **bnclassify** will be useful to practitioners as well as researchers wishing to compare their methods to existing ones.

Future work includes handling real-valued feature via conditional Gaussian models. Straightforward extensions include adding flexibility to the hill-climbing algorithm, such as restarts to avoid local minima.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 785907 (HBP SGA2), the Spanish Ministry of Economy and Competitiveness through the Cajal Blue Brain (C080020-09; the Spanish partner of the EPFL Blue Brain initiative) and TIN2016-79684-P projects, from the Regional Government of Madrid through the S2013/ICE-2845-CASI-CAM-CM project, and from Fundación BBVA grants to Scientific Research Teams in Big Data 2016.

Bibliography

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. [p456]

C. Bielza and P. Larrañaga. Discrete Bayesian network classifiers: A survey. *ACM Computing Surveys*, 47(1), 2014. [p455, 456]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, Z. Jones, G. Casalicchio, and M. Gallo. *mlr: Machine Learning in R*, 2017. URL https://CRAN.R-project.org/package=mlr. R package version 2.11. [p455]

S. G. Bottcher and C. Dethlefsen. *deal: Learning Bayesian Networks with Mixed Variables*, 2013. URL https://CRAN.R-project.org/package=deal. R package version 1.2-37. [p461]

R. Bouckaert. Bayesian network classifiers in Weka. Technical Report 14/2004, Department of Computer Science, University of Waikato, 2004. [p465]

R. R. Bouckaert. *Bayesian Belief Networks: From Construction to Inference*. PhD thesis, Universiteit Utrecht, 1995. [p465]

A. M. Carvalho, T. Roos, A. L. Oliveira, and P. Myllymäki. Discriminative learning of Bayesian networks via factorized conditional log-likelihood. *Journal of Machine Learning Research*, 12:2181–2210, 2011. [p455]

J. Cheng, R. Greiner, J. Kelly, D. A. Bell, and W. Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137:43–90, 2002. [p456]

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004. [p456]

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968. [p457]

G. F. Cooper. The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. *Artificial Intelligence*, 42(2-3):393–405, 1990. [p457]

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992. [p465]

D. Dash and G. F. Cooper. Exact model averaging with naive Bayesian classifiers. In *19th International Conference on Machine Learning (ICML-2002)*, pages 91–98, 2002. [p455, 457, 459]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. [p457]

N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29: 131–163, 1997. [p455, 456, 458, 459]

F. Glover and M. Laguna. Tabu Search. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 3261–3362. Springer-Verlag, New York, NY, 2013. [p465]

D. Grossman and P. Domingos. Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 361–368, 2004. [p455, 456]

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. [p456]

M. Hall. A decision tree-based attribute weighting filter for naive Bayes. *Knowledge-Based Systems*, 20 (2):120–126, 2007. [p457, 459]

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. [p465]

D. J. Hand and K. Yu. Idiot's Bayes - Not so Stupid after All? *International Statistical Review*, 69(3): 385–398, 2001. [p455]

K. D. Hansen, J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, and D. Sarkar. *Rgraphviz: Provides Plotting Capabilities for R Graph Objects*, 2017. URL https://doi.org/10.18129/B9.bioc.Rgraphviz. R package version 2.20.0. [p455]

S. Højsgaard. Graphical independence networks with the gRain package for R. *Journal of Statistical Software*, 46(10):1–26, 2012. [p463]

S. Højsgaard. *gRain: Graphical Independence Networks*, 2016. URL https://CRAN.R-project.org/package=gRain. R package version 1.3-0. [p463]

M. Jaeger. Probabilistic classifiers and the concept they recognize. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, pages 266–273, 2003. [p456, 457]

E. J. Keogh and M. J. Pazzani. Learning the structure of augmented Bayesian classifiers. *International Journal on Artificial Intelligence Tools*, 11(4):587–601, 2002. [p455, 456, 459]

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983. [p465]

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, Cambridge, MA, USA, 2009. [p455, 456]

M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5): 1–26, 2008. [p455]

M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, and T. Hunt. *caret: Classification and Regression Training*, 2017. URL https://CRAN.R-project.org/package=caret. R package version 6.0-78. [p455]

S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17(1):31–57, 1989. [p465]

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. [p463]

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems 12*, pages 505–511. MIT Press, 2000. [p465]

M. J. McGeachie, H.-H. Chang, and S. T. Weiss. CGBayesNets: Conditional Gaussian Bayesian network learning and inference with mixed discrete and continuous data. *PLoS Computational Biology*, 10(6): e1003676, 2014. [p465]

M. Minsky. Steps toward artificial intelligence. *Transactions on Institute of Radio Engineers*, 49:8–30, 1961. [p455, 456, 458]

M. Pazzani. Constructive induction of Cartesian product attributes. In *Proceedings of the Information, Statistics and Induction in Science Conference*, pages 66–77, 1996. [p455, 456, 458, 459]

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, CA, USA, 1988. [p455]

F. Pernkopf and J. A. Bilmes. Efficient heuristics for discriminative structure learning of Bayesian network classifiers. *Journal of Machine Learning Research*, 11:2323–2360, 2010. [p455, 458]

F. Pernkopf and P. O'Leary. Floating search algorithm for structure learning of Bayesian network classifiers. *Pattern Recognition Letters*, 24(15):2839–2848, 2003. [p456]

J. P. Sacha, L. S. Goodenday, and K. J. Cios. Bayesian learning for cardiac spect image interpretation. *Artificial Intelligence in Medicine*, 26(1):109–143, 2002. [p465]

M. Sahami. Learning limited dependence Bayesian classifiers. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-1996)*, volume 96, pages 335–338, 1996. [p456, 458]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. [p456]

M. Scutari. Learning Bayesian networks with the `bnlearn` `R` package. *Journal of Statistical Software*, 35 (3):1–22, 2010. [p455]

M. Scutari and R. Ness. *Bnlearn: Bayesian Network Structure Learning, Parameter Learning and Inference*, 2018. URL `https://CRAN.R-project.org/package=bnlearn`. R package version 4.3. [p455]

I. Tsamardinos and C. F. Aliferis. Towards principled feature selection: Relevancy, filters and wrappers. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*. Morgan Kaufmann Publishers: Key West, FL, USA, 2003. [p465]

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Algorithms for large scale Markov blanket discovery. In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2003)*, pages 376–381. AAAI Press, 2003. [p456]

G. Varando, C. Bielza, and P. Larrañaga. Decision boundary for discrete Bayesian network classifiers. *Journal of Machine Learning Research*, 16:2725–2749, 2015. [p456]

G. I. Webb, J. R. Boughton, and Z. Wang. Not so Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning*, 58(1):5–24, 2005. [p455, 458, 465]

N. A. Zaidi, J. Cerquides, M. J. Carman, and G. I. Webb. Alleviating naive Bayes attribute independence assumption by attribute weighting. *Journal of Machine Learning Research*, 14:1947–1988, 2013. [p455, 457, 459]

N. A. Zaidi, G. I. Webb, M. J. Carman, F. Petitjean, W. Buntine, M. Hynes, and H. De Sterck. Efficient Parameter Learning of Bayesian Network Classifiers. *Machine Learning*, 106(9):1289–1329, 2017. [p455, 457]

F. Zheng and G. I. Webb. Efficient lazy elimination for averaged one-dependence estimators. In *Proceedings of the 23rd International Conference on Machine Learning*, volume 148, pages 1113–1120. ACM, 2006. [p455, 457]

Bojan Mihaljević
Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid
Boadilla del Monte, 28660, Spain
ORCiD: 0000-0002-1656-6135
`bmihaljevic@fi.upm.es`

Concha Bielza
Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid
Boadilla del Monte, 28660, Spain
ORCiD: 0000-0001-7109-2668
`mcbielza@fi.upm.es`

Pedro Larrañaga
Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid
Boadilla del Monte, 28660, Spain
ORCiD: 0000-0003-0652-9872
`pedro.larranaga@fi.upm.es`