Inference of networks
Assignment 1 (of 2)
Network Biology 2018

Lorenz Wernisch `lorenz.wernisch@mrc-bsu.cam.ac.uk`

Submit by 15 March 2018

1 Introduction

I would like to ask you to work through a few exercises by developing some R code to answer questions in the following sections. You can either collect all the code in an .R file and collect your results and reasoning in a separate text document. Alternatively, you might want to use a system like `knitr`, see

`https://joshldavis.com/2014/04/12/beginners-tutorial-for-knitr/`

which I used to write this document. I attach the source text `netinfer.Rnw` and `netinfer.R` for you as an example of how you can use `knitr`. On Linux I compiled these files with

```
Rscript -e "library(knitr); knit('netinfer.Rnw')"
latex netinfer.tex; dvips -Ppdf -o netinfer.ps netinfer.dvi
ps2pdf netinfer.ps netinfer.pdf
```

`knitr` mixes latex with R code in form of an .Rnw document. There are two ways to mix latex with R code. Either you write the R code directly into the .Rnw file:

```
eta <- c(2,2)
eta
```

```
## [1] 2 2
```

Or you keep some R code in a separate .R file `netinfer.R` which is loaded in the setup section at the beginning of this .Rnw document. It contains code sections labelled for example by `## ---- load_network_data`. These labelled code sections can be initiated from within the .Rnw file as follows

```
load("small_network_1.rda")
dat1[1:3,]
```

```
##   A B C
## 1 0 1 0
## 2 1 1 1
## 3 0 1 0
```

```
load("small_network_2.rda")
dat2[1:3,]
```

```
##   A B C
## 1 1 1 1
## 2 0 0 0
## 3 1 1 1
```

One can recall the data:

```
dat1[1:2,]
```

```
##   A B C
## 1 0 1 0
## 2 1 1 1
```

As you can see later code sections in the .Rnw file can use data generated in earlier code sections. The advantage of `knitr` is that apart from the code it also shows all output and plots and makes it easy for a reader to follow your code.

I am aware that the following descriptions might contain mistakes or might not be clear enough. Do not hesitate to email me in case of queries. The following sections are worth a total of 100 marks.

1.1 R packages

In the following I ask you to look in quite a few R packages. They are usually easy to install. However, that requires you have set up your R system in a way that allows you to do that. Packages from the CRAN repository can be installed via

```
install.packages(``bnlearn'')
```

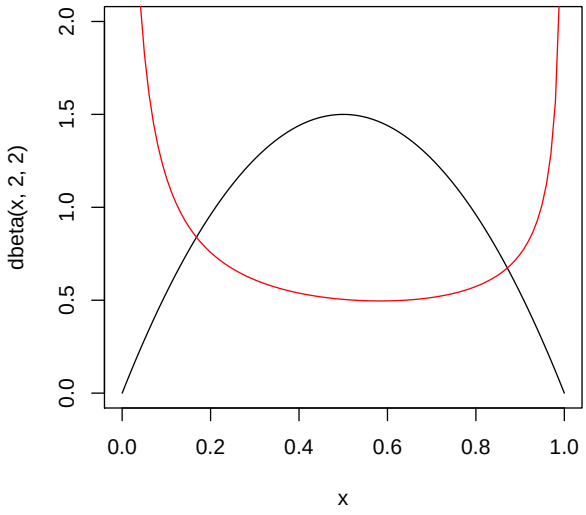
If a package is not on CRAN it is most likely available from Bioconductor, for example to install `Rgraphviz`

```
source("https://bioconductor.org/biocLite.R"); biocLite("Rgraphviz")
```

2 Finding priors for the beta distribution [10 marks]

It's easy to plot the beta distribution

```
x <- seq(0,1,len=100)
plot(x,dbeta(x,2,2),ty="l",ylim=c(0,2))
lines(x,dbeta(x,0.3,0.5),col="red")
```



We assume a beta prior for the probability that a coin lands on head (1) instead of tail (0). Use the minimisation function `optim` on a combination of `pbeta`, and `abs` as well as the properties of the beta distribution from the lecture to find prior beta parameters η_0 and η_1 to express the following prior beliefs.

1. It's a bit unlikely (only 5%) that $\theta \leq 0.25$, and unlikely (5%) that it's above 0.75.
2. The mode (highest point) is 0.4, and the probability is 0.1 for $\theta \leq 0.3$.

Using the second prior, plot prior, likelihood and posterior beta distribution for the coin toss result: 011100101101.

3 Infer a three variable Bayesian network [25 marks]

Load "expression data" on three "genes" each with levels 1 and 2

```
load("small_network_1.rda")
dat1[1:3,]
```

```
##   A B C
## 1 0 1 0
## 2 1 1 1
## 3 0 1 0
```

```
load("small_network_2.rda")
dat2[1:3,]
```

```
##   A B C
## 1 1 1 1
## 2 0 0 0
## 3 1 1 1
```

Get the contingency table

```
table(dat1)
```

```
## , , C = 0
##
##    B
## A    0  1
##   0 22 42
##   1  0  7
##
## , , C = 1
##
##    B
## A    0  1
##   0  6  1
##   1 16  6
```

For three variables there exist quite a few possible Bayesian networks. How many? We will focus only on a few here. In the notation of Lecture 2 compute the likelihood $P(D \mid G_i)$ of the Bayesian networks G_i represented by the following skeleton structures:

The first structure represents independent variables, the second all four Bayesian networks that can be obtained by orienting both edges in both directions. The third structure represents two networks. Using likelihood equivalent priors reduces the number of structures that need checking. Following the notation from the lecture for the beta distribution, assume $\eta_0 = \eta_1 = 1$ if you consider a child with two parents, $\eta_0 = \eta_1 = 2$ when you consider a child with one parent, and $\eta_0 = \eta_1 = 4$ when you consider a (parentless) top or isolated variable.

As a starting point write an R function for the logarithm of the probability of counts of 0 and 1 given beta priors:

$$P(n_9, n_1 \mid \eta_0, \eta_1) = \frac{\Gamma(\eta_0 + \eta_1)}{\Gamma(\eta_0)\Gamma(\eta_1)} \frac{\Gamma(n_0 + \eta_0)\Gamma(n_1 + \eta_1)}{\Gamma(n_0 + n_1 + \eta_0 + \eta_1)}$$

```
lp.beta <- function(n,eta) {
  ## n vector of counts, eta vector of priors
  ## ...
  ## return log P(n|eta)
}
```

Start with computing the log likelihood for the independence model just as in the lecture. You might find the `table` R function useful to extract contingency tables:

```
table(dat1[,"A"])
```

```
##
##  0  1
## 71 29
```

```
table(dat1[,c("A","C")])
```

```
##    C
## A    0  1
##   0 64  7
##   1  7 22
```

Now it is a matter of applying the `lp.beta` function to various rows (or columns) of suitably chosen contingency tables according to a decomposition into conditional probabilities following a Bayesian network.

There is an easy test whether you are on the right track: for equivalent networks (eg the two directions of the edge in the third graph) and likelihood equivalent priors you should get the same log likelihood. If not, something is not quite right. If you do get the same log likelihood values, try changing the priors and observe slightly different log likelihoods for the two edge directions.

If you wanted to use a likelihood equivalent prior of $\eta_0 = \eta_1 = 1$ for parentless variables, what would you have to use for variables with one, and for variables with two parents? Test it on the third graph.

Which of the networks represents the data best?

Plot the posterior for $P(B = 1 \mid A = 1)$ using the third graph.

3.1 R packages for Bayesian networks [10 marks]

The R package `deal` implements (among others) the above Bayesian scoring algorithms and provides inference functions as well. An introductory article can be found at

`https://www.jstatsoft.org/article/view/v008i20`

In order to find a high scoring network, one first needs to define an empty starting network and set up (likelihood equivalent) priors

```
require(deal)
g.start <- network(dat1) # empty network
g.prior <- jointprior(g.start,8) #
```

```
## Warning: Your choice of imaginary sample size is very low
## We advice you to set the imaginary sample size to more than 16
## Imaginary sample size: 8
```

Functions `networkfamily` and `nwfsort` score and enumerate all networks. Apply it to both data sets. Does the likelihood agree with yours?

Unfortunately, `deal` is not great for bigger networks. `bnlearn` is another package for the inference of Bayesian networks with a nice tutorial

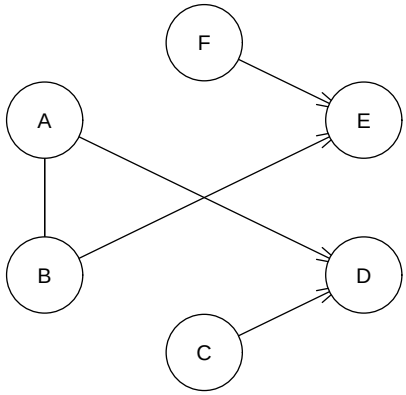
`https://arxiv.org/pdf/0908.3817.pdf`

It is easy to use

```
require(bnlearn)
data(learning.test)
ntw.hc <- hc(learning.test,score="bde")
arcs(ntw.hc)
```

```
##      from to
## [1,] "A"  "B"
## [2,] "A"  "D"
## [3,] "C"  "D"
## [4,] "B"  "E"
## [5,] "F"  "E"
```

```
ntw.pc <- pc.stable(learning.test,alpha=0.05)
plot(ntw.pc)
```



The "bde" score is the discrete Bayesian network score as discussed in Lecture 2. `bnlearn` also implements, as `pc.stable`, the PC algorithm from Lecture 1.

3.2 DREAM5 network challenge

The Dream5 challenge

`http://dreamchallenges.org/project/dream-5-network-inference-challenge/`

provided simulated and real gene expression data and assessed predictions of regulatory pairs returned by participating teams. I used their tool `GeneNetWeaver`

`http://tschaffter.ch/projects/gnw/`

to generate simulated gene expression data for a small subnetwork of *E. coli*. I include the the GeneNetWeaver software in form of a jar file which you can start from a Linux command line as

```
java -jar gnw-3.1.2b.jar
```

to generate your own simulated networks, if you want.

You can load the data I simulated from the network, as well as the gold standard true pairs:

```
load("medium_network.rda")
expr.dat[1:3,1:4]
```

```
##        tyrR      aroF      tyrA      tyrB
## 1 0.8220402 0.1457575 0.0425791 0.0000779
## 2 0.4759674 0.1163276 0.0292339 0.0003840
## 3 0.7029402 0.0946906 0.0372320 0.0039355
```

```
load("medium_network_true.rda")
true.pairs[1:3,]
```

```
##      [,1]   [,2]
## [1,] "tyrR" "aroF"
## [2,] "tyrR" "tyrA"
## [3,] "tyrR" "tyrB"
```

together with the true regulatory pairs in the network as a gold standard. In the following I ask you to apply various algorithms to these data and compare their performance as measured against the gold standard.

3.3 Assessing performance [20 marks]

DREAM5 expected predicted weights for each potential regulatory pair and performed and area-under-the-ROC (AROC) and an area-under-precision-recall-curve (AUPR) analysis based on the weights. We keep it simpler here and want to assess algorithms that return just one network solution without weights. Write an R function that takes a list of genes, a list of predicted gene pairs, and a list of true gene pairs and returns the precision, recall (which is also sensitivity) and specificity, as well as a contingency table of predicted vs true pairs. Also add the F_1 score. You find a nice summary of these performance measures here

`https://en.wikipedia.org/wiki/F1_score`

where the positive condition is the existence of an edge.

```
performance.pairs <- function(genes,pred.pairs,true.pairs) {
  ## compute pr,rc,sp,f1,tab
  ## ...
  c(f1=f1,pr=pr,rc=rc,sp=sp,tab=tab)
}
```

For this exercise *ignore the orientation of pairs*. I found a few R tricks useful. I created a 0 matrix

```
genes <- colnames(expr.dat)
adj.true <- adj.dummy <- matrix(0,length(genes),length(genes))
rownames(adj.true) <- colnames(adj.true) <- genes
```

I could then easily create an adjacency matrix for true pairs

```
adj.true[true.pairs] <- 1
adj.true[1:3,1:10]
```

```
##      tyrR aroF tyrA tyrB aroG cpxR baeR csgD rstA ompR
## tyrR    0    1    1    1    1    0    0    0    0    0
## aroF    0    0    0    0    0    0    0    0    0    0
## tyrA    0    0    0    0    0    0    0    0    0    0
```

A contingency table comparing adjacency matrices

```
adj.dummy[1,4:8] <- 1
table(adj.dummy,adj.true)
```

```
##           adj.true
## adj.dummy    0    1
##         0 1536   59
##         1    3    2
```

In order to extract the performance values from such table, some care needs to be taken to make the adjacency matrix symmetric (eg by `(adj + t(adj)) %% 2`) and to only count each pair once.

Compute the performance values for the edge list from the `bnlearn` Bayesian inference algorithm `hc(...,score=''bde'')` and the PC algorithm `pc.stable(...)`. Don't be disappointed if the PC algorithm predicts only few edges, by increasing `alpha` you can get a few more. But are the few edges at least good predictions?

3.4 More algorithms [10 marks]

One of the best performing algorithms from the DREAM5 competition was GENIE3

`https://bioconductor.org/packages/3.7/bioc/vignettes/GENIE3/inst/doc/GENIE3.html`

a comparatively simple algorithm that regresses target genes on a set of potential regulator genes and picks out the best predictor genes to define regulatory pairs.

```
require(GENIE3)
expr <- t(data.matrix(expr.dat))
## g3.weights <- GENIE3(expr)
```

Use various cutoffs on the edge weights to get a range of lists of regulatory pairs which you can each assess. What cutoff produces the highest F_1 value?

An algorithm that generates Markov networks is `GeneNet`

```
require(GeneNet)
pcor.dyn = ggm.estimate.pcor(expr.dat)
```

```
## Estimating optimal shrinkage intensity lambda (correlation matrix): 0.1327
```

```
gn.edges = network.test.edges(pcor.dyn,direct=TRUE)
```

```
## Estimate (local) false discovery rates (partial correlations):
## Step 1... determine cutoff point
## Step 2... estimate parameters of null distribution and eta0
## Step 3... compute p-values and estimate empirical PDF/CDF
## Step 4... compute q-values and local fdr
## Step 5... prepare for plotting
##
## Estimate (local) false discovery rates (log ratio of spvars):
## Step 1... determine cutoff point
## Step 2... estimate parameters of null distribution and eta0
## Step 3... compute p-values and estimate empirical PDF/CDF
## Step 4... compute q-values and local fdr
## Step 5... prepare for plotting
```

```
## Get the 10 highest scoring edges
gn.net = extract.network(gn.edges, method.ggm="number", cutoff.ggm=10)
```

```
##
## Significant edges:  10
##     Corresponding to  1.28 %  of possible edges
##
## Significant directions:  0
##     Corresponding to  0 %  of possible directions
## Significant directions in the network:  0
##     Corresponding to  0 %  of possible directions in the network
```

```
gn.net[1:3,1:5]
```

```
##        pcor node1 node2         pval         qval
## 1 0.5338394     8    35 2.220446e-16 8.472725e-14
## 2 0.5236355     2     3 2.220446e-16 8.472725e-14
## 3 0.4073949    20    24 1.834088e-12 4.665647e-10
```

```
genes[gn.net[,2]]
```

```
##  [1] "csgD" "aroF" "spy"  "tyrR" "rstA" "tyrR" "ung"  "baeR" "ompR" "baeR"
```

the 2nd column of `gn.net` contains the number of regulators and the 3rd column of the regulated genes. Use them to get the performance values.

3.5 Your algorithm [25 marks]

Finally, I would like you to write your own network inference algorithm based on a regression approach similar to GENIE3. GENIE3 uses random forests to regress genes on potential regulators. You can use a regression algorithm of your choice. The simplest solution would be to use a standard linear regression and use p-values to select predictor genes. However, regression algorithms that are based on regularisation might work better, for example, an elastic net regression as implemented by `glmnet` from the `glmnet` package. An excellent machine learning type algorithm that might perform even better than random forests is `xgboost` from the `xgboost` package.

In summary, your task is to choose a regression algorithm for regressing each gene on all the others using data `expr.dat`. Add all significant predictors together with the target gene to the list of regulatory pairs. Assess your algorithm using your function `performance.pairs`.