Bayesian Networks 2014-2015

Assignment II – Learning Bayesian Networks

1 Introduction

The purpose of this assignment is to test and possibly expand your knowledge about *learning* Bayesian networks from data, by exploring various issues such as comparison of learning algorithms, dealing with missing data and evaluation of the networks learnt. Recall that learning Bayesian networks involves both *structure learning*, i.e., learning the graph topology from data, and *parameter learning*, i.e., learning the actual, local probability distributions from data. There are basically two approaches to structure learning: (i) search-and-score structure learning, and (ii) constraint-based structure learning. Recent approaches are also based on the combination of these two classes of algorithms.

Search-and-score algorithms search for a Bayesian network structure that fits the data best (in some sense). They start with an initial network structure (often a graph without arcs or a complete graph), and then traverse the search space of network structures by in each step either removing an arc, adding an arc, or reversing an arc. Read the paper by Castello and Kočka [1] for a good overview of the principles and difficulties associated with this learning method. Recent search-and-score-algorithms take Markov equivalence into account, i.e., they search in the space of equivalence classes of Bayesian networks and the scoring method they use give the same score for equivalent networks. Bayesian networks with different graph topologies that are included in the same Markov equivalence class represent exactly the same conditional-independence information by d-separation. Examples of search-and-score algorithms are K2 and inclusion-driven learning, usually based on hill-climbing (greedy) search.

Constraint-based algorithms carry out a conditional (in)dependence analysis on the data. Based on this analysis an undirected graph is generated (to be interpreted as a Markov network). Using additional independence tests, this network is converted into a Bayesian network. Constraint-based learning algorithms allow for the easy incorporation of *background knowledge*, i.e., prior knowledge on dependences or independences that hold for the domain under consideration. Examples of constraint-based learning algorithms are PC, NPC, grow-shrink, and incremental association. A good paper discussing the difficulties with constraint-based methods is the paper by Chickering and Meek [2].

For the underlying theory, consult the book "Bayesian Artificial Intelligence" [3] and the two papers referred to above, which can be downloaded from the course assignments webpage. You may consult more specialised books, if required.

In the following Sections 2, 3 and 4 you will find descriptions of the software, datasets and the tasks that you need to perform in order to complete this assignment.

You are free to do something different in the context of Bayesian-network structure learning, e.g., to investigate learning *dynamic* Bayesian networks from temporal data or to work with your own dataset, but it is required to obtain approval from the lecturer in that case.

2 Software

The environment for machine learning that is supposed to be used for this assignment is **R**—a general purpose computing environment for applied statistics. You can download the latest version of R from

`http://cran.r-project.org`

which also contains extensive documentation and introductory manuals to get started. R is not only an interactive computing environment but it is also a scripting language which allows you to write programs. You may also wish to use the new interactive development environment of R called RStudio, which can be downloaded from

`http://www.rstudio.org`

Within R BNLEARN [5] is a package that provides a free implementation of some Bayesian network structure learning algorithms, which appeared in recent literature. Many score functions and conditional independence tests are provided for both independent use and the learning algorithms themselves, as well as basic functions for computing the conditional probability tables. The BNLEARN package provides functions for saving the learnt network structure and parameters in the `.net` format in order, for example, to easily load the resulting network in SAMIAM, and for exporting a fitted Bayesian network to another R package–GRAIN [6]–that uses the junction tree algorithm for inference tasks. Advanced plotting options are provided by the RGRAPHVIZ[1] package. In addition, in this tutorial we will use the built-in R CATOOLS package for evaluating the performance of Bayesian networks using Receiver Operating Characteristic (ROC) curves [7].

The software packages and their dependencies can be installed and loaded within R by:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("RBGL")
> biocLite("Rgraphviz")
> install.packages("gRain")
> install.packages("bnlearn")
> install.packages("caTools")
```

Note that for Unix-like OSes it best is to install with root permission and, thus, to install the packages in the general accessible parts of the directory tree (usually `/usr/local/bin` and `/usr/local/lib/R`).

Once these packages are installed on your computer (which needs to be done only once), they can be used by loading the respective libraries into R:

```
> library(Rgraphviz)
> library(gRain)
> library(bnlearn)
> library(caTools)
```

The library RBGL will be automatically loaded after loading gRain. Alternatively, GRAIN and BNLEARN can be downloaded with the respective manual (in pdf) from:

[1] RGRAPHVIZ has an external dependency on Graphviz, so you need to install the latter on your computer before installing RGRAPHVIZ. You can download the latest version of Graphviz from `www.graphviz.org`

`http://cran.r-project.org/web/packages/gRain/index.html`
`http://cran.r-project.org/web/packages/bnlearn/index.html`

On-line help documentation about the BNLEARN functions, networks and data used within the package is available at:

`http://www.bnlearn.com/documentation/`

Important: Appendix A gives a brief introduction into some of the most important aspects of R, which are required to be studied if you are not familiar with R. Appendix B gives a brief introduction to the use of the Bayesian network packages mentioned above, which will help you complete the assignment.

3 Datasets

Learning is impossible without the availability of data. For this assignment, you will use the following datasets, which are subsequently described and can be downloaded from the course assignments webpage:

- **Iris dataset** (small dataset): real data of iris flower samples. There are no missing values.
- **NHL dataset** (small dataset): clinical data of patients with non-Hodgkin lymphoma (NHL) of the stomach. It is a real dataset obtained from medical researchers and includes missing values (denoted by 'NA', i.e., 'Not Available').
- **Breast cancer dataset** (large dataset): clinical data of patients with breast cancer disease. It has been generated from existing Bayesian network using sampling and is, thus, artificial. There are no missing values.

3.1 Iris dataset

The Iris dataset is introduced by Ronald Fisher in 1936 and consists of 50 samples from each of three species of iris flowers (iris setosa, iris virginica and iris versicolor). Four attributes were measured from each sample—the length and the width of sepal and petal, in centimeters. Based on the combination of these four attributes the goal is often to learn to distinguish among the irises.

3.2 Non-Hodgkin lymphoma (NHL) dataset

The gastric NHL dataset only incorporates variables that are widely used by clinicians in choosing the appropriate therapy for patients [4]. The relevance of most of these variables is supported by literature on prognostic factors in gastric NHL.

First, the information used in the clinical management of primary gastric NHL was subdivided in *pretreatment information*, i.e. information that is required for treatment selection, *treatment information*, i.e. the various treatment alternatives, and *posttreatment information*, i.e. side effects, and early and long-term treatment results for the disease. The most important pretreatment variables are the variable 'clinical stage', which expresses severity of the disease according to a common clinical classification, and histological classification, which stands for the assessment by a pathologist of tumour tissue obtained from a biopsy.

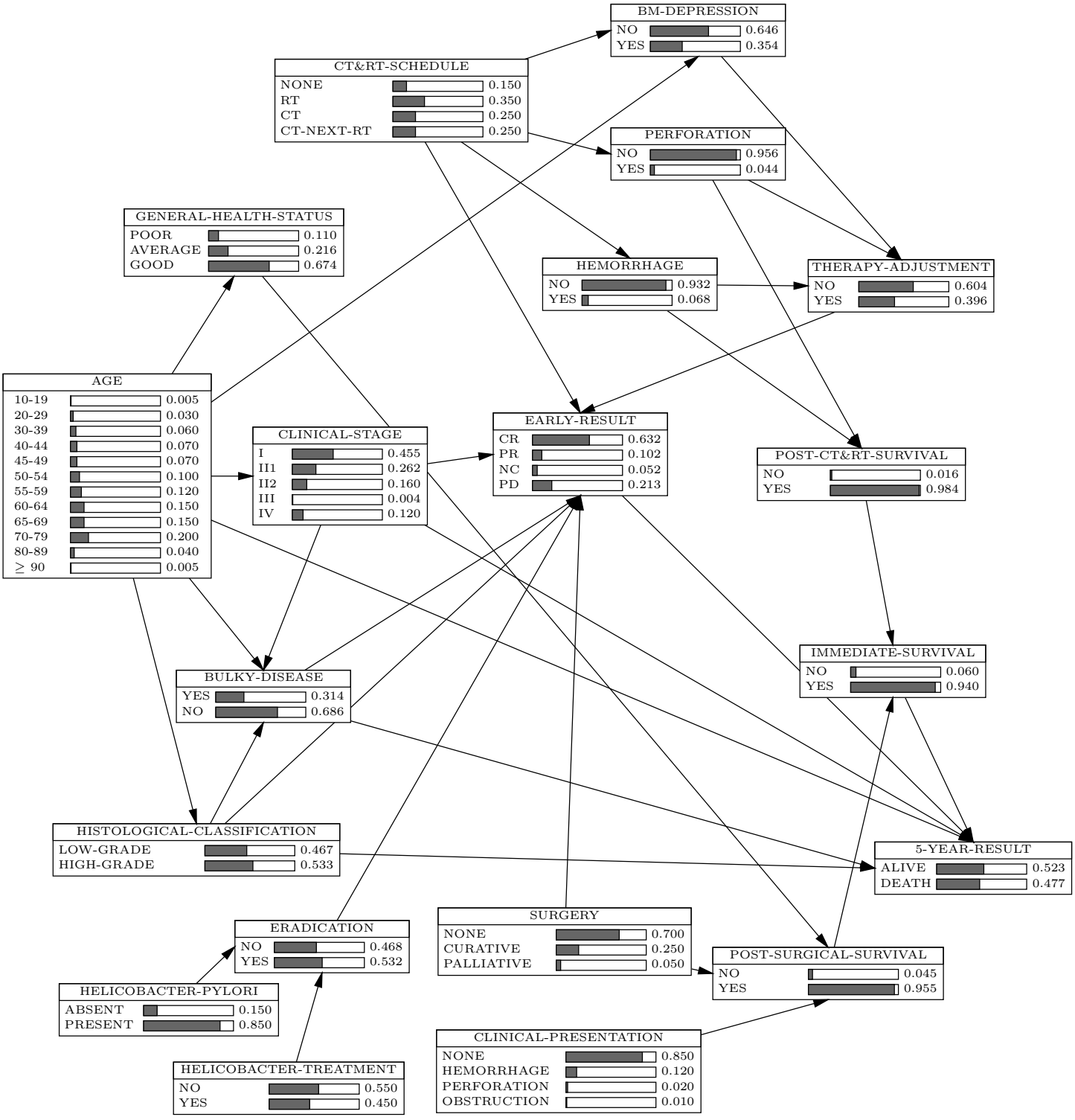

Figure 1: A Bayesian network with prior probability distributions for gastric NHL.

Various treatments are in use for gastric NHL such as chemotherapy, radiotherapy, and a combination of these two, which has been represented as the single variable 'CT&RT-

SCHEDULE' with possible values: chemotherapy (CT), radiotherapy (RT), chemotherapy followed by radiotherapy (CT-next-RT), and neither chemotherapy nor radiotherapy (none). Furthermore, surgery is a therapy with is modelled by the variable 'SURGERY' with possible values: 'curative', 'palliative' or 'none', where curative surgery means total or partial resection of the stomach with the complete removal of tumour mass. Finally, prescription of antibiotics is also possible.

The most important posttreatment variables are the variable 'EARLY RESULT', being the endoscopically verified result of the treatment, six to eight weeks after treatment (possible outcomes are: complete remission – i.e. tumour cells are no longer detectable –, partial remission – some tumour cells are detectable –, no change or progressive disease), and the variable '5-YEAR RESULT', which represents the patient either or not surviving five years following treatment. A Bayesian network with prior probability distributions for gastric NHL is shown in Figure 1.

3.3 Breast cancer dataset

Breast cancer is the most common form of cancer and the second leading cause of cancer death in women. Every 1 out of 9 women will develop breast cancer in her life time. Every year in The Netherlands 13.000 women are diagnosed with breast cancer. Although it is not possible to say what exactly causes breast cancer, some factors may increase or change the risk for the development of breast cancer. These include age, genetic predisposition, history of breast cancer, breast density and lifestyle factors. Age, for example, is the greatest risk factor for non-hereditary breast cancer: women with age of 50 or older has a higher chance for developing breast cancer than younger women. Presence of BRCA1/2 genes leads to an increased risk of developing breast cancer irrespective of other risk factors. Furthermore, breast characteristics such as high breast density are determining factors for breast cancer.

The main technique used currently for detection of breast cancer is mammography, an X-ray image of the breast. It is based on the differential absorption of X-rays between the various tissue components of the breast such as fat, connective tissue, tumour tissue and calcifications. On a mammogram, radiologists can recognize breast cancer by the presence of a focal mass, architectural distortion or microcalcifications. Masses are localised findings, generally asymmetrical in relation to the other breast, distinct from the surrounding tissues. Masses on a mammogram are characterised by a number of characteristics, which help distinguish between malignant and benign (non-cancerous) masses, such as size, margin, shape. For example, a mass with irregular shape and ill-defined margin is highly suspicious for cancer whereas a mass with round shape and well-defined margin is likely to be benign. Architectural distortion is focal disruption of the normal breast tissue pattern, which appears on a mammogram as a distortion in which surrounding breast tissues appear to be 'pulled inward' into a focal point, leading often to spiculation (star-like structures). Microcalcifications are tiny bits of calcium, which may show up in clusters, or in patterns (like circles or lines) and are associated with extra cell activity in breast tissue. They can also be benign or malignant. It is also known that most of the cancers are located in the upper outer quadrant of the breast. Finally, breast cancer is characterised by a number of physical symptoms: nipple discharge, skin retraction, palpable lump.

Breast cancer develops in stages. The early stage is referred as *in situ* ('in place'), meaning that the cancer remains confined to its original location. When it has invaded the surrounding fatty tissue and possibly has spread to other organs or the lymphs, so-called metastasis, it

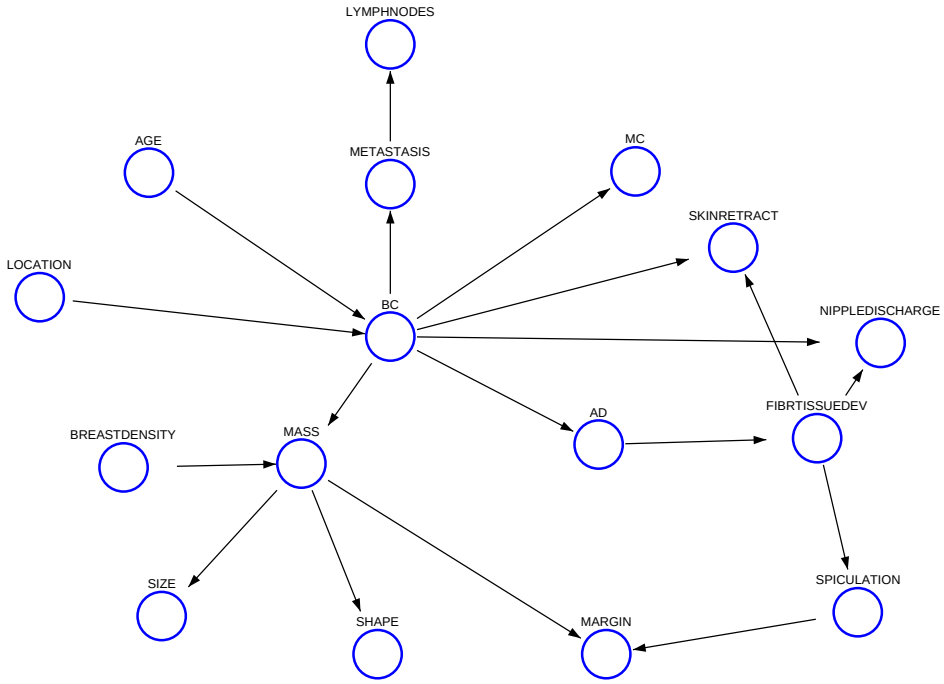

Figure 2: Bayesian network for breast cancer diagnosis.

is referred to as *invasive* cancer. It is known that early detection of breast cancer can help improve the survival rates. Computerized techniques appear to assist medical experts in this respect. Bayesian networks are especially useful given the uncertainty and complexity in mammographic analysis. Figure 2 presents a causal model for breast cancer diagnosis based on the knowledge presented above. All the nodes are assumed to be discrete and the values for each variable are given in Table 1.

3.4 Missing values

Real-world data sets often contain missing values, which may be due to unknown reasons or omissions when data are recorded. Many statistical and learning methods cannot deal with missing values directly, and there are two main approaches to cope with this problem before learning and testing is applied to:

- **Remove the cases with missing data**. However, this approach should be applied with caution as it can result in the loss of a large amount of valuable data, thus leading to a decrease in the robustness of the models learnt. In R there are various ways to obtain the set of complete/missing cases in a dataset, as shown in the appendix Section A.4.

- **Filling (imputing) missing data**. This means that the missing value is replaced with an estimate of the actual value. For instance, by *mean imputation* the missing value is replaced by the mean of the variable in question, whereas by *multiple imputation*, rather than a single imputed value, multiple ones are derived from a prediction equation. There are various R packages that contain functions for imputing missing values; see Section A.5 in the appendix for one of them.

Table 1: Definition of the variables from the breast cancer dataset

Node	Values
Age	< 35, $35 - 39$, $50 - 74$, > 75 (in years)
Location	UpOut, UpIn, LowOut, LowIn (in quadrants)
Breast Cancer	No, Invasive, InSitu
Density	Low, Medium, High
Mass	No, Malignant, Benign
Size	< 1, $1 - 3$, > 3 (in cm)
Shape	Other, Round, Oval, Irregular
Margin	Ill-defined, Well-defined
Architectural Distortion	Yes, No
Fibrous Tissue Development	Yes, No
Skin Retraction	Yes, No
Nipple Discharge	Yes, No
Microcalcification	Yes, No
Metastasis	Yes, No
Lymph Nodes	Yes, No
Spiculation	Yes, No

4 Practical assignment

Learning Bayesian networks from data is far from easy!
Learning Bayesian networks is an active area of research and most software that is available is experimental in nature. Thus, do not expect perfect software packages in this area and be prepared to experiment with the software. Furthermore, the learning problem suffers from combinatorial explosion; so, do not expect that each dataset made available can be handled easily.

In the next sections, it is described (in numbered tasks in italics) what is expected from you to complete this assignment.

4.1 Comparison of two learning algorithms

For this part of the assignment you have to investigate the effect of two data aspects—discretisation and size—and compare the results obtained by a search-and-score- and a constraint-based learning algorithm

(1) Investigate what is the effect of data discretisation on the structure of the learnt Bayesian network, e.g., in terms of number of arcs, changes in arc direction, etc., by considering three discretisations of the continuous features in the iris dataset with different number of bins (see Section B.3). Do this for both classes of learning algorithms and compare the results.

(2) Investigate what is the effect of the size of the dataset on the structure of the learnt Bayesian network by considering three subsets of the breast cancer dataset with different

number of samples. Do this for both classes of learning algorithms and compare the results.

4.2 Comparison with a manually constructed Bayesian network

For this part of the assignment you need to compare the learnt and manually constructed Bayesian networks (shown in Figure 1) for non-Hodgkin lymphoma (NHL) of the stomach.

Important remarks about the NHL dataset:

- The values of the variables in the NHL dataset are integers but they represent discrete (not continuous) data. Make sure to load the dataset as discrete in R, i.e., the column classes should be of type `factor` (see the end of section A.4 in the appendix).
- The NHL dataset contains missing data, which is problematic for many learning algorithms, including the ones implemented in the BNLEARN package. To tackle this problem fill in the missing data by choosing a method, for example, as discussed in Section 3.4. Motivate your choice.
- The number of variables in the NHL dataset is less than that in the manually constructed Bayesian network in Figure 1, which is a realistic situation. Nevertheless, you can still compare the learnt and manually constructed networks for the common parts as well as for the differences observed.

(3) Define measures that can be used to determine the quality of a learning algorithm in terms of a known Bayesian network structure. Motivate the definition of these measures.

(4) Use these measures in order to evaluate the quality of both classes of learning algorithms for the NHL dataset.

(5) Compare the marginal probability distributions of the learnt NHL Bayesian network with those of the manually constructed network.

4.3 Diagnostic Bayesian network structures

The breast cancer dataset includes the variable 'Breast Cancer' that is used to classify patients into one of three different categories:

1: *No* breast cancer

2: *Invasive* breast cancer

3: *Insitu* breast cancer

As the class variable acts as the *output* of a Bayesian network that models the random variables contained in the dataset, where the other variables represent the input (also called evidence or features), if present, a diagnostic Bayesian network has often a *special* structure. Popular special Bayesian-network structures for diagnostic applications are naïve Bayesian networks (NBN) and tree-augmented Bayesian networks (TAN) (see Lecture 3 "Building Bayesian Networks" for examples on these types of networks).

(6) Learn a Bayesian network from the breast cancer dataset using a search-and-score or constraint-based algorithm.

(7) Develop a special purpose (e.g., NBN or TAN) Bayesian network (see Section B.2 in the appendix).

(8) Compare the Bayesian network obtained by step (6) and the special purpose Bayesian network from (7) with the manually constructed network in Figure 2 in terms of network structure, goodness of fit scores (e.g., likelihood) and accuracy using measures such as misclassification error and area under the ROC curve from cross-validation (see Section B.2 in the appendix).

4.4 Results

Write a comprehensive report in which for tasks (1) to (8) you:

- ***motivate your choices of algorithms and measures***
- ***present and analyse the results you have obtained. In the analysis, please describe your insights, explain whether the obtained results confirm what you expected or differ from that, and why you think that is the case.***
- ***conclude briefly what you have learnt about learning of Bayesian networks based on this assignment***

If you have written some programs in R for this assignment, please include them as appendix in the report. The report should be submitted via email to `peterl@cs.ru.nl` ***and it will be assessed as part of the overall course assessment.***

The main criteria for evaluation of the assignment are:

- ***Completed work on all the tasks.***
- ***Completeness and clarity of the report (presentation of the results, depth of the analysis, motivation of the choices made).***
- ***Out-of-the-box thinking (interesting ways for performing the tasks, suggestions for extending the work)***

.

References

[1] R. Castelo and T. Kočka, On inclusion-driven learning of Bayesian networks, *Journal of Machine Learning Research*, 527–574, 2003.

[2] D.M. Chickering and C. Meek, On the incompatibility of faithfulness and monotone DAG faithfulness. *Artificial Intelligence*, 170, 653-666, 2006.

[3] K.B. Korb and A.E. Nicholson, *Bayesian Artificial Intelligence*, Chapman & Hall: Boca Raton, 2004.

[4] P.J.F. Lucas, H. Boot, and B.G. Taal. Computer-based decision-support in the management of primary gastric non-Hodgkin lymphoma. *Methods of Information in Medicine*, 37, 206–219, 1998, `www.cs.ru.nl/~peterl/mim.pdf`.

[5] M. Scutari. Learning Bayesian networks with the bnlearn R package, *Journal of Statistical Software*, 35(3), 1–22, 2010, `www.jstatsoft.org/v35/i03/paper`.

[6] Søren Højsgaard. Graphical independence networks with the gRain package for R, *Journal of Statistical Software*, 46(10), 1–26, 2012, `www.jstatsoft.org/v46/i10/paper`.

[7] J.A. Hanley and B.J. McNeil. The meaning and use of the area under a Receiver Operating Characteristic (ROC) curve. *Radiology*, 143, 29–36, 1982, `http://www.medicine.mcgill.ca/epidemiology/hanley/software/Hanley_McNeil_Radiology_82.pdf`.

Appendix

A Introduction to R

R is a publicly and freely available interactive programming environment for mathematical and statistical computing very similar to Matlab. As an interactive environment, it shares features with programming languages such as Lisp, Python, and Prolog. Even though it is possible to write programs in R, most of the time it is used with packages that have been developed for particular purposes. The amount of programming needed to use these packages is usually limited.

A.1 Help facilities

You can find descriptions of the packages available in your installation of R using

```
> help.start()
```

which will start up a browser. You can also obtain a description of functions in R by using a question mark (but here you need to know the name of the functions):

```
> ?read.csv
```

And you will see that `read.csv` is the function from the `utils` package of R used to read tables with data in comma-separated format:

```
------------------------------------------------------------------------------
read.table                package:utils                R Documentation

Data Input

Description:

     Reads a file in table format and creates a data frame from it,
     with cases corresponding to lines and variables to fields in the
     file.
```

```
Usage:

     read.table(file, header = FALSE, sep = "", quote = "\"'",
                dec = ".", row.names, col.names,
                as.is = !stringsAsFactors,
                na.strings = "NA", colClasses = NA, nrows = -1,
                skip = 0, check.names = TRUE, fill = !blank.lines.skip,
                strip.white = FALSE, blank.lines.skip = TRUE,
                comment.char = "#",
                allowEscapes = FALSE, flush = FALSE,
                stringsAsFactors = default.stringsAsFactors(),
                fileEncoding = "", encoding = "unknown", text)

     read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
              fill = TRUE, comment.char="", ...)

etc.

------------------------------------------------------------------------
```

There are many other functions in the `utils` package. Have a look at some of these utilities by clicking on `utils` in the browser help facility.

Extended information about the basic and advanced commands in R with related examples can be found online, for example, at:

`http://cran.r-project.org/doc/manuals/R-intro.pdf`
`http://www.statmethods.net/`
`http://www.r-tutor.com/r-introduction`

A.2 Loading packages into R

After starting R:

```
> library(bnlearn)
> library(gRain)
```

Alternatively, if you use RStudio you can simply select the package you wish to load under tab "Packages" on the right panel.

Some other packages that are needed are automatically selected. However, sometimes you also need to install certain graphical libraries or mathematical libraries. This is mostly a matter or just adding these from your distribution.

A.3 Datatypes

R includes many common datatypes, called *modes* in the language, one will find in other programming languages; syntax and semantics are similar. R is a weakly typed language; types are usually determined from the objects supplied. We give some example:

- `vectors` can be created by the function `c` (concatenation):

```
> v = c(1,2,3)
> v[1]
[1] 1
> v[3]
[1] 2
> v[1:2]
[1] 1 2
```

Because here a vector of numbers is created, the associated mode is '`numeric`'. Here `1:2` is a shortcut for the vector $[1, 2]$ ($[1 : n]$ gives you the sequence of numbers $1, 2, \ldots, n$, $n \geq 1$), which here acts as an index.

- `lists` can be created by the function `list`:

```
> x = list(1, 2, 3, 4)
> x[[2]]
[1] 2
```

Element of list are accessed by means of double brackets `[[ · ]]`. It is also possible to give the components of a list a name:

```
> x = list(one = 1, two = 2, three =  3)
> x[2]
$two
[1] 2
> x[[2]]
[1] 2
```

Note the difference here between `x[2]` and `x[[2]]`. Alternative, it is now also possible to access list components by means of the component's name (field):

```
> x$two
[1] 2
```

or

```
> x[["two"]]
[1] 2
```

And this makes list very similar to structures in other languages.

- `matrix` and `array` are similar to `vector`, but multidimensional. One can concatenate vectors, lists, and arrays in different ways by calling `c` (concatenation), `rbind` (row concatenation), or `cbind` (column concatenation):

```
> x = c(1, 2, 3, 4)
> y = c(5, 6, 7, 8)
> c(x, y)
```

```
[1] 1 2 3 4 5 6 7 8
> rbind(x, y)
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    5    6    7    8
> cbind(x, y)
     x y
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
```

- `factor` is an important datatype for statistical applications, and, therefore, also in Bayesian networks. The idea is to define the (finite) domain of a variable based on the values that are available in the dataset:

```
> s = c("yes","no","yes","no","yes","no")
> s
[1] "yes" "no"  "yes" "no"  "yes" "no"
> a = factor(s)
> a
[1] yes no  yes no  yes no
Levels: no yes
```

First, we assign a vector of yes/no string values to the variable `s`. Next, we turn this vector into a factor by calling the function `factor`. The variable `a` is now a factor variable that has {`no`, `yes`} as a domain; 'no' and 'yes' are called the *levels* of the variable. Thus, '`a`' can now be used as a random variable.

A.4 Reading in datasets

A dataset in R, where it is called a *data frame*, has the form of a file with on the first row the names of the variables (or attributes), followed by the actual records or *cases*. See Table 2 for an example.

Name	,	Age	,	Gender	,	Income
John	,	20	,	male	,	20000
Petra	,	30	,	female	,	80000
Pierre	,	18	,	male	,	NA

Table 2: Example of R dataset (called data frame in R).

In various R packages, including the package `datasets`, there is a large number of internal datasets, which can easily be loaded in R by using the `data` function, e.g.,:

```
> data(namedata)
```

where `namedata` is the name of the dataset.

Data files that are external of R can be loaded in different ways, as discussed next. Let us assume that the data in Table 2 are stored in the external file with name `exdata`. The data are then loaded into R as follows:

```
> data = read.table("exdata", sep = ",")
```

The first argument of `read.table` is the name of the file; `sep = ","` means that a comma acts as separator of the elements in the data frame. It is also possible to use any other separator. If a comma is used, then

```
> data = read.csv("exdata")
```

will yield exactly the same result. The content of the variable `data` in R is now as follows:

```
> data
    Name Age   Gender  Income
1   John  20     male   20000
2  Petra  30   female   80000
3 Pierre  18     male      NA
```

Data frames are similar to two-dimensional arrays. One can access elements from a data frame by array indices:

```
> data[1,2]
[1] 20
> data[2,3]
[1]   female
Levels:   female   male
```

Note that R has automatically turned the variable 'Gender' into a factor with two levels: 'female' and 'male'. We can also access entire rows or columns from data frames (and in facts also from arrays):

```
> data[1,]
 Name Age Gender  Income
1 John  20   male   20000
> data[,3]
[1]   male     female   male
Levels:   female   male
```

It is also possible to extract the names of the variables from the data frame:

```
> names(data)
[1] "Name"   "Age"    "Gender" "Income"
```

Sometimes R does not know whether a variable in a dataset is a factor or not. This in particularly happens when the values in the records are numbers. Numbers such as 0, 1, 2, 3 might be interpreted as real values, and one may decide to define a normal (Gaussian) distribution on a corresponding continuous variable. However, 0, 1, 2, and 3 can also be the values of a discrete variable. One can convert a variable into a factor as follows:

```
> data = read.csv("exdata")
> data[,4]
[1] 20000 80000    NA
> data[,4] = factor(data[,4])
> data[,4]
[1] 20000 80000 <NA>
Levels: 20000 80000
```

Thus by the assignment, the `Income` variable has become a factor with two levels $\{20000, 80000\}$. Converting all the variables of a data frame into factors can be done by means of a for loop:

```
for (j in 1:length(data[1,])) {
  data[,j] = factor(data[,j])
}
```

Alternatively, the type of variables can be specified when loading the data:

```
> data = read.csv("exdata", colClasses = c(rep("factor", N))),
```

where `N` is the number of columns (variables) in `exdata`. Here the function `rep` replicates `N` times `"factor"`.

A.5 Coping with missing values

In `exdata` 'NA' means 'Not Available', i.e., a *missing value. Even though Table 2 suggests otherwise, it is important that there are no blanks between the comma and the 'NA' value in the file.* One can check on the presence of missing values in the data by `is.na`:

```
> is.na(data)
      Name   Age Gender Income
[1,] FALSE FALSE  FALSE  FALSE
[2,] FALSE FALSE  FALSE  FALSE
[3,] FALSE FALSE  FALSE   TRUE
```

It is also possible to extract whether or not each record (case) in the data is complete, i.e., it does not contain missing values:

```
> complete.cases(data)
[1]  TRUE  TRUE FALSE
```

This result can then be used to select the records (cases) without missing values from the data:

```
> cm = complete.cases(data)
> cdata = data[cm, ]
> cdata
   Name Age Gender Income
1  John  20   male  20000
2 Petra  30 female  80000
```

The `na.omit` function yields the same results without the need of intermediate computation:

```
> na.omit(data)
   Name Age Gender Income
1  John  20   male  20000
2 Petra  30 female  80000
```

A more advanced way for coping with missing values is provided, for example, in the R package Hmisc[2], whose `impute` function will replace missing values with the median non-missing value for continuous variables, and with the most frequent category for categorical (factor) or logical variables. One can also specify other statistical functions for use with continuous variables instead of the default `median`, e.g., `mean`, or constants or randomly sampled values to insert for numeric or categorical variables.

```
> data_complete = data
> for (j in 1:length(data[1,])) {
    data_complete[,j] = impute(data[,j])
  }
```

A.6 Programming in R

The R language has a C-like syntax, but is weakly typed as most scripting languages. It includes the usual statement of a scripting language. The *assignment statement* (which has already been used in this document) supports two different assignment operators:

```
var <- expr
```

and

```
var = expr
```

are equivalent. The language includes the following control structures:

```
if (cond) expr
if (cond) cons.expr  else  alt.expr

for (var in seq) expr
while (cond) expr
repeat expr
break
next
```

As an example, consider:

```
> for(x in 1:5) print(x)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

[2] http://cran.r-project.org/web/packages/Hmisc

An *expr* can be a block of statements, which are combined using curly brackets { } as in C.

Functions are defined as follows:

```
functionName <- function(arg1, arg2, ..., argn)
{
  <body>
}
```

A.7 Operating system interface and history

You can obtain your working directory from R by:

```
> getwd()
[1] "/home/john/Practical"
```

You can also change your working directory by:

```
> setwd("/home/john")
> getwd()
[1] "/home/john"
```

You can read in programs into R by the `source` command:

```
> source("test.R")
```

You can write the output produced by R to a file by `sink`:

```
> sink("output")
```

If you save your working session, the result is saved in `.Rhistory` and `.RData`. Remove them if you don't want these anymore.

B Learning and evaluating BNs in R using the Visit-to-Asia data

The Visit-to-Asia problem concerns the three disorders—tuberculosis, lung cancer and bronchitis—whose network is shown in Figure 3. We use this problem to present the basics for Bayesian network learning and evaluation in R. Have a look at Table 3 for the corresponding variable names between the data and the network.

Table 3: Correspondences between the asia dataset and the Bayesian network shown in Figure 3.

Name	Explanation
D	Dyspnoea?, a two-level factor with levels 'yes' and 'no'
T	Has tuberculosis, a two-level factor with levels 'yes' and 'no'
L	Has lung cancer, a two-level factor with levels 'yes' and 'no'
B	Has bronchitis, a two-level factor with levels 'yes' and 'no'
A	Visit to Asia?, a two-level factor with levels 'yes' and 'no'
S	Smoker?, a two-level factor with levels 'yes' and 'no'
X	Positive (chest) X-ray?, a two-level factor with levels 'yes' and 'no'
E	tuberculosis or lung cancer, a two-level factor with levels 'yes' and 'no'

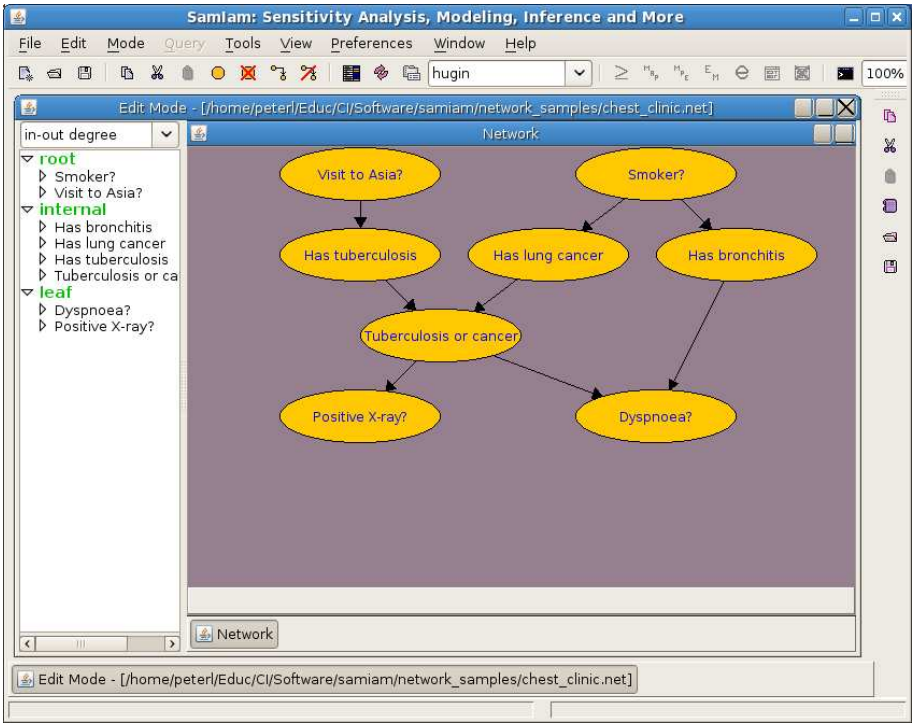

Figure 3: Screenshot of SamIamwith the Visit-to-Asia network.

B.1 Learning a Bayesian network

Make sure that you loaded the bnlearn and gRain libraries into R!

Having loaded the data in R by:

```
> data(asia)
```

you can use various algorithms for learning the network structure. One of the simplest score-based learning algorithm is the hill-climbing (hc) (see slides of the lectures)

```
> net = hc(asia)
```

You can plot the network learnt by:

```
> plot(net)
```

and via Menu ≫ Plots ≫ Zoom plot... in RStudio you can get a better insight in the plot. You can save the network plot as an image , e.g. to use it in the report, via Plots ≫ Save Plot as Image... .

Missing arcs can be added to the learnt `net` structure using the `set.arc` function.

```
> net = set.arc(net, "A", "T")
```

The network's probability distribution can be learnt using the `bn.fit` function and then inspect it:

```
> fitted = bn.fit(net,asia)
> fitted
```

To see how well the Bayesian network model just learnt fits the data you can use the `score` function:

```
> score(net, asia)
```

Note also that different measures can be selected in the `score` function (use `?score` for that).

BNLEARN includes many other structure learning algorithms, including a whole set of algorithms for constraint-based structure learning such as grow-shrink (gs). In the last case, you may get partially directed networks, i.e., networks with directed *and* undirected edges. You can orient the undirected egdes manually or by using `cextend`:

```
> netgs = gs(asia)
> netgs_directed = cextend(netgs)
> netgs_directed

  Bayesian network learnt via Constraint-based methods

  model:
   [A][S][T][X][L|S][B|S][E|T:L][D|B:X]
  nodes:                                8
  arcs:                                 6
    undirected arcs:                    0
    directed arcs:                      6
  average markov blanket size:          2.00
  average neighbourhood size:           1.50
  average branching factor:             0.75

  learning algorithm:                   Grow-Shrink
  conditional independence test:        Mutual Information (discrete)
  alpha threshold:                      0.05
  tests used in the learning procedure: 64
  optimized:                            TRUE
```

B.2 Bayesian classifiers and ROC analysis

One way of using a Bayesian network is as a method to classify new cases. Basically, classification is mapping the evidence from the case to a case label or category, and there are many other methods, such as classification tree and rules, and neural network, that can also be used for this purpose. Given a Bayesian network with joint probability distribution P and variable C of interest (often called the 'class variable'), the problem is to determine a value of C with maximum probability, or:

$$c_{\max} = \arg\max_{c} P(c \mid \mathcal{E})$$

with $\mathcal{E} \subseteq \{E_1, \ldots, E_m\}$ being the evidence (measurements, observations) for the case at hand.

The naïve Bayesian classifier is a Bayesian network having the structure shown in Figure 4. As it has a fixed structure, only the parameters need to be learnt. See Lecture 3, slides 21–25 for an example of a naïve Bayesian classifier. In BNLEARN a naïve Bayes classifier can be

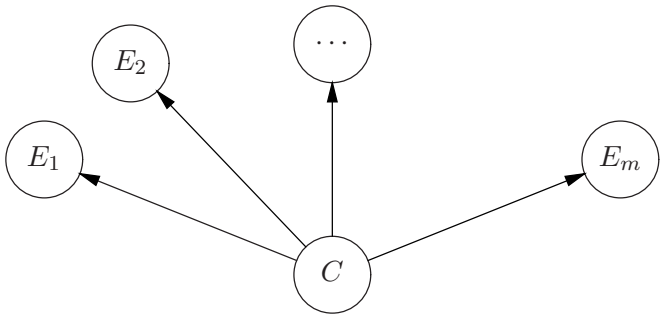

Figure 4: Naïve Bayesian network that can be easily used as classifier.

learnt by using the `naive.bayes` function and by choosing the class (parent) variable, e.g., "Has tuberculosis" (`T`), while the remaining variables are the effects (children).

```
> netnb = naive.bayes(asia, "T", names(asia)[c(1:2,4:8)])
> plot(netnb)
```

Cross validation. Evaluation of the quality of a Bayesian classifier can be done by means of *cross validation*. Using the same dataset for learning and testing is not a good idea, as then the evaluation results are likely to be (very) good due to *overfitting*. Cross validation involves learning the network (parameters) on all parts of the data, or *folds*, but one, and evaluating the resulting network on the single fold that was left out in the learning process. This is done repeatedly from fold to fold. For example, in 10-fold cross validation, a Bayesian network model is first learnt from equally sized folds 2 to 10, then tested on the first fold (also of equal size). Next, the model is learnt from fold 1, 3 to 10, then tested on the second fold, etc.

Consider again the Visit-to-Asia problem. You can perform 3-fold cross-validation (cv) learning of the `net` structure, which was previously learnt using the hill-climbing algorithm and the prediction error for one of the diagnostic variables (`T, L, B` or `E`) as a loss function, e.g.:

```
> netcv = bn.cv(asia, net, loss = "pred", k = 3,
                        loss.args = list(target="T"), debug = TRUE)
```

Setting the debug option to `TRUE` displays the results for the loss function per fold. The resulting structure `netcv` is a list containing elements described in

```
http://www.bnlearn.com/documentation/man/bn.kcv.class.html
```

Using the 3-fold cross-validation results for `netcv`, you can predict the values for the test cases in each fold k, $k = 1, 2, 3$ by:

```
## Convert the fitted BN to gRain object
> netcvfitk = as.grain(netcv[[k]]$fitted)

## Create the k-th test subset
```

```
> asia_testk = asia[netcv[[k]]$test, ]

## Compute posterior probabilities P(response | predictors) for the k-th subset
> pred_testk = predict(netcvfitk, response = c("T"), newdata = asia_testk,
                predictors = names(asia_testk)[-3], type = "distribution")
```

Here the parameter `[-3]` is the index of the response variable (`"T"`) in the testset that needs to be excluded from the list of predictors, whereas `type` indicates the type of output–the probability (`"distribution"`) or the most probable class (`"class"`). Note that the `predict` function may give you the following message for a particular observation `n`:

```
Finding for row n has probability 0 in the model.  Exiting...
```

A simple solution is to remove the observation from the dataset by:

```
> asia_testk = asia_testk[c(1:n-1, n+1:N), ]
```

where `N` is the number of rows in the dataset obtained from `nrow(asia_testk)`, and then execute the `predict` function.

ROC analysis. Another popular way to evaluate the quality of Bayesian classifiers is Receiver Operating Characteristic (ROC) analysis. This technique was originally developed in the UK during World War II to help British pilots in making a distinction between enemy (German) aeroplanes and their own aeroplanes: too many British pilots attacked their colleagues. For this part, you need to load the caTools package in R.

To compute the network performance using ROC curve you need to:

```
## Create the complete testset by concatenating the 3 test subsets
> asia_test = rbind(asia_test1,asia_test2,asia_test3)

## Create the complete set of predicted probabilities for
## the value "yes" of the class variable "T" by concatenating
## the results for all 3 test subsets
> pred_test = data.frame(c(pred_test1$pred$T[ ,2], pred_test2$pred$T[ ,2],
                                                   pred_test3$pred$T[ ,2]))

## Compute and plot the ROC curve
> colAUC(pred_test, asia_test[ ,3], plotROC = TRUE)
```

Note that the parameter `[ ,3]` is the column number of the target variable in the testset, which you used to build `netcv`.

B.3 Data discretisation

Discretisation of data has been studied for more than 20 years as one of the major preprocessing steps in data analysis. Its goal comprises the transformation of continuous variables into a finite number of discrete values, or bins, to facilitate: (i) the improvement in classification

performance, (ii) the induction process of a classifier, or (iii) the interpretability of the models learnt. The BNLEARN package contans the `discretize` function, which allows automatic discretisation. For more information see:

`http://www.bnlearn.com/documentation/man/preprocessing.html`

B.4 Reading and saving .net files

You can use .net files within samiam or genie, so it is convenient to be able to load or save these in R as well. There are a number of options, which are very similar:

(1) use the bnlearn `read.net` function;

(2) use the gRain to bnlearn integration from the gRain and bnlearn packages;

(3) use the deal to bnlearn integration (deal is a package that is somewhat related to bnlearn)

We only illustrate options (1) and (2):

```
> net1 = read.net("nhl.net")
```

Alternatively:

```
> net2 = loadHuginNet("nhl.net")
```

If `loadHuginNet` is used then the result is an 'independent network' and this needs to be converted to a bn-fit object (a Bayesian network with probability tables) first:

```
> fit.net2 = as.bn.fit(net2)
```

Type

```
> fit.net2
```

to see how it looks like.

You now have a version of the network, which is not yet a bnlearn bn-class object but a bn-fit object (i.e. with parameters). Actually, `read.net` has yielded a bn-fit object directly, so the conversion illustrated above is not needed. You can now convert both objects to a bn-class object in bnlearn by calling "`bn.net`":

```
> bn1 = bn.net(net1)
> bn2 = bn.net(fit.net2)
> bn1

Random/Generated Bayesian network

  model:
   [Age][HelicobacterTreatment][ClinicalPresentation][Surgery][CTRTSchedule]
   [H.Pylori][ClinicalStage|Age][HistologicalClassification|Age]
   [Eradication|HelicobacterTreatment:H.Pylori][Hemorrhage|CTRTSchedule]
   [Perforation|CTRTSchedule][GeneralHealth|Age][BMDepression|Age:CTRTSchedule]
```

```
  [PostsurgicalSurvival|ClinicalPresentation:Surgery:GeneralHealth]
  [TherapyAdjustment|Hemorrhage:Perforation:BMDepression]
  [PostCTRTSurvival|Hemorrhage:Perforation]
  [BulkyDisease|Age:ClinicalStage:HistologicalClassification]
  [EarlyResult|ClinicalStage:Surgery:TherapyAdjustment:
              CTRTSchedule:HistologicalClassification:
              Eradication:BulkyDisease]
  [ImmediateSurvival|PostsurgicalSurvival:PostCTRTSurvival]
  [FiveyearResult|Age:ClinicalStage:EarlyResult:
                  HistologicalClassification:BulkyDisease:ImmediateSurvival]
nodes:                                  20
arcs:                                   35
  undirected arcs:                      0
  directed arcs:                        35
average markov blanket size:            6.90
average neighbourhood size:             3.50
average branching factor:               1.75

generation algorithm:                   Empty
```

The networks `bn1` and `bn2` are similar. You can also save networks in .net format using the `write.net` or `saveHuginNet` function. In the last case, you have to convert from bnlearn format to gRain format. See the gRain integration of bnlearn.