The Bayes Net Explorer package

Watkins W.S.[1]

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

April 4, 2025

Overview

The Bayes Net Explorer (BNE) package is a program package to automate 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 automate network conditional probability estimation for a target variable given one or more conditional variables using Bayesian networks The relationship among input variables is learned directly from the data and is visualized with an undirected network or directed acyclic graph (DAG). Any target variable can be analyzed conditionally with any combination of variables in the network. For small networks with binary variables, a target variable can be assessed with all conditional variables in all variable states. The absolute and relative risks ratios for any target outcome with any combination of conditional variables can be calculated. Markov blankets can be displayed graphically to allow selection and refinement of the most relevant variables for a specific hypotheses and to identify conditional variables with the largest impact on the target.

This tutorial describes using the BNE package for simple Bayesian network analyses. The functions are presented in the typical order of use. Worked examples, using common data sets for 1) transportation preferences for 500 commuters, and 2) type-2 diabetes risk, are presented. The specific examples describe only basic usage. Users are also encouraged to try the many additional options available for more detailed Bayesian network and risk prediction analyses.

Requirements

The user should have Julia (v1.10+) installed. Additionally, a fully working version of R (v4.4+) and R-studio is needed for network analyses. A GKS-term, such as XQuartz (MacOS) or XGKS (linux), must also be installed to visualize graphics. This package has been developed and tested with Julia 1.11.0 and R 4.4.1 in MacOS-Sequoia and Linux centos environments. The current package utilizes the network learning and propagation methods in bnstruct and gRain packages from R. It is anticipated that future versions of this package will incorporate structure learning algorithms in julia.

Installation

First, install R-studio on your machine. Install Bioconductor. Use Bioconductor to install the R packages and their dependencies using the command BiocManager::install(c("package name")). Install each of the packages listed below. Make sure that you can properly load each package using the library command. Type library("bnstruct") at the R-studio prompt and repeat for the other libraries. Make sure that there are no package errors or missing dependencies before installing the BayesNetExplorer package.

- gRbase
- ggraph
- Rgraphviz
- gRain
- RBGL
- graph
- bnstruct

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

```
julia> ]
(@v1.11) 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. The package will be registered as it is further developed.

```
(@v1.11) pkg> add https://github.com/ScottWatkins/BayesNetExplorer
```

The package and all dependencies will be installed automatically. Type the delete key to get back to the julia prompt. Now load the packages for use.

```
julia> using DataFrames
julia> using BayesNetExplorer
```

That's it. You are ready to explore!

A quick note before proceeding, the Julia language has a programming structure that is similar to other data science languages such as python, R, and matlab. Keep this structure in mind while going through the tutorial.

Functions have the following structure:

```
function(input1, input2, ... inputN; keyword1=value, keyword2=value, ...)
```

Functions may output one or more values that can be assigned to variables:

```
x = sum(1:3)     The variable x is assigned the integer value 6.
x = sum(1:3);    The semicolon suppresses output to the screen.
```

See any variable by typing the variable at the prompt (*e.g.,* julia>x). See help information for a function by typing a ? and then the function name (*e.g.,* julia>?sum)

1 Preparing data sets for the BayesNetExplorer

1.1 Converting the raw data to input files

First, format your data for analysis. BNE *always* uses a simple table input format. Sample ids are *always* in column 1. Variables start in column 2. All variables must be labeled with a simple alphanumeric string that does not contain spaces or special characters (*e.g.,* height or BMI30, but not 30BMI). Use comma- (csv) or tab-delimited (tsv) text files for input. These formats allow easy exchange between BNE and excel, other spreadsheets, and SQL databases for large projects. Use the following command to format the data file.

```
df, ids, vars = format_file(”datafile.csv”);

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     recode 0/1 to 1/2 for Boolean data      [false]
recode12        recode 0/1 to 1/2  [false]
```

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 in REPL later. The recoded variable mapping is used throughout all additional functions.

Discussion: Input data are simple fully-labeled tables 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. Variable states will be automatically mapped and converted to ordinal values for fast computation. For datasets with many variables, use the datacols option to select a subset of columns to be used for the analysis. The columns processed here will become the nodes in the the Bayesian network.

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 and variables are labeled.

```
IID,TRANSPORTION,AGE,CITY,EDU,OCC,SEX
id1,car,adult,big,high,emp,F
id2,train,young,big,uni,emp,F
id3,car,adult,big,uni,self,M
id4,other,small,high,emp,F
```

Another example includes simple true/false variables that describe discretized continuous variables. It is recommended to conceptually align variable states with the hypothesis being tested, if possible. For instance, if the 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   A1C_65
1       true    true    true    true
2       false   false   true    false
3       false   true    false   false
4       true    false   true    true
```

Now, create a new working directory for the analysis. Move (cd) into that directory. Start julia and load the DataFrames and BNE packages.

```
> julia
julia> using DataFrames
julia> using BayesNetExplorer
```

Use the built-in shell to copy a data file to your current directory:

```
julia> ;
shell> cp ~/.julia/dev/BayesNetExplorer/data/survey_data.csv ./
shell> <backspace/delete>
```

Format the file using the format_file() function. Try the command with and without the semicolon.

```
julia> 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 for computation. 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 return values contain the data, the ids, and the variables future use.

Try formatting a Boolean data set using the very small T2D.tsv example data set. You must set the delimiter since this file is not a csv file. The recode_bool key word indicates strict Boolean input data containing only true/false and/or 0/1 variables.

```
julia>format_file("~/.julia/dev/BayesNetExplorer/data/T2D.tsv",recode_bool=true)
```

Look at the disk files. The recode.map shows the Boolean data are now represented with 1 and 2 (*i.e.,* 0 is now 1, and 1 is now 2). A new uniform Boolean file, recoded.bool.out provides a unified true/false data set for future use. Hint: You can use the showcodes() function at any time to show the variable information for the current analysis.

```
julia> showcodes()
```

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), the missing values can be imputed. Be sure to avoid imputing major blocks of data where multiple samples are all missing the same variables. These samples should be removed. Network construction is always better with complete accurate data.

Read the example data set for imputation into a data frame in the REPL. Note that the variables are numerical (Int64) and that samples 3, 10, and 15 have missing data.

```
julia> dfm = CSV.read("~/.julia/dev/BayesNetExplorer/data/impute_example.csv",
DataFrame)
```

Use the impute_dataframe() function to impute missing data. Important: the data to be imputed must contain numerical values only.

```
julia> dfi = impute_dataframe(dfm)
```

dfm is the original 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. Imputed two-state data can be output as a Boolean file on disk by setting boolout to true. The dfi data frame now has no missing data and can be written back to disk (*e.g.,* julia> CSV.write("filename", dfi)) or used directly. Now proceed with the analysis using the imputed file by running the format_file() function on the new imputed data file or data frame.

Note: The use of the julia> prompt is omitted from commands in the tutorial going forward.

2 Bayesian networks

The BayesNetExplorer (bne) function 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 about 15 nodes. In any analysis, you will specify a conditional probability query in the form $P(A = Astate|B = Bstate)$ where n is the number encoding a specific state of the variable. Load the survey data, and examine the df, ids, and vars variables.

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

To see the input variables states and their numerical codes, type showcodes() at anytime.

2.1 Simple conditional queries

Now, try creating a simple Bayesian network with the survey data set. What is the probability that university attendance is related to age in our survey. University is coded as EDU=2. The AGE variable in our survey data has three states (adult, old, young). Let's start by conditioning on adults which are represented by AGE=1 (remember to add the semi-colon at the end).

```
bne("P(EDU=2|AGE=1)");
```

Scrolling up, you can see a table showing each variable's state and a numeric code. The table also includes the counts and baseline (marginal) probabilities for each variable state over the entire data set. The baseline probability of a high school education is 73%, while the baseline probability for university level education is 27% . Scrolling back down, the results of our conditional probability query shows that the conditional probability of university education given being an adult is 31%. Contrast this value with the baseline probability of 27%. Now, calculate the conditional probability of university attendance for the older people in our survey.

```
bne("P(EDU=2|AGE=2)");
```

The output shows that the probability of university attendance for older people is only 12.5% as compared to the baseline probability of 27%. Thus, the conditional probabilities indicate a trend of increasing higher education over time for in the survey participants.

2.2 Networks

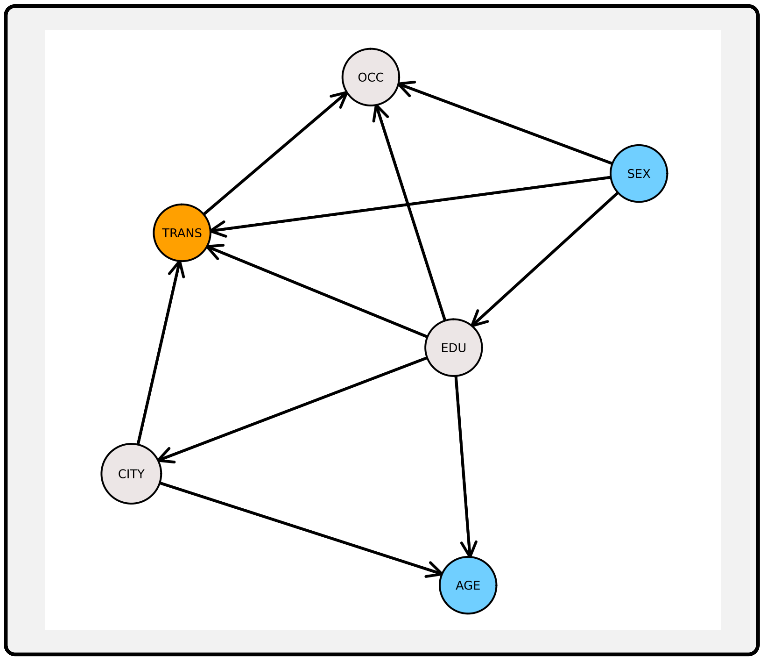
For each query, a graphical representation of the corresponding Bayesian network is created in a separate window. The target variable (EDU) is shown in orange. Each variable is represented as a node. Nodes (vertices) are connected by edges (arcs). These relationships among the variables and the connecting edges are learned by a network learning algorithm. By default, no edge directionality is shown because causality is often difficult to determine. Nevertheless, seeing the directed acyclic graph (DAG) can provide a better understanding of the relationships found by the structure-learning algorithm. The directed network can be seen using the DAG=true keyword.

Now, let's see what effect of being young (AGE=3) and female (SEX=1) has on using transportation by train (TRANS=3). The following query specifies the conditional probability of train transportation given being female and being young. The keyword DAG=true is added to produce a network with directed edges.

```
bne("P(TRANS=3|AGE=3,SEX=1)", DAG=true);
```

The results show that young, university-educated females travel less by train than others in our survey, overall. These results provide insight into the transportation preferences of one segment of the population. Examine the network (Figure 1). The learning algorithm has optimized the relationships among the variables in the network producing a directed acyclic graph (DAG). The DAG shows that sex is a parent of transportation. Other variables also have a direct impact on transportation. Age, however, is only distantly connected to transportation and is formally outside of the target's Markov blanket. This suggests that, given all the data in the survey, age is less important than sex on transportation choice in this query. Try some additional queries to test this hypothesis. It is critical to note that other network learning algorithms and networks constructed with expert knowledge may produce different DAGs. Target probability estimates may change when the inferred networks produce different Markov blankets for the target.

Figure 1: Survey data network



2.3 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 previous results for transportation by iterating over all states of the two conditional variables, EDU and AGE. Notice that six return variables are now collected from the bne run.

```
dfp, vt, tf, adjM, adjMb, p = bne("P(TRANS=3|AGE=3,SEX=1)", iterate="nodes");
```

The dft variable is a data frame that contains the conditional probabilities from our query. View the table by typing dfp at the prompt. Each variable state for the target variable TRANS (that is, TRANS_car, TRANS_other, TRANS_train) is shown in a column and these columns are sorted, sequentially. The sum of the target column probabilities for each row is 1. The next column contains the conditional variables followed by a column showing the corresponding conditional variable codes. Each conditional variable state is tested alone and in combination with the other conditional variables and states. In this case, there are 11 possibilities: 3 for AGE, 2 for EDU, plus 6 combinations of AGE and EDU (Table 1).

Table 1: Conditional Probability of Transportation given Age and Sex.

Target	TRANS_car	TRANS_other	TRANS_train	CondVariables	CondStates
TRANS	0.59	0.21	0.20	AGE,SEX	3,1
TRANS	0.59	0.20	0.21	SEX	1
TRANS	0.59	0.17	0.24	AGE,SEX	2,1
TRANS	0.59	0.20	0.21	AGE,SEX	1,1
TRANS	0.58	0.15	0.26	AGE	2
TRANS	0.58	0.18	0.24	AGE	3
TRANS	0.58	0.17	0.25	AGE	1
TRANS	0.58	0.14	0.28	AGE,SEX	2,2
TRANS	0.57	0.16	0.27	AGE,SEX	3,2
TRANS	0.57	0.15	0.28	SEX	2
TRANS	0.57	0.15	0.28	AGE,SEX	1,2

Examine the conditional probabilities for transportation (columns two, three and four). Iteration of all combinations of the age and sex conditional variables reveals several interesting features about transportation: 1) age and sex have minimal impact on transportation by car, 2) transportation by other means is somewhat impacted, and 3) the largest impact of age and sex is on travel by train. Moreover, sex is more important than age on the decision to travel by train. Females (SEX=1) of any age are less likely to choose train travel than males (SEX=2) of any age. There is an eight precent difference between the minimum and maximum conditional probabilities.

Now set the iterate keyword to iterate="all", rerun the query, and examine the output again. The dfp data frame shows the target variable states with all combinations of conditional variables.

There are 323 combinations. This number increases exponentially with the number of variables. Notice that many queries have probabilities close to one or zero. This usually indicates that there are few if any actual observations that have that particular combination of conditional variables. There are several options to control the output and limit or filter queries.

3 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 risk factors.

We will go through an example of how to calculate relative and absolute risk from a probability table generated with the bne() function. Next, we will process and sort the targets and target conditions and apply a probability filter to remove conditions with low probabilities. This type of analysis is currently limited to two-state systems. Continuous or multistate variables can still 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.

3.1 Type-2 diabetes risk: a worked example

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) $\geq$ 30, age $\geq$ 40 years, and fasting glucose $\geq$ 130 mg/dL. The variable states have been discretized and are represented as Boolean values (0/1). Format the data set as before. Examine the df, ids, and variables, and check the encoding.

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

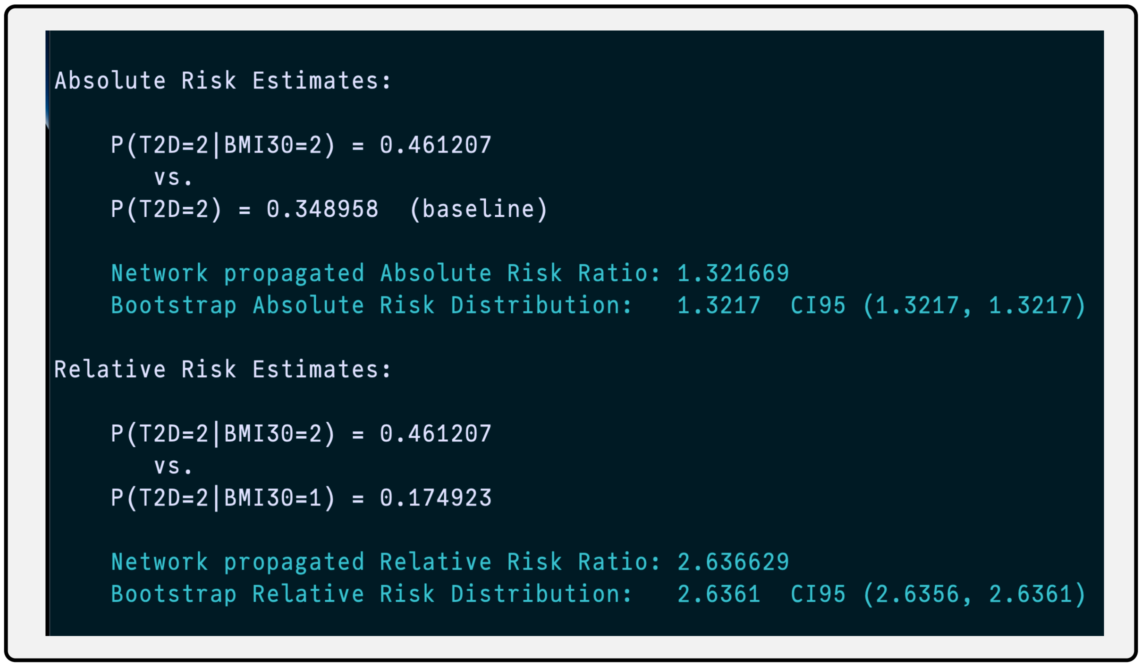
The state of each risk variable is aligned with expert knowledge about each of these variables. That is, T2D=2 indicates the patient has diabetes, BMI30 = 2 means the patient's BMI is >30, etc.

Run bne to perform a conditional probability query for T2D given high BMI. Add the keyword relrisk=true to calculate the absolute and relative risks. The risk estimates will be automatically bootstrapped (default 100 bootstraps) to determine a credible region about the point estimate.

Each bootstrap is performed by randomly resampling the data matrix (with replacement), recreating the network, and recalculating the probabilities and risk estimates. The final output is shown in Figure 2.

```
dfp, vt, tf, adjM, adjMb, p = bne(”P(T2D=2|BMI30=2)”, relrisk=true);
```

Figure 2: Type-2 Diabetes Risk given High BMI



```
Absolute Risk Estimates:

    P(T2D=2|BMI30=2) = 0.461207
       vs.
    P(T2D=2) = 0.348958  (baseline)

    Network propagated Absolute Risk Ratio: 1.321669
    Bootstrap Absolute Risk Distribution:   1.3217  CI95 (1.3217, 1.3217)

Relative Risk Estimates:

    P(T2D=2|BMI30=2) = 0.461207
       vs.
    P(T2D=2|BMI30=1) = 0.174923

    Network propagated Relative Risk Ratio: 2.636629
    Bootstrap Relative Risk Distribution:   2.6361  CI95 (2.6356, 2.6361)
```

The screen output now shows the absolute and relative risk ratios for T2D given high BMI. Point estimates for the risk ratios are produced by propagating the probabilities through the Bayesian network. The bootstrap estimates can be contrasted with the point estimates. The bootstrap estimate is the median estimate from 100 randomly resampled datasets. A 95% empirical confidence interval from this distribution is reported as the credible interval for the estimate. The point estimate and bootstrapped estimate are typically very similar, provided there are adequate samples for the query. When sample sizes are large, the empirical distribution and 95% confidence interval is highly constrained by the network structure, and the credible region is typically very narrow. When sample sizes are small, the resampled network structures can differ from the original network leading to differences between the bootstrapped and point estimates. When the difference between the point estimate and the bootstrap estimate is >10%, the query is flagged, and the user should carefully examine the underlying reason for the difference.

Next, use the iterate keyword to ”all” to estimate the probabilities for all variable state combinations.

```
dfp, vt, tf, adjM, adjMb, p = bne(”P(T2D=2|BMI30=2)”, iterate=”all”);
```

The baseline probability of T2D is nearly 35 percent in this random sample of a high-risk population. Now examine the first and last lines of the conditional probability data frame (dfp). As expected, T2D_true (column three) shows that T2D is more probable than the baseline T2D estimate given one, two, or three risk factors. With all three risk factors, the probability of having type-2 diabetes is 81.5 percent! In contrast, T2D_false (column 1) shows that not having those risk factors greatly increases the probability of not having type-2 diabetes (Table 2).

Table 2: Conditional Probability of T2D given BMI, Glucose, and Age.

Target	T2D_false	T2D_true	CondVariables	CondStates
T2D	0.91	0.09	BMI30,AGE40,GLUC130	1,1,1
T2D	0.88	0.12	BMI30,GLUC130	1,1
T2D	0.87	0.13	BMI30,AGE40	1,1
T2D	0.83	0.17	AGE40,GLUC130	1,1
T2D	0.83	0.18	BMI30	1
T2D	0.79	0.21	GLUC130	1
T2D	0.75	0.25	BMI30,AGE40,GLUC130	2,1,1
|	|	|	|	|
T2D	0.40	0.60	BMI30,AGE40	2,2
T2D	0.37	0.63	GLUC130	2
T2D	0.31	0.69	BMI30,AGE40,GLUC130	2,1,2
T2D	0.31	0.69	AGE40,GLUC130	2,2
T2D	0.27	0.73	BMI30,GLUC130	2,2
T2D	0.18	0.82	BMI30,AGE40,GLUC130	2,2,2

3.2 Absolute and relative risk ratios

We often want to know the absolute and relative risks and risk ratios for more than one set of conditional variables and states. The Risk Ratio calculator, RRalculator(), provides a method to discover and calculate absolute and relative risks ratios for a target given one or more conditional variables. To get started, let's examine the six variables returned by the bne run used in risk calculation above, $P(T2D = 2|BMI30 = 2)$. These bne output variables are used by the RRcalculator() function to calculate the absolute and relative risk ratios (ARR and RRR) for the target with combinations of conditional variables. Remember, using the iterate keyword will iterate all possible combinations for 1) nodes listed in the query or 2) all variables in the current network.

```
dfp, vt, tf, adjM, adjMb, p = bne(”P(T2D=2|BMI30=2)”, iterate=”all”);
```

- dfp is a data frame with output probabilities
- vt is a variable table (data frame) showing the variables and states
- tf is an array with the baseline target frequencies for each target state
- adjM and adjMb are the network adjacency matrices
- p is an array containing the conditional probabilities for the input query

Use the RRcalculator() to calculate the absolute and relative risks ratios for type-2 diabetes given combinations of the three risk factors in the network. The inputs for the RRcalculator() function are the dfp data frame, the vt table, and the target state baseline frequency. The function also requires you to specify the target state (*i.e.,* T2D_true). The baseline frequency for T2D=2 (true) can be entered as 0.34895 or can be used from the tf array (*i.e.,* tf[2]). The variable_table values are specified by the vt table. The minimum number of samples allowed in the final and conditional subsets can be set, if desired.

```
RRtable = RRcalculator(dfp, target_state=”T2D_true”,
target_state_freq=tf[2], variable_table=vt, mincounts=[1, 20])
```

The RRcalculator() returns a risk ratio table. The RRtable has several columns. The most important columns for most runs are the absolute and the relative risks ratios. These risks are defined below. View the RRtable and examine the absolute and relative risk ratio estimates. The table can be sorted to find the most or least impactful sets of conditional variables (*e.g.,* sort(RRtable, 4)). Find the line where all conditional risk factors are true. This line shows that the absolute risk ratio for T2D given high BMI, older age, and high blood glucose is 2.3-fold higher than the T2D in this population. Moreover, the relative risk ratio of T2D given these three risk factors is more than 9-fold greater than not having any of the three risk factors.

3.3 Risk definitions

Absolute (baseline) risk: the frequency of the trait in the population. This is the number of individuals that have the target trait divided by the total number of individuals in the sample population. This value is also known as the marginal probability of the variable.

Absolute risk ratio: the frequency of the trait in the conditional subset compared to the baseline absolute risk. In other words, the conditional probability estimate divided by the absolute risk. For example, $P(T2D = true|BMI30 = true)/P(T2D = true)$.

Relative risk ratio: the frequency of the target trait in the conditional subset compared to the frequency of the target trait in samples not having those specific traits. For example, $P(T2D = true|BMI30 = true, GLUC130 = true)/P(T2D = true|BMI30 = false, GLUC130 =$

$false)$. In other words, it is the target's risk given the conditional variable states V1=true and V2=true, as compared to the target's risk of having conditional variable states V1=false, and V2=false, the contrapositive query. The relative risk ratio can become quite large and sometimes non-linear as more conditional variables are considered and conditional subset counts become small. Additionally, relative risk ratios can be used to isolate a specific conditional variable state in the context of other conditional variables. These types of custom relative risk estimates can be calculated by specifying the relative risk denominator in the bne and cpq programs. Using the example above but setting the conditional variable states in the denominator to V1=true and V2=false predicts the effect of V2 on the target in the context of V1=true.

3.4 Optimization and Interpretation

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

Sample size is 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. Low frequency variables exacerbate this problem especially when they 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 individual with V1=true and V2=true may not occur in the data set, and the query probability would be estimated using a zero-state correction factor. Nevertheless, Bayesian networks can still provide valid estimates for relatively small sample sizes because they 1) can leverage prior information about each variable, 2) provide network propagation to arrive at a best estimate given the data, and 3) use resampling methods to obtain empirical credible intervals for those estimates.

Relative risk ratios are affected by small sample sizes. If a conditional query combination has an adequate number of individuals in the query (numerator) but very few or zero samples in contrapositive query (denominator), the relative risk ratio can become extremely large. Careful consideration of this situation is needed to assess if an "impressive" result is accurate and will be reproducible in other data sets.

Another consideration in the creation and interpretation of Bayesian networks involves the magnitude of the impact of a conditional variable on a target outcome. Structure learning algorithms will optimize these relationships placing the most influential conditional variables within the target's Markov blanket. For testing variables of small effect on a target, it may be advantageous to start with networks that include only a few variables and then sequentially add new variables into the model, evaluating which variables are included and which variables are displaced in the Markov blanket.

3.5 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 when it becomes available. Make sure the input data has high accuracy and low missingness. Check the initial frequencies for all classes for each variable. Consider omitting variables with a minimum class frequency of less than 5 percent. Alternatively, merge low-frequency classes if appropriate.

2. Pay very close attention to the network structure. Different combinations of input variables can change the network structure. Different learning and scoring methods can also produce different network structures even using the same input variables. Check the Markov blanket for the query target! As long as all nodes positions for all query variables remain consistent in the Markov blanket, the probability and risk estimates should be comparable.

3. Consider results with at least 20 or more samples in the conditional subset. If the target state is rare, increase that number. If 20 is too high, set the minimum counts lower 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 set to 1000 to perform bootstrap resampling of the data and generate 95 percent confidence intervals (CI95) around the network propagated relative risk ratio estimates. Large intervals often indicate low numbers in the conditional sample set. The network point estimate should be similar to the median estimate from the resampled distribution. The CI95 around the bootstrap estimate is typically small and stable due to network constraint. Differences can be the result of a low number of samples for that query or other factors such as more than one equally good networks. In these cases, the results should be interpreted with caution. It is also useful to run the same query using cpq() to get the CI95 using non-network methods.

5. Check for technical issues. Be critical. Does it make sense that you just identified a new factor that leads to a 500-fold increased risk of autism? Keep in mind that most variables with extremely large effects on well-studied target conditions are known and have been reported in the literature.

6. A key feature of Bayesian networks is the ability to identify the relationships among variables that influence a trait. Adding one or more new variables to an existing network may change the relationships among the target and conditional variables. When new variables displace the original conditional variable(s) from the target's Markov blanket, the target is no longer conditionally dependent on that variable, given the new data and network. The estimated impact of a displaced variable on a target may decrease considerably. Importantly, the new variables are likely to be more relevant and have a higher impact on the target.

7. It is useful to compare the network-based estimate to the cpq estimate. When the conditional variables are all in the Markov blanket, the two methods tend to give similar results. If the estimates are different, it is likely that the network will show what other variables are influencing

the probability estimate. The network provides the best explanation of how all input data is related to the target variable.

4 Additional functions

4.1 Non-network-propagated conditional probabilities

The conditional probability query function, cpq(), allows you to estimate conditional probabilities without network traversal. These probabilities are obtained directly from the original data using the standard conditional probability definition. 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 expected to be conditionally dependent. Unlike the network, the only variables affecting the probability calculation are those that occur in the query itself. Bootstrapping over samples is used to provide a 95% confidence interval.

Comparing the cpq() results to the bne() can be instructional. A simple query may produce a probability estimate substantially higher or lower 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. The network provides information on how the probabilities are propagated through other variables (network nodes) and the pathway that leads to the final estimate. Variables outside the Markov blanket are conditionally independent of the target in the network calculation. This situation may occur when there is a clear temporal ordering of variable being analyzed.

The cpq function also implements binomial and Fisher tests to provide perspective. These tests assume variable independence and should be interpreted as such.

4.1.1 Merging many similar variables to represent a single outcome

When dealing with large data sets such as electronic medical records, it is often desirable to create a target outcome that includes multiple related variables. For instance, individuals with asthma may be represented by 15 different icd10dx codes that indicate various asthma types. Given a list of columns with 0/1 data where 1 indicates the presence of the phenotype, the **targetmaker()** function combines all variables (column names) passed in a string or a text file, one variable per line, into a single new variable and then adds that variable to the data frame. The new data frame can be used as input data for cpq. The targetmaker functionality is now incorporated into the phenomaker function().

4.1.2 Combining specific variables and states to create a new target or conditional variable

Often, one thinks of a new combination of variables in specific states to test as either a conditional or a target variable. The **phenomaker()** function allows you to combine any number of variables with specific states into a single new variable using AND (intersect) and OR (union) set operators. This function allows the user to create, for instance, a target variable that includes only individuals with arrhythmia, high fitness, and a positive family history of a disease if they live in Utah but not California. Likewise, a specific conditional variable with, for instance, selected medication combinations such as medA, and medB, but not C, or any use of D can be quickly created. Merging complex combinations of variables into a single new variable often improves computation times for big screening projects. Phenomaker can also merge a list of variables in a string or a file like targetmaker.

4.2 Scanning hundreds of variables against a target

The cpqscan() function allows the user to perform pairwise conditional probability tests, calculate absolute and relative risk ratio estimates and get binomial test significance across a full set of variables. The scan can identify conditional variables with the highest impact on a target variable. Composite variables, made with the targetmaker() or phenomaker() functions, can be added to the dataframe and tested across all variables. Once a small set of variables are identified, the standard cpq() and bne() functions can be used to test the combinatorial effects of multiple conditional variables.

4.3 Naive Bayes tools

The bne naive Bayes calculator, bnenbc(), function allows the user to experiment with a naive Bayes approach to the probability calculations. Here, variables are considered to be 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.

4.4 Network plotting

The plot_network() function provides the user with a means to create custom networks based on bne output. 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.

4.5 RRtable bootstrapping

The bootstrapRRtable() function is provided to automatically run network-based bootstrapping queries using a relative risk table.

4.6 Correlation

Ideally, variables going into Bayesian network will be conditionally dependent but not highly correlated. Collections of collinear variables can produce undesirable effects on network structure and mask the impact of other key variables in the network. Collinear variables can be identified using the pairwise_correlations() and the correlation_analyzer() functions. Plots containing the correlations and P-values can be created.

4.7 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 provided for each estimate. Run times with bootstrapping on moderate data sets should be approximately two evaluations per second. For large data sets, this function is mostly replaced by the cpqscan() function.

5 Acknowledgments

This code was inspired by the many discussions of the utility and effectiveness of Bayesian networks 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.

Appendices

A Workflow examples

A.1 Scan a large set of variables for conditional association

For datasets that are in the range of 10000 observations and 1000 variables, the cpqscan() function can be used without a need to alter the data. Use cpqscan with bootstraps=1 to speed up the runs. Then, obtain valid bootstraps on selected variables from the scan.

Large data sets of more than a million observations and 20,000 variables can also be analyzed in BNE. Read a large compressed disk file, bigdata.csv.gz with CSV.read(). The decompression is handled automatically.

For most operations, break up the operations into columnar chunks. That is, address the main dataframe by creating a new small dataframe that points to a subset of columns of the main dataframe. The julia language is column centric and chunking by column improve calculation speed.

df = CSV.read("bigfile.csv.gz", DataFrame)

Now, select the columns to compute on in a new df.

dfx = select(df, 1,2:10) dfx = select(df, 1,11,23,44)

Continue as usual using the smaller (non-copied) dataframe.

A.2 Notes on large file computations

Benchmarks suggests that a dataframe of size(1e6, 2e4) is computationally trackable on a laptop. For best performance, conditional probability calculations should be executed on the smallest number of variables possible. That is, subset the key variables for the calculation from the large dataframe into a new dataframe using the smallest number of columns. Repeat subsetting as necessary – overwriting the subset dataframe – when scanning hundreds of variables. Reading a compressed file of 500Mb requires several minutes. Writing a compressed file of this size requires a couple of hours.