Bayesian Network Analysis with bnlearn

Samantha Gonzales

2023-09-28

Data Acquisition

The data set for this analysis comes from GSE14193, where we will use the 6-weeks wild type and PiZ mice data to perform analysis on a predetermined set of genes.

1. Read the GEO page on the data set to determine what data they provide and in what formats. For this data set, the raw count data from an RNA-seq experiment is available
2. Download the file `GSE141593_RAW.tar` and unzip it. There will be one `.txt.gz` file per sample - since we are only interested in the first 10 samples, we can delete or set aside the others. Unzip the individual text files and take a look at their format so that they can be properly read into R later.

R Packages

Below is a list of R packages needed and how to install them. I highly recommend looking at the websites for each to get an idea of how they work. Additionally, more information on any functions used can be found by typing `?function.name()` in the console.

1. BiocManager – used to install Bioconductor packages;

   `install.packages("BiocManager")`

2. DESeq2 – tool for differential expression analysis; used here to normalize expression data

   `BiocManager::install("DESeq2")`

3. openxlsx - write dataframes to excel files

   ```
   install.packages("openxlsx")
   ```

4. bnlearn - for bayesian network analysis

   ```
   install.packages("bnlearn")
   ```

5. Rgraphviz - for visualizing graphs

   ```
   BiocManager::install("Rgraphviz")
   ```

Data Pre-Processing

Some Bayesian network analysis tools require a specific format for the input, so the count data will need to be processed into the correct format. The end goal of this step is a matrix of discrete expression values for each gene per sample, in which the rows represent the samples, and the columns are the genes plus a column for the experimental condition (wild type or PiZ mutant).

1. merge samples into a single matrix

Since the counts are in separate files, we will need to read in each file in R and combine them into a single matrix based on the transcript IDs. Below is a summary of the steps taken to merge multiple files into one data frame.

- use the `list.files()` function to create a variable containing all the file names for each sample
- use the `list()` function create an empty list variable that will contain each sample's file contents
- use a `for` loop to (1) read in each file, (2) assign names to the columns in the data, and (3) save it in the list. The first column, which contains the gene IDs, should all have the same name. The second column, which contains the counts for that particular sample, should be given a unique name that helps identify which sample it is once they are merged.
- use `Reduce(function(x,y) merge(x,y, all = TRUE, by = "transcript.id"), individual.files)` to merge all of the data frames contained in the list `individual.files` by the transcript.id column.

```
gse <- "GSE141593"

count.files <- list.files("./data/", pattern = "counts.txt")
gsm <- gsub(pattern = "_.*", replacement = "", count.files)

# read sample files into a list
individual.files <- list()
for (f in 1:length(count.files)){
  filepath <- paste0("./data/", count.files[f])
  tmp <- read.table(filepath)

  # give the count column a unique name for merging later
  colnames(tmp) <- c("transcript.id", gsm[f])
  individual.files[[f]] <- tmp

}

# merge the datasets into one matrix by the transcript id column
merged <- Reduce(function(x,y) merge(x,y, all = TRUE, by = "transcript.id"), individual.files)

# remove first 5 rows since we don't need them
merged <- merged[-c(1:5),]

# move transcript.id column to rownames and remove
rownames(merged) <- merged$transcript.id
merged <- merged[,-1]
head(merged)
```

2. normalize and scale counts using DESEq2's estimateSizeFactors()

Normalize the count matrix using DESeq2's median of ratio's method then convert to Z-score.

- create a DESeq2 object from the newly merged count matrix
- filter low-expression genes
- use `estimateSizeFactors()` to estimate size factors for each sample
- return normalized counts using DESeq2's `counts()` function with normalized set to TRUE
- standardize using the `scale()` function on the *transposed* count matrix

```
library(DESeq2)

# make a quick metadata table for DESeq2
metadata <- data.frame(sampleID = gsm,
                       treatment = c(rep("wild type", 5), rep("PiZ mutant", 5)))
```

```
metadata$treatment <- factor(metadata$treatment, levels = c("wild type", "PiZ mutant"), labels = c("wt"

# create DESeq object from count matrix, filter low expression genes and estimate size factors
diffxp <- DESeqDataSetFromMatrix(countData = merged,
                                 colData = metadata,
                                 design = ~ treatment)

keep <- rowSums(counts(diffxp) > 10) >= 3
diffxp <- diffxp[keep,]
diffxp <- estimateSizeFactors(diffxp)

#normalize and standardize
norm.counts <- counts(diffxp, normalized = TRUE)

standardized.counts <- scale(t(norm.counts))

# make sure its in the expected orientation
standardized.counts[1:5,1:5]
```

3. discretize gene expression based on Z-score

Label the gene expression values as

- "No Change" if the Z score is between -1 and 1,
- "Low" if the Z score is less than or equal to -1
- "High" if the Z-score is greater than or equal to 1

```
discretize_gene_vector <- function(gene_vector){

  is_low <- gene_vector < -1
  is_high <- gene_vector > 1
  is_none <- !is_low & !is_high

  gene_vector[is_none] <- "noChange"
  gene_vector[is_low] <- "low"
  gene_vector[is_high] <- "high"

  return(gene_vector)
}

discrete.counts <- apply(standardized.counts, MARGIN = 2, discretize_gene_vector)

# double check
discrete.counts[1:5,1:5]
standardized.counts[1:5,1:5]
```

4. subset genes of interest and save final matrix

- read in file containing the genes of interest and find the index of the matching column names of the new discrete count matrix
- create and save a data frame using the "treatment" column in the metadata and the relevant subset of the count matrix

I saved the processed data set in 2 locations and formats: The first one is an R object in the folder that will contain the bnlearn analysis, which will make it easy to load and use. The second one is optional, but I also kept an Excel copy in case I need to refer to it later or want to use it in a separate analysis.

```
library(openxlsx)
# filter for genes of interest, and add treatment column -------------------
genes <- read.table("./data/zn_transporter_genes.txt", sep = "\n")$V1
zn.goi <- which(colnames(discrete.counts) %in% genes)

pizmice.counts <- data.frame(pizMutation = metadata$treatment,
                             as.data.frame(discrete.counts[,zn.goi]))

save(pizmice.counts, file = paste0("./analysis/bnlearn/", gse, "_zn_transporter_gene_expr.RData"))
write.xlsx(pizmice.counts, file = paste0("./data/", gse, "_zn_transporter_gene_expr_clean.xlsx"))
```

Running bnlearn

1. learn the structure from the data/domain knowledge

Learn a structure using the data generated from the previous step. There are different types of algorithms available in bnlearn and other tools learn structures from data. The score-based Hill Climbing algorithm is used here via the function `hc()` . Note that for bnlearn, the data type must be a factor. Also, if you know from literature certain relationships between your genes of interest, you can set the direction of specific arcs using the `set.arc()` function.

```
library(bnlearn)
library(Rgraphviz)

#keep track of sample ids
sample.id <- rownames(pizmice.counts)
pizmice.counts <- data.frame(lapply(pizmice.counts, FUN = function(x) factor(x)))
str(pizmice.counts)

# re-add sample ids to row names
rownames(pizmice.counts) <- sample.id
rm(sample.id)

# Hill Climbing
pzm.hc <- hc(pizmice.counts, restart = 500)

# quick look via Rgraphviz
graphviz.plot(pzm.hc, layout = "dot")
```

2. Estimate parameters

Once we have a structure, the distribution parameters can be estimated using the structure and the data. For discrete data, we can use MLE or Bayes method's to fit the model.

```
pz.mle.fit <- bn.fit(pzm.hc, data = pizmice.counts, method = "mle")
pz.mle.fit
```