`bnclassify` usage

Bojan Mihaljević, Concha Bielza, Pedro Larrañaga

2020-03-12

Abstract

This vignette gives detailed usage examples and shows how to combine the functions.

Contents

1 Introduction

The `bnclassify` package implements state-of-the-art algorithms for learning discrete Bayesian network classifiers from data, as well as functions for using these classifiers for prediction, assessing their predictive performance, and inspecting and analyzing their properties. This vignette gives detailed usage examples and shows how to combine the functions. Other resources provide additional information:

- `vignette("overview", package="bnclassify")` provides an overview of the package and background on the implemented methods.
- `?bnclassify` provides a concise overview of the functionalities, with pointers to relevant functions and their documentation.
- `vignette("methods", package="bnclassify")` provides details on the underlying methods and documents implementation specifics, especially where they differ from or are undocumented in the original paper.

2 Data

Throughout the vignette we will use the car evaluation data set. It has six discrete features, describing car properties such as buying price or the number of doors, and 1728 instances assigned to four different classes (unacc, acc, good, vgood). See `?car` for more details.

```
library(bnclassify)
data(car)
dim(car)
#> [1] 1728    7
head(car)
#>   buying maint doors persons lug_boot safety class
#> 1  vhigh vhigh     2       2    small    low unacc
#> 2  vhigh vhigh     2       2    small    med unacc
#> 3  vhigh vhigh     2       2    small   high unacc
#> 4  vhigh vhigh     2       2      med    low unacc
#> 5  vhigh vhigh     2       2      med    med unacc
#> 6  vhigh vhigh     2       2      med   high unacc
```

3 Workflow

Using `bnclassify` generally consists of four steps:

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

In between those steps, you may also want to inspect the model's properties.

Below is an example of the four steps done in four lines.

```
nb <- nb('class', car) # Learn a naive Bayes structure
nb <- lp(nb, car, smooth = 1) # Learn parameters
cv(nb, car, k = 10) # 10-fold Cross-validation estimate of accuracy
#> [1] 0.8576045
head(predict(nb, car)) # Classify the entire data set
#> [1] unacc unacc unacc unacc unacc unacc
#> Levels: unacc acc good vgood
```

While there are multiple alternatives to `nb` for the first step, you are most likely to use `lp`, `cv`, and `predict` for steps 2-4. We will elaborate on all four steps throughout the rest of the vignette.

4 Network structure

4.1 Learning

`bnlassify` provides one function per each structure learning algorithm that it implements. Grouped according to algorithm type (see `vignette("bnclassify-technical")`), these are:

Naive Bayes:

- `nb`

CL ODE:

- `tan_cl`

Greedy wrapper:

- `tan_hc`
- `tan_hcsp`
- `fssj`
- `bsej`

They all receive the name of the class variable and the data set as their first two arguments, followed by optional arguments.

The following learns three different structures with three different algorithms.

```
# Naive Bayes
nb <- nb('class', car)
# ODE Chow-Liu with AIC score (penalized log-likelihood)
ode_cl_aic <- tan_cl('class', car, score = 'aic')
# Semi-naive Bayes with forward sequential selection and joining (FSSJ) and
# 5-fold cross-validation
fssj <- fssj('class', car, k = 5, epsilon = 0)
```

For details on the learning algorithms, see the corresponding functions (e.g., `?tan_cl`) and `vignette("bnclassify-technical")`.

4.2 Analyzing

The above `nb`, `ode_cl_aic`, and `fssj` are objects of class `bnc_dag`. There are a number of functions that you can perform on such objects.

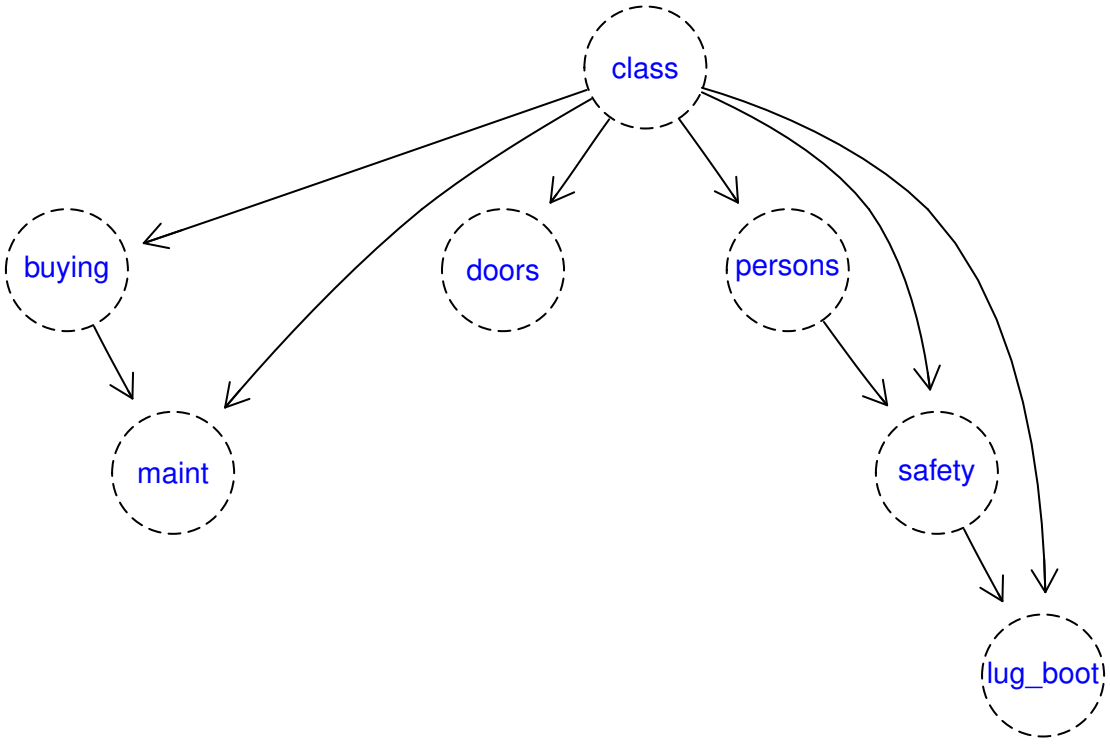
Printing the object to console outputs basic information on structure:

```
ode_cl_aic
#>
#>   Bayesian network classifier (only structure, no parameters)
#>
#>   class variable:        class
#>   num. features:   6
#>   num. arcs:   9
#>   learning algorithm:    tan_cl
```

The above tells that the `ode_cl_aic` object is a network structure without any parameters, the name of the class variables is "class", it has six feature nodes and nine arcs, and it was learned with the `tan_cl` function.

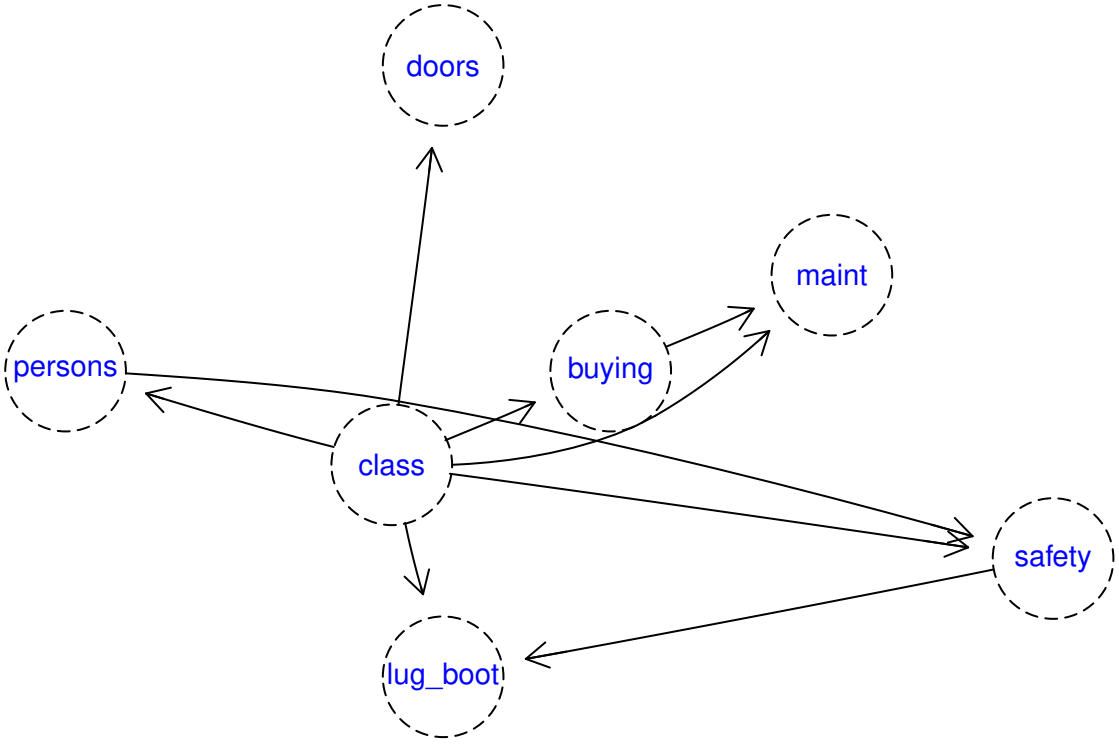
Plotting network structure can reveal probabilistic relationships among the variables:

```
plot(ode_cl_aic)
```



If the network is not displaying properly, e.g., with node names overlapping in large networks, you may try different layout types and font sizes (see `?plot.bnc_dag`).

```
plot(ode_cl_aic, layoutType = 'twopi', fontsize = 15)
```



An alternative to plotting, useful when the graph is large, is to query for the families that compose the structure (a family of a node is itself plus its parents in the graph).

```
families(ode_cl_aic)
#> $buying
#> [1] "buying" "class"
#>
#> $maint
#> [1] "maint"  "buying" "class"
#>
#> $doors
#> [1] "doors" "class"
#>
#> $persons
#> [1] "persons" "class"
#>
#> $lug_boot
#> [1] "lug_boot" "safety"   "class"
#>
#> $safety
#> [1] "safety"  "persons" "class"
#>
#> $class
#> [1] "class"
```

`narcs` gives the number of arcs in a structure.

```
narcs(nb)
#> [1] 6
```

Functions such as `is_ode`, `is_nb`, or `id_semi` query the type of structure. For example:

```
is_ode(ode_cl_aic)
#> [1] TRUE
is_semi_naive(ode_cl_aic)
#> [1] FALSE
```

For more functions to query a network structure, see `?inspect_bnc_dag`.

5 Network parameters

5.1 Learning

`bnclassify` provides three parameter estimation methods, all implemented with the `lp` function.

- Bayesian and maximum likelihood estimation
- AWNB
- MANB

`lp` which takes the network structure and the dataset from which to learn parameters as its first two arguments.

To get Bayesian parameter estimates assuming a Dirichlet prior, provide a positive `smooth` argument to `lp`.

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

For AWNB or MANB parameter estimation, provide the appropriate arguments to `lp`, in addition to `smooth`.

```
awnb <- lp(nb, car, smooth = 0.01, awnb_trees = 10, awnb_bootstrap = 0.5)
manb <- lp(nb, car, smooth = 0.01, manb_prior = 0.5)
```

The `bnc` function is shorthand for learning both structure and parameters in a single step. Provide the name of the structure learning algorithm, as a character, and its optional arguments in `dag_args`.

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

5.2 Analyzing

`lp` and `bnc` return objects of class `bnc_bn`, which are fully specified Bayesian network classifiers (i.e., with both structure and parameters).

Printing the `ode_cl_aic` object now also shows how many free parameters there are in the model (131).

```
ode_cl_aic
#>
#>   Bayesian network classifier
#>
#>   class variable:        class
#>   num. features:   6
#>   num. arcs:   9
#>   free parameters:   131
#>   learning algorithm:    tan_cl
```

`params` lets you access the conditional probability tables (CPTs). For example, the CPT of the `buying` feature in `nb` is:

```
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  0.26776800777 0.2812467451 0.0001448436 0.0001537515
#>   vhigh 0.2975190903 0.1875065097 0.0001448436 0.0001537515
```

`nparams` gives the number of parameters of the classifier.

```
nparams(nb)
#> [1] 63
```

For more functions for querying a `bnc_bn` object, see `?inspect_bnc_bn`

5.3 Interface to `bnlearn`, `gRain`, and `graph`

You can convert a `bnc_bn` object to `bnlearn` (Scutari 2010), `gRain` (Højsgaard 2012) and `graph` (Gentleman et al. 2015) objects to leverage functionalities from those packages, such as Bayesian network querying or inference.

Use

- `as_graphNEL` for `graph`
- `as_grain` for `gRain`

For `bnlearn`, first convert to `gRain` and then convert the `gRain` object to a `bnlearn` one (see `bnlearn` docs for how to do this).

The following uses `gRain` to ge the marginal probability of the `buying` feature: (NOTE: not currently working due to recent changes in the gRain package)

```
# a <- lp(nb('class', car), car, smooth = 1)
# g <- as_grain(a)
# gRain::querygrain.grain(g)$buying
```

6 Selecting features

Some structure and parameter learning methods perform feature selection:

- `fssj` and `bsej` : embedded wrapper
- MANB: Bayesian model averaging
- AWNB: weighting

`fssj` and `bsej` perform feature selection while learning structure. On the car evaluation data they both select all features.

```
length(features(fssj))
#> [1] 5
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
bsej <- bsej('class', car, k = 5, epsilon = 0)
length(features(bsej))
#> [1] 6
```

MANB has computed zero posterior probability for the arc from `class` to `doors` and 100% probability for arcs to the other features.

```
manb_arc_posterior(manb)
#>       buying        maint        doors      persons     lug_boot       safety
#> 1.000000e+00 1.000000e+00 3.937961e-20 1.000000e+00 9.980275e-01 1.000000e+00
```

This means that it has effectively omitted `doors` from the model, rendering it independent from the class.

```
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
```

It has left the other features' parameters unaltered.

```
all.equal(params(manb)$buying, params(nb)$buying)
#> [1] TRUE
```

The AWNB method has decreased the effect of each feature on the class posterior, especially `doors`, `lug_boot`, and `maint`, also modifying their local distributions towards independence from the class.

```
awnb_weights(awnb)
#>    buying     maint     doors   persons  lug_boot    safety
#> 0.5773503 0.5000000 0.3931064 0.8535534 0.4355240 0.8535534
```

6.1 External feature selection

You can use `R` packages such as `mlr` (Bischl et al. 2015) or `caret` (Kuhn 2008) to select features prior to learning a classifier with `bnclassify`. See Section 10 for how to do it with `mlr`.

7 Evaluating

7.1 Network scores

The are three functions for computing penalized log-likelihood network scores of `bnc_bn` objects.

- `logLik`
- `AIC`
- `BIC`

In addition to the model, provide them the dataset on which to compute the score.

```
logLik(ode_cl_aic, car)
#> 'log Lik.' -13307.59 (df=131)
AIC(ode_cl_aic, car)
#> [1] -13438.59
BIC(ode_cl_aic, car)
#> [1] -13795.87
```

7.2 Predictive accuracy

`accuracy` lets you compute the classifier's predictive accuracy on a given data set. You need to provide the vectors of predicted and true labels.

```
p <- predict(nb, car)
accuracy(p, car$class)
#> [1] 0.8738426
```

`cv` estimates predictive accuracy with stratified cross-validation. Indicate the desired number of folds with `k`.

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
cv(ode_cl_aic, car, k = 10)
#> [1] 0.9386636
```

Each `bnc_bn` object records the structure and parameter learning methods that were used to produce it. `cv` just reruns these methods. Hence, the above is the accuracy estimate for `tan_cl` with the AIC score and Bayesian parameter estimation with `smooth = 0.01`.

To keep the structure fixed and evaluate just the parameter learning method, set `dag = FALSE`:

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
```

```
cv(ode_cl_aic, car, k = 10, dag = FALSE)
#> [1] 0.9386636
```

To get the accuracy for each of the folds, instead of the mean accuracy, set `mean = FALSE`.

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
cv(ode_cl_aic, car, k = 10, dag = FALSE, mean = FALSE)
#>          [,1]
#> 1   0.9252874
#> 2   0.9248555
#> 3   0.9534884
#> 4   0.9651163
#> 5   0.9479769
#> 6   0.9479769
#> 7   0.9302326
#> 8   0.9127907
#> 9   0.9306358
#> 10 0.9482759
```

Finally, to cross-validate multiple classifiers at once pass a list of `bnc_bn` objects to `cv`.

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
accu <- cv(list(nb = nb, ode_cl_aic = ode_cl_aic), car, k = 5, dag = TRUE)
accu
#>          nb ode_cl_aic
#>  0.8582303  0.9345913
```

7.3 More

General-purpose machine learning packages such as `mlr` or `caret` provide additional options for evaluating a model, including bootstrap resampling and performance measures such as the area under the curve. See Section 10 for how that could be done with `mlr`.

8 Predicting

We can use a `bnc_bn` object to classify data instances, with `predict`.

Here we use the naive Bayes to predict the class for our entire data set.

```
p <- predict(nb, car)
# We use head() to print the first elements of vector p
head(p)
#> [1] unacc unacc unacc unacc unacc unacc
#> Levels: unacc acc good vgood
```

You can also get the class posterior probabilities.

```
pp <- predict(nb, car, prob = TRUE)
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
```

9 Miscellaneous

You can compute the (conditional) mutual information between two variables with `cmi`. Mutual information of `maint` and `buying`:

```
cmi('maint', 'buying', car)
#> [1] 0
```

Mutual information of `maint` and `buying` conditioned to `class`:

```
cmi('maint', 'buying', car, 'class')
#> [1] 0.07199921
```

10 Complementing `bnclassify` with `mlr`

General-purpose machine learning packages, such as `mlr` and `caret`, provide many options for feature selection and model evaluation. For example, the provide resampling methods other than cross-validation and performance measures other than accuracy. Here we use `mlr` to:

1. Perform and evaluate wrapper feature selection using `tan_cl`
2. Estimate the accuracy of `tan_cl` and random forest

To use a `bnc_bn` object with `mlr`, call the `as_mlr` function.

```
library(mlr)
ode_cl_aic_mlr <- as_mlr(ode_cl_aic, dag = TRUE, id = "ode_cl_aic")
```

The obtained `ode_cl_aic_mlr` behaves like any other classifier supported by `mlr`.

10.1 Wrapper feature selection

Set up sequential forward search with 2-fold cross validation and `ode_cl_aic_mlr` as the classifier.

```
# 5-fold cross-validation
rdesc = makeResampleDesc("CV", iters = 2)
# sequential floating forward search
ctrl = makeFeatSelControlSequential(method = "sfs", alpha = 0)
```

```
# Wrap ode_cl_aic_mlr with feature selection
ode_cl_aic_mlr_fs = makeFeatSelWrapper(ode_cl_aic_mlr, resampling = rdesc,
                                       control = ctrl, show.info = FALSE)
t <- makeClassifTask(id = "car", data = car,
        target = 'class', fixup.data = "no", check.data = FALSE)
```

Select features:

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
# Select features
mod <- train(ode_cl_aic_mlr_fs, task = t)
sfeats <- getFeatSelResult(mod)
sfeats
```

`mlr` makes it easy to evaluate the predictive performance of the combination of feature selection plus classifier learning. The following estimates accuracy with 2-fold cross-validation:

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
r = resample(learner = ode_cl_aic_mlr_fs, task = t,
             resampling = rdesc, show.info = FALSE, measure = mlr::acc)
```

10.2 Comparing to random forest

With `mlr` you can compare the predictive performance of `bnclassify` models to those of different classification paradigms, such as random forests.

```
rf <- makeLearner("classif.randomForest", id = "rf")
classifiers <- list(ode_cl_aic_mlr, rf)
suppressWarnings(RNGversion("3.5.0"))
set.seed(0)
benchmark(classifiers, t, rdesc, show.info = FALSE, measures = mlr::acc)
```

References

Bischl, Bernd, Michel Lang, Jakob Richter, Jakob Bossek, Leonard Judt, Tobias Kuehn, Erich Studerus, and Lars Kotthoff. 2015. *Mlr: Machine Learning in R*. http://CRAN.R-project.org/package=mlr.

Gentleman, R., Elizabeth Whalen, W. Huber, and S. Falcon. 2015. *Graph: A Package to Handle Graph Data Structures*.

Højsgaard, Søren. 2012. "Graphical Independence Networks with the `gRain` Package for `R`." *Journal of Statistical Software* 46 (10): 1–26.

Kuhn, Max. 2008. "Building Predictive Models in `R` Using the `caret` Package." *Journal of Statistical Software* 28 (5). American Statistical Association: 1–26.

Scutari, Marco. 2010. "Learning Bayesian Networks with the `bnlearn` `R` Package." *Journal of Statistical Software* 35 (3): 1–22.