Learning Bayesian Networks from gene expression data

Modestas Filipavičius

February 22, 2018

Here we use R package deal1 to analyse yeast cell-cycle regulator genes and learn optimal Bayesian Network.

Install the packages and load and preprocess the data.

YeastCC data: This data package contains an ExpressionSet instance for the yeast cell cycle microarray experiment. The dataset contains gene expression measures (log-ratios, with Cy3-labeled common reference) for 6,178 yeast genes in 77 conditions.

```
# Install deal and load yeastCC data
# install.packages("deal")
library(deal)
#source("https://bioconductor.org/biocLite.R")
#biocLite("yeastCC")
library(yeastCC)

# Load the Yeast cell cycle regulator dataset
yeastCC = yeastCC
# First 3 genes are: [1] "YAL001C"   "YAL002W"   "YAL003W"
# Last genes are:    [1] "YPR202W" "YPR203W" "YPR204W"

# 77 conditions for each gene are measured in log-ratios:
#          cln3.1 cln3.2 clb2.2 clb2.1 alpha0 alpha7 alpha14 alpha21 alpha28 alpha35
# YAL001C   0.15     NA  -0.22   0.07  -0.15  -0.15   -0.21    0.17   -0.42   -0.44
# length(exprs(yeastCC[1]))
```

Turn data object yeastCC into an expression matrix (function exprs) with genes as columns and experiments as rows.

```
expr_df = as.data.frame(exprs(yeastCC))
expr_mt = t(as.matrix(expr_df))
expr_mt[1:5, 1:5]
```

```
##        YAL001C YAL002W YAL003W YAL004W YAL005C
## cln3.1    0.15   -0.07   -1.22   -0.09   -0.60
## cln3.2      NA   -0.76   -0.27    1.20    1.01
## clb2.2   -0.22   -0.12   -0.10    0.16    0.24
## clb2.1    0.07   -0.25    0.23   -0.14    0.65
## alpha0   -0.15   -0.11   -0.14   -0.02   -0.05
```

Restrict the genes to cell cycle regulators.

The ORF names for the 800 cell cycle regulated genes are stored in orf800.

```r
# Create a mask with yeast gene indeces that correspond to 800 cell regulator genes
mask = which(colnames(expr_mt) %in% orf800)

# Subset cell regulators
expr_cc_mt = expr_mt[ , mask]
expr_cc_mt[1:5, 1:5]
```

```
##        YAL022C YAL040C YAL053W YAL067C YAR003W
## cln3.1   -0.22    3.43    0.34   -0.15    0.41
## cln3.2   -0.86    2.75    0.90   -1.25    0.25
## clb2.2   -0.25    0.36   -0.58      NA   -0.62
## clb2.1    0.89    0.72   -0.92    0.07   -0.29
## alpha0   -0.36    1.04    0.21    0.01   -0.30
```

Replace missing data (=NA) with each gene's median.

```r
replace_na_with_col_medians = function(x) {
    x = replace(x, is.na(x), median(x, na.rm = T))
    return(x)
}

cat("Number of NA's before cleanup is:", length(which(is.na(expr_cc_mt) == TRUE)))
```

```
## Number of NA's before cleanup is: 2643
```

```r
expr_cc_mt = apply(expr_cc_mt, 2, replace_na_with_col_medians)
cat("Number of NA's after cleanup is:", length(which(is.na(expr_cc_mt) == TRUE)))
```

```
## Number of NA's after cleanup is: 0
```

```r
# # Check if correct w/ 1st and 2nd genes
# which(is.na(expr_cc_mt[, 2]))          # cdc28.140 idx61; cdc28.110 idx58
# median(expr_cc_mt[, 2], na.rm = T)     # -0.5  -0.045
```

Calculate the range between the 25th and 75th quantile for each gene and keep genes with a value > 1.6.

The interquartile range of an observation variable is the difference of its upper and lower quartiles. It is a measure of how far apart the middle portion of data spreads in value.

```r
# Calculate IQRs for every gene/column, select IQRs with >1.6 and subset the data with them
iqrs = apply(expr_cc_mt, 2, IQR)
mask = which(iqrs > 1.6)
expr16 = as.data.frame(expr_cc_mt[, mask])
expr16[1:5,1:5]
```

```
##        YBR054W YBR088C YER124C YGL028C YGR108W
## cln3.1   -2.56    2.05    0.07   -0.31    0.21
## cln3.2   -1.74    1.67   -1.29    0.23    0.87
## clb2.2    2.76   -2.06   -0.32   -0.49    2.28
## clb2.1    3.38   -1.43   -0.10    0.16    1.66
## alpha0   -1.35   -1.15    0.92    2.45   -1.88
```

Add a dummy gene with expression 1 over all experiments (deal needs at least one discrete variable).

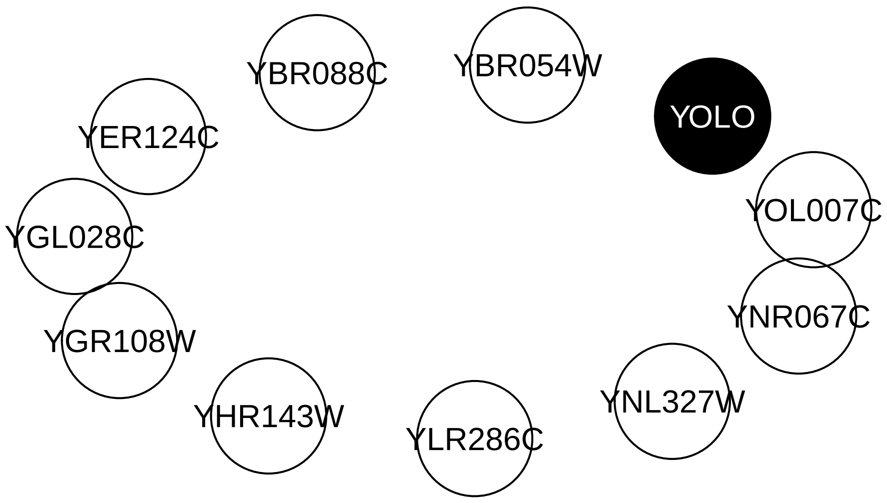
```
YOLO = as.factor(rep(1, length(expr16[, 1]))) # must be a factor because discrete variable expected in
expr16 = cbind(expr16, YOLO)
cat("Class of dummy discrete variable should be a factor:", class(expr16[1, 11]))
```

```
## Class of dummy discrete variable should be a factor: factor
```

Learn the optimal network

Create an empty network prior.

```
bn_expr16 = network(expr16) # to visually inspect local probs: inspectprob=TRUE
# nodes(bn_expr16) # YOLO node should be discrete!
plot(bn_expr16)
```



What are the local probability distributions of the genes?

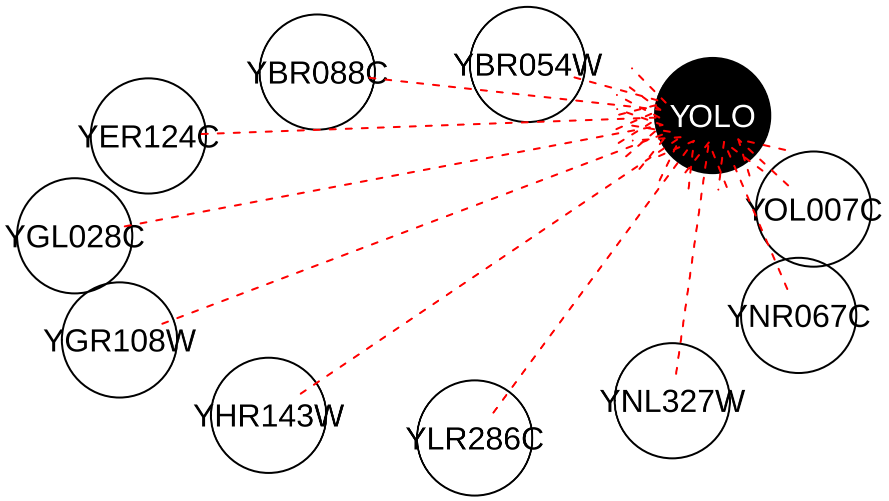
```
localprob(bn_expr16)
```

```
## $YBR054W
##                s2 Intercept:YBR054W
##        1.42234485        0.02649351
##
## $YBR088C
##                s2 Intercept:YBR088C
##      0.9768984314     -0.0005194805
##
## $YER124C
##                s2 Intercept:YER124C
##        1.41456404       -0.02246753
##
## $YGL028C
##                s2 Intercept:YGL028C
##       1.455174903      -0.009350649
##
## $YGR108W
##                s2 Intercept:YGR108W
##        0.92446915        0.06779221
##
## $YHR143W
##                s2 Intercept:YHR143W
##        1.04158249       -0.01097403
##
## $YLR286C
##                s2 Intercept:YLR286C
##      1.9945786136     -0.0007792208
##
## $YNL327W
##                s2 Intercept:YNL327W
##        1.41701617       -0.02616883
##
## $YNR067C
##                s2 Intercept:YNR067C
##        1.60312299       -0.02415584
##
## $YOL007C
##                s2 Intercept:YOL007C
##        1.09477190       -0.01090909
##
## $YOLO
## 1
## 1
```

Ban all outgoing edges for the dummy node (use function banlist).

```
# build a table with edges between 11 and other nodes
mybanlist = matrix(c(1,2,3,4,5,6,7,8,9,10,
                     11,11,11,11,11,11,11,11,11,11),
                   ncol=2)

banlist(bn_expr16) =  mybanlist
plot(bn_expr16)
```



Create a joint prior with imaginary sample size 5, N=5

```
bn_expr16.prior = jointprior(bn_expr16, N=5)
```

```
## Imaginary sample size: 5
```

```
head(bn_expr16.prior)
```

```
## $jointalpha
## 1
## 5
##
## $jointnu
## 1
## 5
##
## $jointrho
## 1
## 5
##
## $jointmu
##       YBR054W       YBR088C     YER124C      YGL028C    YGR108W
## :1 0.02649351 -0.0005194805 -0.02246753 -0.009350649 0.06779221
##        YHR143W       YLR286C     YNL327W     YNR067C     YOL007C
## :1 -0.01097403 -0.0007792208 -0.02616883 -0.02415584 -0.01090909
##
## $jointsigma
## $jointsigma$`:1`
##          YBR054W   YBR088C  YER124C  YGL028C   YGR108W  YHR143W  YLR286C
## YBR054W 1.422345 0.0000000 0.000000 0.000000 0.0000000 0.000000 0.000000
## YBR088C 0.000000 0.9768984 0.000000 0.000000 0.0000000 0.000000 0.000000
## YER124C 0.000000 0.0000000 1.414564 0.000000 0.0000000 0.000000 0.000000
## YGL028C 0.000000 0.0000000 0.000000 1.455175 0.0000000 0.000000 0.000000
## YGR108W 0.000000 0.0000000 0.000000 0.000000 0.9244692 0.000000 0.000000
## YHR143W 0.000000 0.0000000 0.000000 0.000000 0.0000000 1.041582 0.000000
## YLR286C 0.000000 0.0000000 0.000000 0.000000 0.0000000 0.000000 1.994579
## YNL327W 0.000000 0.0000000 0.000000 0.000000 0.0000000 0.000000 0.000000
## YNR067C 0.000000 0.0000000 0.000000 0.000000 0.0000000 0.000000 0.000000
## YOL007C 0.000000 0.0000000 0.000000 0.000000 0.0000000 0.000000 0.000000
##          YNL327W  YNR067C  YOL007C
## YBR054W 0.000000 0.000000 0.000000
## YBR088C 0.000000 0.000000 0.000000
## YER124C 0.000000 0.000000 0.000000
## YGL028C 0.000000 0.000000 0.000000
## YGR108W 0.000000 0.000000 0.000000
## YHR143W 0.000000 0.000000 0.000000
## YLR286C 0.000000 0.000000 0.000000
## YNL327W 1.417016 0.000000 0.000000
## YNR067C 0.000000 1.603123 0.000000
## YOL007C 0.000000 0.000000 1.094772
##
##
## $jointphi
## $jointphi$`:1`
##          YBR054W  YBR088C  YER124C YGL028C  YGR108W YHR143W  YLR286C
## YBR054W 5.689379 0.000000 0.000000  0.0000 0.000000 0.00000 0.000000
## YBR088C 0.000000 3.907594 0.000000  0.0000 0.000000 0.00000 0.000000
## YER124C 0.000000 0.000000 5.658256  0.0000 0.000000 0.00000 0.000000
## YGL028C 0.000000 0.000000 0.000000  5.8207 0.000000 0.00000 0.000000
## YGR108W 0.000000 0.000000 0.000000  0.0000 3.697877 0.00000 0.000000
## YHR143W 0.000000 0.000000 0.000000  0.0000 0.000000 4.16633 0.000000
## YLR286C 0.000000 0.000000 0.000000  0.0000 0.000000 0.00000 7.978314
## YNL327W 0.000000 0.000000 0.000000  0.0000 0.000000 0.00000 0.000000
## YNR067C 0.000000 0.000000 0.000000  0.0000 0.000000 0.00000 0.000000
## YOL007C 0.000000 0.000000 0.000000  0.0000 0.000000 0.00000 0.000000
##          YNL327W  YNR067C  YOL007C
## YBR054W 0.000000 0.000000 0.000000
## YBR088C 0.000000 0.000000 0.000000
## YER124C 0.000000 0.000000 0.000000
## YGL028C 0.000000 0.000000 0.000000
## YGR108W 0.000000 0.000000 0.000000
## YHR143W 0.000000 0.000000 0.000000
## YLR286C 0.000000 0.000000 0.000000
## YNL327W 5.668065 0.000000 0.000000
## YNR067C 0.000000 6.412492 0.000000
## YOL007C 0.000000 0.000000 4.379088
```

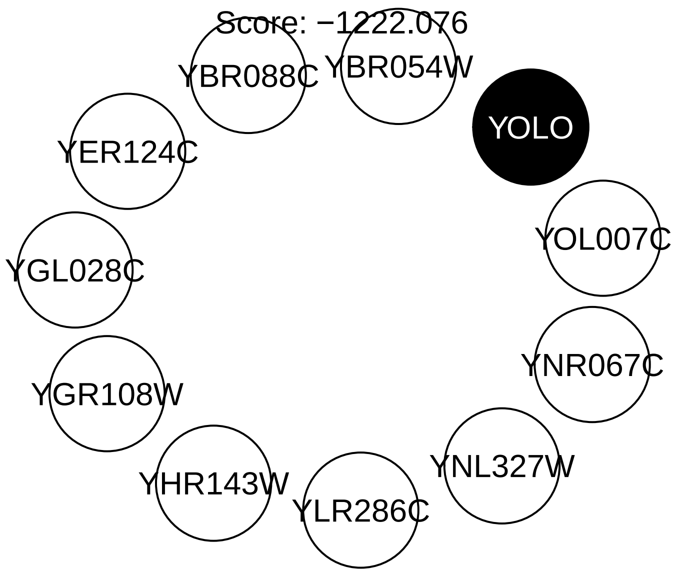
Learn the initial network (use getnetwork to turn the result into an actual network).

```r
# master prior
bn_expr16.m_prior = learn(bn_expr16, expr16, bn_expr16.prior)

# initial network
bn_expr16.initial = getnetwork(bn_expr16.m_prior)
cat("Initial network score is:", bn_expr16.initial$score)
```

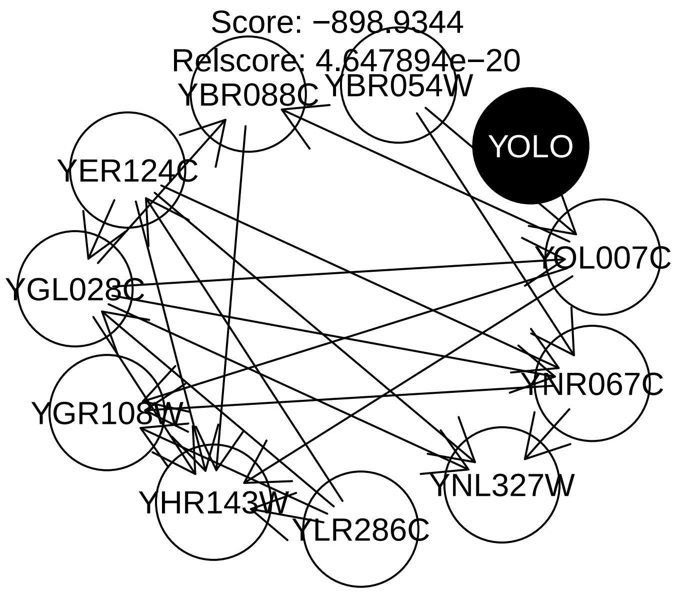
```
## Initial network score is: -1222.076
```

```r
plot(bn_expr16.initial, showban = FALSE)
```



Search for the optimal network and plot it

```r
plot(bn_expr16.optimal, showban = F)
```



Optimal network score is:

```r
bn_expr16.optimal$score
```

```
## [1] -898.9344
```