The Bayesian Network Explorer package

Watkins W.S.[1], Hernandez, E.J.[1], and Tristani-Firouzi, M.[1]

[1]Department of Human Genetics, University of Utah, Salt Lake City, Utah, 84105, USA.

July 11, 2023

Overview

The Bayesian Network Explorer (BNE) package is a program to simplify the creation, analysis, and interpretation of Bayesian networks. Input data is read from a standard table of observations and variables. Bayesian networks are learned directly from the data using exact or approximate methods. Conditional probabilities are created and propagated through the network.

The main purpose of this package is to allow a user to select a target variable and calculate conditional probability for the target given one or more conditional variables. The relationship among all input variables is learned from the data and is visualized with a directed acyclic graph (DAG) or an undirected network. The target variable can be analyzed alone or conditionally in any combination of variables in the network. If variables are binary, all possible combinations of a target-state and all conditional variable-states can be traversed for small exact networks. The absolute and relative risks for any target outcome with any combination of conditional variables can be calculated. All combinations of multi-state and discretized continuous target variables with all combinations of user-selected conditional variables can also be traversed. Markov blankets for target variables are indicated graphically in the DAGs to allow the use to select the most relevant variables for a specific hypotheses.

The current package wraps the bnstruct and gRain packages from R. The bnstruct package is used to implement several network learning methods to learn the network structure of the data. It is expected that future versions of this package will incorporate structure learning algorithms in julia.

Requirements

The user must have Julia (v1.9+) installed. Additionally, a fully working version of R (v4.3+) is recommended. A GKS-term, such as XQuartz (MacOS) or XGKS (linux), must also be installed to visualize graphs. This package has been developed and tested with Julia 1.9.1 and R 4.3.1 in an MacOS-Ventura (Intel64-emulated) environment.

Installation

First, install R on your machine. Install Bioconductor. Use Bioconductor to install the R packages and their dependencies using Bioconductor.install()

- gRbase
- RBGM
- graph

- qgraph
- Rgraphviz
- bnstruct
- gRain

Make sure that you can use bnstruct and gRain packages on the R command line or within R-studio before installing the BayesNetExplorer package.

Install the julia software package from https://julialang.org/downloads/ and make sure you can start the julia REPL in a terminal. Next, add the DataFrames package to you julia installation.

julia]
(@v1.9) pkg> add DataFrames

Now, install the BayesNetExplorer (BNE) package from github. The BNE package is currently unregistered, therefore please use the developer's github account to obtain this package.

(@v1.9) pkg> add https://github.com/ScottWatkins/BayesNetExplorer

Type the delete key to get back to the julia prompt and load the packages for use.
julia> using DataFrames
julia> using BayesNetExplorer

1 Preparing data sets for the BayesNetExplorer

The following section presents the main functions of the BNE package. The functions are presented in the typical order of use. A discussion section provides additional details. Worked examples, using common data sets for 1) transportation preferences for 500 commuters, and 2) diabetes risk, are presented. The specific examples are minimal and users are encouraged to try the many additional options available for detailed Bayesian network and risk prediction analyses.

1.1 Converting the raw data to input files

First, format your data for analysis. BNE always uses a simple table with sample ids in column 1 followed by the variables. All variable must be labeled. Use comma- (csv) or tab-delimited (tsv) files text files for input.

df, ids, vars = format_file(data; datacols=[], delim=",")

Options:

datacols	use a subset of columns e.g. [2,5,7]	[]
delim	set the delimiter for input file	[","]
clean	clean data, use with minfreq, etc.	[false]
minfreq	min frequency for any variable state	[0.0]
recode_bool [false]	recode $0/1$ to $1/2$ and write bool disk file	
recode12 [false]	recode $0/1$ to $1/2$ and write $1/2$ disk file	

The format_file() function creates the BN.data and BN.header disk files which will be used for creating the networks and doing analyses. The output is shown in the REPL. Each variable in the raw data file is recoded to numerical values $1, 2, ..., N$. The mapping between the numerical states and the original states can be found in the recode.map disk file and is listed directly to the REPL later. This mapping will be used throughout all additional functions.

Discussion: The raw data must be formatted in a simple fully-labeled table of observations in rows and variables in columns. The data must be binary, discreet, or continuous discretized. Raw data can be numeric or text-based. Boolean data (true/false, $0/1$) should not be mixed with regular string variables. For text data input, use Y/N instead of true/false or $0/1$. For extremely large dataset, Boolean input is faster. Variable states will be automatically mapped and converted to numerical value for computation. A short example for text-based input using the popular survey data set is shown below. There are six variables, two with three states and four with two states. All observations (instances) and variables (features) are labeled.

Example:
id,AGE,CITY,EDU,OCC,SEX,TRANS
id1,adult,big,high,emp,F,car
id2,young,big,uni,emp,F,train
id3,adult,big,uni,self,M,car
id4,old,small,high,emp,F,other

Another typical example might include several simple true/false variables that describe discretized continuous variables. It is recommended to conceptually align variable states with the hypothesis being tested. For instance, if hypothesis is, "These factors are predictors of late onset diabetes", all true states should align with the expected prediction for increased for diabetes risk so that true represents increased risk across variables. This variable alignment simplifies the downstream interpretation of multistate conditional probability queries.

ID	T2D	AGE70	BMI35	A1c65
1	true	true	true	true
2	false	false	true	false
3	false	true	false	false
4	true	false	true	true

1.1.1 Worked example

The test_data directory contains some small data sets used in this example. The location of the test data sets for the unregistered BNE package is typically the /.julia/dev/BayesNetExplorer/test_data/ directory.

Start the julia REPL.
Type ; in the REPL to get a shell prompt.
Use the shell to copy the file /.julia/dev/BayesNetExplorer/test_data/survey_data.csv to your working directory.

>df, ids, vars = format_file("survey_data.csv")

Look in your current directory. The BN.data and BN.header files contain the formatted data. The recode.map shows the mappings from the input text data to numerical data. The recode.out file contains the numerical data table. Note that all input data has been converted to variables states $1, 2, ...N$ starting with one. The df, ids, and vars are returned for future use. Please note that all return values can also be output to a single variable which will result in a tuple containing df, ids, and vars.

To format a boolean data set (0/1 or true/false variables), try formatting the T2D example data set. You must set the delimiter if the file is not a CSV file. The recode_bool key word indicates two-data data.

>format_file("/path/to/T2D.testdata.tsv", delim='"', recode_bool=true)

Look at the disk files. Boolean data are now represented with 1 and 2.

1.2 Imputation

The raw data for input must be complete. Missing values must be removed or imputed. As long as observations have only some missing data (up to 20 percent, recommended), they can be used and missing the values imputed. Be sure to avoid imputing major blocks of data where multiple samples are all missing the same variables. These samples should be removed. Prediction accuracy is always better with complete accurate data.

Load data to be imputed into a data frame in the REPL, then use the impute_dataframe() function to impute missing data. The dataframe to be imputed must (currently) contain numerical values only. Note: bnstruct can also impute data, but an end goal of this project is a fully-julia package, so data imputation is implemented separately in julia.

dfi = impute_dataframe(dfm; boolout=false, k=10)

dfm is the numerical data frame with missing data, and dfi is the imputed numerical data frame. Data is imputed using a nearest-neighbors approach with a default of k=10 nearest-

neighbors. Strictly two-state data can be output as a Boolean file on disk by setting boolout to true. Multi-state numerical variables can be also be imputed.

Example:

Read the raw data into a data frame.

dfm = CSV.read("/path/to/test_data/impute_example.csv", DataFrame)

The rawdata dataframe, dfm, contains missing values for ID3, ID10, and ID15.

dfi = impute_dataframe(rawdata)

The dfi dataframe now has no missing data and can be written back to disk...

CSV.write("imputed_data.csv", impdata, delim=",")

Now proceed with the analysis using the imputed file. Be sure to run the format_file() function on the new imputed data file.

2 Creating the Bayesian network

The bne function is a wrapper that automates the process of creating and analyzing Bayesian networks. You can easily create and obtain results for any conditional probability query over an exact network of up to 15 nodes. The input files are the BN.data and BN.header files created by the format_file() function. In any analysis, you will specify a target feature (f) to analyze. Try the following simple example using the survey data set after creating the BN.data and BN.header files with format_file(). Add the semicolon at the end to suppress printing return values.

bne("BN.data", "BN.header", f="TRANS");

The screen output shows the baseline probabilities for the feature "TRANS". The TRANS state 0 indicates all variable states for transportation are shown.

```
P(TRANS=0|)

TRANS
1         2         3
0.5799771 0.1697062 0.2503167
```

Scrolling up, you can see a table showing the mapping of each analysis variable state to each original variable state. This table also include the counts for each variable and every

variable data. Thus, in our sampled data, 58 percent of people drive their car to work while 25 percent take the train, and 17 percent use other transportation.

Important: Although this example uses multistate variables, it is strongly recommended that variables be coded to binary states for most analyses.

A graphical representation of the Bayesian network is created in a separate window. The target feature (TRANS) is shown in orange. Nodes are connected by lines. By default, no direction in shown because causality is usually difficult to determine. Nevertheless, seeing the directed acyclic graph (DAG) provide a better feel for the relationships found by the structure-learning algorithm. The directed network can be seen using the DAG=true keyword.

Now include a conditional variable query to see its effect on the target variable probabilities. To query of the probability of A given B, ($P(A|B)$), list each conditional variable in the array (g) (e.g. g=["EDU"]). The specific state of the conditional variable is specified in the gs array (e.g. gs=["2"]). Let's see what effect having a university education (EDU) and being young (AGE) has on the target variable transportation (TRANS). The following query specifies the probability for all transportation states (car, other, and train) given a university education and being young (ei. $P(TRANS)|(EDU = 2, AGE = 3)$). Refer to the translation table in the REPL to match 2 to uni and 3 to young, respectively.

bne("BN.data", "BN.header", f="TRANS", g=["EDU", "AGE"], gs=["2","3"], DAG=true);

P(TRANS=0|EDU=2,AGE=3)

TRANS
1 2 3
0.5537891 0.2071520 0.2390589

The results show that young, university-educated people travel less by car and more by other means of transportation than the general public, as represented by our sample random sample of 500 people. The results provide insight into the transportation preferences of one segment of the population.

2.1 Iteration

One important feature of the BNE package is the ability to traverse the probability of a target given many variables and their states. The iterate keyword controls this function. We will now explore the target TRANS and iterate over all states of the two given variables EDU and AGE. It is not necessary to specify the conditional states (gs) because all states will be examined. Notice that six different variables are now collected from the bne run.

cpt, tt, tf, adjM, adjMB, probout = bne("BN.data", "BN.header", f="TRANS", g=["EDU", "AGE"], iterate="nodes", DAG=true);

The conditional probability table (cpt) variable is a table that contains all the major results. Each variable state for the target variable TRANS (TRANS_car, TRANS_other, TRANS_train) gets a column and these column are sorted, sequentially. The sum of all the target column probabilities in each row is 1. Next is a column of the conditional variables followed by a column showing the corresponding conditional variable states. Each variable state is traversed alone and in combination with the other variables and their state. In this case, there are 11 possibilities: 3 for AGE, 2 for EDU, plus 6 combinations of AGE and EDU. The translation table (tt) variable shows the frequency of the variables and the mapping of the name state to the numerical state (e.g., young=3, car=1). RRcalculator(cpt; target_state="TRANS_2", target_state_freq=tf[2], translation_table==tt, mincounts=[1,20], minp=0.01, maxp=0.99).

Target	car	other	train	CondVars	CondStates
TRANS	0.59	0.16	0.24	AGE,EDU	3,1
TRANS	0.58	0.15	0.25	EDU	1
TRANS	0.58	0.15	0.25	AGE,EDU	1,1
TRANS	0.58	0.14	0.26	AGE,EDU	2,1
TRANS	0.58	0.15	0.26	AGE	2
TRANS	0.58	0.17	0.23	AGE	3
TRANS	0.57	0.17	0.25	AGE	1
TRANS	0.55	0.20	0.23	AGE,EDU	3,2
TRANS	0.55	0.20	0.24	EDU	2
TRANS	0.55	0.20	0.24	AGE,EDU	1,2
TRANS	0.55	0.20	0.24	AGE,EDU	2,2

We can see that young people with a high school education are most likely to travel by car, while older university-educated people are least likely to travel by car. The difference in car travel preference is relatively small (four percent) but provides information on which demographic groups are using pubic transportation.

Now set the iterate keyword to iterate="all" and examine the output again. The cpt variable shows the target variable states with all conditional variables and all combinations of all conditional variables. There are 323 non-redundant combinations. This number increases exponentially with the number of nodes.

2.2 Risk Calculations

One of the primary goals of a Bayesian network analysis is to understand how conditional variables change the risk of a target event relative to the base rate of the target event. For example, the population baseline incidence of a middle aged male experiencing a heart attack may be about one percent. Certain factors such as fatty diet, smoking, vigorous exercise, etc. may increase or decrease that general risk. Bayesian networks are well-suited for discovering and quantifying the contribution of those additional risk factors.

The relative risk calculator, RRcalculator() function, provides a straight-forward method to discover and calculate absolute and relative risks for a target given one or more conditional variables. An example of the function call is shown below. The input variable types are shown after the double colon. Variable types are used extensively in the julia language. The type of a variable x can be checked by the command typeof(x). Please pay close attention to types if you receive an error message.

RRcalculator(cpt::DatatFrame; target_state::String, target_state_freq::Float64, translation_table::DataFrar
mincounts::Array, minp::Float64, maxp::Float64)

We will go through an example of how to calculate relative and absolute risk from a probability table generated with the bne() function, process and sort all targets and target conditions and apply a probability filter can be applied to remove conditions with low probabilities. This type of analysis is currently limited to two-state systems. Continuous or multistate variables can be used if they are discretized. For example, systolic blood pressure could be divided into high and low based on a cut-off value of 140 mmHg.

Using a data set of individuals sampled from a population with a high risk of type-2 diabetes (T2D), we will the assess relative risk of having T2D given three potential risk factors. These data are discretized and contain three variables that may be related to having or developing diabetes based on evidence from previous studies. These variables are: body mass index (BMI) $>$ 30, age $>$ 40 years, and fasting glucose $>$ 130.

2.3 Type-2 diabetes: a worked example

First, prepare the input. In this example, all variable states have been discretized and are represented as Boolean values (0/1, true/false). Note also the use of return values to capture the data, ids, and features into variables for the function.

>df, ids, features = format_file("/path/to/T2D.csv", recode_bool=true)

Run bne to make a network with four nodes and note that the original 0 states now map to 1 and the 1 states now map to 2. The 2 state is aligned with expert knowledge about these variables in that 2 indicates higher risk for diabetes. That is, T2D = 2 indicates the patient has diabetes, BMI30 = 2 means BMI is >30, etc.

> cpt, tt, tf, adjM, adjMb = bne("BN.data", "BN.header", f="T2D", iterate="all", DAG=true);

The output indicates that 35 percent of the population has T2D. This is the baseline risk for T2D in this population, assuming that our sampling is truly random.

Examine the first and last lines of the conditional probability table. We can see that, as expected, our aligned risk factors are less frequent in the T2D_false samples and very frequent in our T2D_true samples.

Target	False	True	CondVars	CondStates
T2D	0.91	0.08	BMI30,AGE40,GLUC130	1,1,1
T2D	0.87	0.12	BMI30,GLUC130	1,1
T2D	0.87	0.12	BMI30,AGE40	1,1
.				
.				
.				
T2D	0.30	0.69	AGE40,GLUC130	2,2
T2D	0.26	0.73	BMI30,GLUC130	2,2
T2D	0.18	0.81	BMI30,AGE40,GLUC130	2,2,2

The RRcalculator can now be used to calculated the absolute and relative risks of each of the traits and the traits in combination with each other. The input for the RRcalculator is the variable cpt (data frame) from bne. The function also requires the target state (i.e. T2D_true) and the frequency (baseline absolute risk) of that target state obtainted from the from the tf variable (tf[2] = 0.34895). The tt variable again shows the translation_table. The minimum number of samples in allowed in the final and conditional subsets can be set if desired.

>RRtable = RRcalculator(cpt, target_state="T2D_2", target_state_freq=tf[2], translation_table=tt, mincounts=[1,20])

The RRcalculator() returns a relative risk table. The RRtable has several columns. The most important columns for most runs are the absolute and the relative risks. We define risk categories as follows.

Baseline absolute risk: the frequency of the trait in the sample population. This is the number of samples that have the target trait divided by the number in the sample population. This is the standard definition of absolute risk.

Absolute risk ratio: the frequency of the trait in the conditional subset compared to the baseline absolute risk. In other words, if you have the conditional variable states HighBMI=true, HighGlucose=true, ..., what is your risk compared to the population baseline risk.

Relative risk ratio: the frequency of the target trait in the conditional subset compare to the frequency of the target trait in samples not having those traits. This is sometimes referred to as the contrapositive risk. In other words, if you have conditional variable states V1=true, V2=false, and V3=true, what is your risk compared to those that have conditional variable states V1=false, V2=true, V3=false.

The relative risk ratio can be quite large and sometimes non-linear as more variables are considered, but it is an important metric for identifying and assessing environmental exposure variables or patient behavioral traits to affect outcomes. For instance, in another experiment you find a relative risk ratio of 10 for heart disease when V1("high stress")=true and V2("aerobic exercise")=false and V3("vaping")=true. The result immediately suggests that eliminating high stress, vaping, and getting exercise will greatly improve outcomes for this trait. Using the rrrdenom keyword allows you to set the exact states of the conditional

variables in the denominator of the relative risk calculation to isolate the effects of specific variables.

2.4 Optimization and Interpretation

Bayesian networks provide an excellent way to analyze the relationships among all variables in a data set. The networks and the relative probabilities provide an explainable artificial intelligence (AI) approach to investigate many questions. Key concepts for a reproducible analysis is 1) a large sample size and 2) an independent replication data set.

Sample size is very important. As the number of conditional nodes in a specific query or global analysis increase, the number of samples in the conditional subset will often become very small. This is often the case when low frequency variables are combined in a conditional query. Consider a data set with 50 samples. Variables V1=true and V2=true both have baseline frequencies of 0.1. The sample size is too small to assess conditional independence V1 and V2 or the P(target)|(V1=true, V2=true) with high accuracy (small confidence intervals).

Relative risk ratios are strongly affected by small sample sizes in the conditional query. If a conditional query combination of several variables of leads to a number of samples with the given query combination but a very small or zero samples with the contrapositive combination, the relative risk ratio will be extremely large. Careful consideration of sample size is needed to assess if result is accurate and reproducible in other data sets.

For example, in the tutorial results above, the first row of the cpt table shows that self-employed university-educated males living in a big city prefer to travel by car. This seems like a reasonable result. There are only six samples of 500 that meet these inclusion criteria in the data set. As expected, based on the probability, all six prefer car travel. Now consider sampling one additional individual that prefers train travel. The one additional sample shifts the estimated probability by 14 percent. The effect could be much greater if the conditional subset was only three individuals.

2.4.1 Getting reproducible results

The following techniques can be used to obtain high-quality, reproducible results.

1. Use the largest data set available and continue to add more data if it becomes available.

2. Check the initial frequencies for all classes for each variable. Consider omitting variables with a minimum class frequency of less than 1-5 percent. Alternatively, merge low-frequency classes, if appropriate.

3. Consider results with at least 20 or more samples in the conditional subset (mincond-count=20). If the target state is rare, increase the number. If 20 is too high, set to 5 or 10 but consider whether the overall size of the data set should be larger to perform this query.

4. Bootstrap the result. Use the bne() function with the rr_bootstrap option to perform bootstrap resampling of the data to generate 95 percent confidence intervals around the network propagated relative risk estimates. Large intervals often indicate low numbers in the conditional sample set. The network point estimate should be similar to the estimate from the resampled distribution. Strong deviation can be the result of a low number of samples for that query, and in these cases, the results should be interpreted with caution. The CI95 around the network estimate is typically small and stable due to network constraint. It is also useful to run the same query using bnemle() to get the CI95 based strictly on the counts of variables in the sample set. The CI95 from bnemle() is typically large until the query contain dozens of samples.

5. Hand check the final results in the original data before reporting. Be critical. Does it make sense that you just identified a new factor that leads to a 500-fold increased risk of autism? Most variables with extremely large effects on a target are known.

3 Additional functions

3.1 Non-network-propagated conditional probabilities

The bnemle() function allows you to estimate conditional probabilities without network traversal. These probabilities are obtained directly from the original data using a probability tree style traversal. Samples meeting all conditional criteria form a new sample space from which the target probability is then calculated. In this approach, like the network approach, conditional variables are not independent. Unlike the network, the only variables affecting the probability calculation are those that occur in the query itself. A Laplace correction is used if zero-count conditional sample spaces are encountered. Bootstrapping over samples is used to provide a 95% confidence interval.

Comparing the results from the direct estimate to the network can be instructional. A simple query may produce a probability estimate substantially higher than a similar network-based estimate. In such cases, it is useful to examine the network nodes separating the target node and the conditional variable. If the conditional variable lies outside the Markov blanket then the network provides information on how the risk is propagated through other variables (network nodes) and the pathway that leads to that risk estimate. This situation may occur when there is a clear temporal ordering of variable being analyzed.

3.2 Naive Bayes conditional probabilities

The bnenbc() function allows the user to experiment with a naive Bayes approach to the probability calculations. Here, variables are considered independent. The bnenbc() function also serves as a naive Bayes classifier to test the probability of a hypothesis against the alternate hypothesis given the input set of conditional variables. Bootstrapping over samples is used to provide a 95% confidence interval.

3.3 Network plotting

The plot_network() function provides the user with a means to create custome networks created by bne. The input for plot_network() is the adjacency matrix from bne. The user can control the color and size of the nodes as well as line size and other attributes. Undirected networks can be moralized. Information can also be added to each edge.

3.4 Miscellaneous functions

3.4.1 Bootstrapping

The bootstrapRRtable() function is provided to automatically run network-based bootstrapping on a series of queries in a relative risk table (RRtable).

3.4.2 Correlation

Ideally, variables going into Bayesian network will be conditionally dependent but not highly correlated. Collections of colinear variables can produce undesirable effects on network structure and mask the impact of other key variables in the network. Colinear variables can be identified using the pairwise_correlations() and the correlation_analyzer() functions.

3.4.3 Feature selection

The feature_selector() function allows the target and conditional variables to be evaluated in pairs using a small network. A user-specifed number of simulated random variables are used to create a small network. This approach can be used to evaluate the effect of a conditional variable(s) on a target variable in the context of a network.

The bnescan() functions provides a fast method for testing the $P(Y|X_{1,2,...,N})$ where Y is the target variable and $X_{1,2,...,N}$ represents all other variables in the input dataset. The method quickly identifies the variables with the highest and lowest impact on the target as assessed by the absolute and relative risk ratios. A confidence interval is also provide for each estimate. Run times with bootstrapping on moderate data sets should be approximately two evaluations per second. This function can help to identify high-impact conditional dependencies for further testing in large datasets.

4 Acknowledgments

This code was inspired by the many discussions of the utility and effectiveness of Bayesian network and explainable AI systems in the laboratories of Drs. Tristani and Yandell at the

University of Utah. Many thanks to E.J. Hernandez and S. Wesolowski for their expertise in Bayesian and graphical systems analyses.

Please send comments, bug reports, and feature requests to scott.watkins@genetics.utah.edu.