Bioinformatics Project

Advanced Analytic Techniques, Assignment 2

Anisha Mariam Abraham - 110416080

2024-09-22

Initial Set-up

The required packages and libraries are installed to setup the working environment.

```
install.packages("rmarkdown")
```

```
## The following package(s) will be installed:
## - rmarkdown [2.28]
## These packages will be installed into "~/Library/CloudStorage/OneDrive-UniversityofSouthAustralia/Se
##
## # Installing packages --------------------------------------------------------
## - Installing rmarkdown ...                      OK [linked from cache]
## Successfully installed 1 package in 8.6 milliseconds.
```

```
install.packages("tinytex")
```

```
## The following package(s) will be installed:
## - tinytex [0.53]
## These packages will be installed into "~/Library/CloudStorage/OneDrive-UniversityofSouthAustralia/Se
##
## # Installing packages --------------------------------------------------------
## - Installing tinytex ...                        OK [linked from cache]
## Successfully installed 1 package in 6.9 milliseconds.
```

```
tinytex::install_tinytex(force = TRUE) # install TinyTeX to generate PDF reports
```

```
## tlmgr install tlgpg

## tlmgr update --self

## tlmgr install tlgpg

## tlmgr --repository http://www.preining.info/tlgpg/ install tlgpg

## tlmgr option repository 'https://mirror.aarnet.edu.au/pub/CTAN/systems/texlive/tlnet'

## tlmgr update --list
```

```r
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install(c("graph", "RBGL", "Rgraphviz"))
```

```
## 'getOption("repos")' replaces Bioconductor standard repositories, see
## 'help("repositories", package = "BiocManager")' for details.
## Replacement repositories:
##     CRAN: https://packagemanager.posit.co/cran/latest

## Bioconductor version 3.18 (BiocManager 1.30.25), R 4.3.1 (2023-06-16)

## Warning: package(s) not installed when version(s) same as or greater than current; use
##   `force = TRUE` to re-install: 'graph' 'RBGL' 'Rgraphviz'

## Installation paths not writeable, unable to update packages
##   path: /Users/anisha/Library/Caches/org.R-project.R/R/renv/sandbox/R-4.3/aarch64-apple-darwin20/ac5
##   packages:
##     boot, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nlme,
##     spatial, survival

## Old packages: 'bit', 'jsonlite', 'renv'
```

```r
install.packages("pcalg")
```

```
## The following package(s) will be installed:
## - pcalg [2.7-12]
## These packages will be installed into "~/Library/CloudStorage/OneDrive-UniversityofSouthAustralia/Se
##
## # Installing packages --------------------------------------------------------
## - Installing pcalg ...                          OK [linked from cache]
## Successfully installed 1 package in 6.9 milliseconds.
```

```r
library(pcalg)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(bnlearn)
```

```
##
## Attaching package: 'bnlearn'
```

```
## The following objects are masked from 'package:pcalg':
##
##     dsep, pdag2dag, shd, skeleton
```

```
library(ppcor)
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(gRain)
```

```
## Loading required package: gRbase
```

```
##
## Attaching package: 'gRbase'
```

```
## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, nodes, parents
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(e1071)
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:bnlearn':
##
##     impute
```

Introduction the BRCA-50 Dataset

The BRCA-50 dataset is a comprehensive resource designed to aid in the study of breast cancer. It contains detailed information about the expression levels of 50 critical genes associated with breast cancer. This dataset is instrumental for researchers aiming to understand the role of these genes in the disease and how their expression varies between cancerous and normal tissues.

The dataset includes a total of 1,212 samples. Among these, 112 samples are from normal cases, labeled as 'N', and 1,100 samples are from cancer patients, labeled as 'C'. Each sample provides expression levels for 50 different genes, including notable ones such as FIGF, LYVE1, and CD300LG. These genes have been identified as important in breast cancer research due to their potential roles in the disease.

In terms of data format, each row in the dataset represents a sample, and each column corresponds to the expression level of one of the 50 genes. Additionally, there is a column indicating the class label of each sample—whether it is from a normal case or a cancer patient.

```
# Read the CSV file into a data frame
brcaData <- read.csv("BRCA_RNASeqv2_top50.csv", header = TRUE, sep = ",", stringsAsFactors = FALSE)

# Display few rows of data frame
head(brcaData)
```

```
##        FIGF    LYVE1   CD300LG     SCARA5     PAMR1     SDPR     MYOM1
## 1  544.1270 1360.952  602.8571   882.2222 2508.8889 1425.079  195.2381
## 2  846.2623 3508.792 1736.9064  2060.7428 1572.1674 4267.795  460.7428
## 3  741.3442 3423.116 4074.3381  6321.7923  675.1527 7398.676 2019.8574
## 4  905.1546 9641.237 2786.5979 10177.3196 1591.7526 3809.794  966.4948
## 5 1093.8680 4783.355 2650.7514  3507.0129 1523.2185 5314.676 1707.1561
## 6  913.3363 3324.652 2725.6399  1906.6008 2103.7270 6595.420  724.2928
##       BTNL9    KCNIP2    SLC2A4     PDE2A        LEP    ACVR1C   ABCA10
## 1  717.7778  283.8095   68.8889  658.4127   987.3016   57.7778 164.1270
## 2 2261.9652 2579.7837  400.1881 1729.0080 16210.6253  761.2600 244.0997
## 3 4518.3299 8273.9308 1040.2240 2649.1853 37746.4358 3481.6701 244.3992
## 4 2610.3093 5992.7835  958.7629 2597.9381  7954.1237 2986.0825 321.1340
## 5 2827.3706 5088.7792 1287.5636 2153.5831 28808.9305 2052.1002 271.2715
## 6 2493.0400 6469.2411  792.0970 1814.5487 13086.6637 1602.6044 528.9627
##        AQP7    GPR146    ATP1A2    FXYD1  ARHGAP20      NPR1     ATOH8
## 1  213.9683  262.5397  123.1746 302.5397  110.4762  807.9365  628.2540
## 2 2694.4993  900.7992 1587.2120 782.3225  278.7024 3299.2948 1084.7203
## 3 9354.8880 1536.1507 5110.9980 742.8717  759.1650 5919.5519 1054.4807
## 4 3496.9072 1191.7526 4108.2474 588.1443  816.4948 4698.4536 1208.7629
## 5 4275.9406 1051.9087 1224.1368 667.9328 1170.4680 3651.9187  788.4436
## 6 4052.9861  889.5375  573.8662 724.2928 1705.8824 3189.4926 1246.5200
##       ABCA9   ALDH1L1  ADAMTS5      RDH5      GPAM       CA4   KLHL29   GPIHBP1
## 1  396.1905  175.8730 1400.952  217.2603  1293.333   70.1587 505.3968  383.8095
## 2 1042.5952 1243.0654 1906.159 1804.9760  8317.818  754.1138 509.2619 1598.4955
## 3 2514.7658 1870.6721 2656.823 3389.8371 42443.992 1114.5621 588.0855 1929.2261
## 4 2755.1546 3053.0928 8410.825 2129.8969 22096.392  428.8660 592.7835  775.7732
## 5  853.3341 1800.3447 3519.698 2639.5298 51813.823  656.2232 644.0257 1169.0043
## 6 1022.9008  885.9452 3348.002 1780.8621 23591.828  722.9457 713.9650  918.2757
##   LOC728264    MAMDC2  TMEM132C     ITIH5      HSPB7     HSPB6      DMD
## 1 4158.4127 1114.4444  194.9206  2855.238   288.8889  2822.222 1161.270
## 2 1922.7080  717.1641 1130.2304 13662.811  5728.6319  9107.287 1048.237
## 3  418.5336  938.2943 3162.9328 30184.827 10879.8371 20623.727 3536.660
## 4  455.6701 2045.2062 2252.0619  9205.155  5174.7423 14653.608 2420.619
## 5  529.3696  912.8333 1305.1279 15925.976  5655.2291 14374.947 2742.964
## 6 2588.6843  974.3556 1524.4724 11991.917  3330.0404 10604.400 2626.403
##      SPRY2   IGFBP6    CXCL2      EBF1       KLB   CLEC3B  TMEM220   IBSP
## 1 1142.540 1773.333 617.7778  538.7302   80.6349 2584.127 211.7460 2.2222
## 2 1176.117 6479.737 248.9892 1916.3141 1270.8980 6389.093 352.4213 0.0000
## 3 1567.719 3161.405 386.4562 4590.6314 3139.0020 6565.682 312.6273 0.0000
```

```
## 4 1173.196 5676.289  63.4021 3756.7010 1383.5052 7955.670 202.5773 0.0000
## 5 1301.713 1690.568 342.0167 3006.9172 4086.1482 6804.717 296.6422 0.0000
## 6 2016.614 1788.954 814.0997 2040.8621 2154.0189 3478.222 298.1590 0.0000
##      HIF3A    IGSF10      CIDEC   C2orf40      LEPR   ANGPTL1 class
## 1  70.4762  644.7619   716.1905 1760.3175  770.9333  230.7937     N
## 2 374.2360  435.5430  7336.7710  225.2938 1397.3785  459.2384     N
## 3 653.2587 1254.5825 15562.9633  207.7393 1645.2037 1347.7597     N
## 4 354.6392 1019.5876 12504.7113  224.7423 2484.2216  596.3918     N
## 5 283.9568 1367.5789 13990.4835  157.1033 1857.0726  677.6907     N
## 6 310.7319 1704.9843  7711.2708  414.9079 2087.7863  550.0674     N
```

Question 1

Use a causal structure learning algorithm to find the gene regulatory network, i.e. the network showing the interactions between genes, using the gene expression data. Explain how the algorithm works.

Answer:

A gene regulatory network is a map that shows how different genes interact with each other. For example, it might show which genes influence the expression of other genes. Understanding these interactions helps researchers learn more about how genes control each other and how they contribute to diseases like cancer. The **PC (Peter-Clark) algorithm** is a method used to discover causal relationships between variables, such as genes, based on statistical data.

Working of PC Algorithm can be explained using an example from the dataset. Let's consider three genes - **FIGF**, **MYOM1**, **ARHGAP20** and **EBF1**.

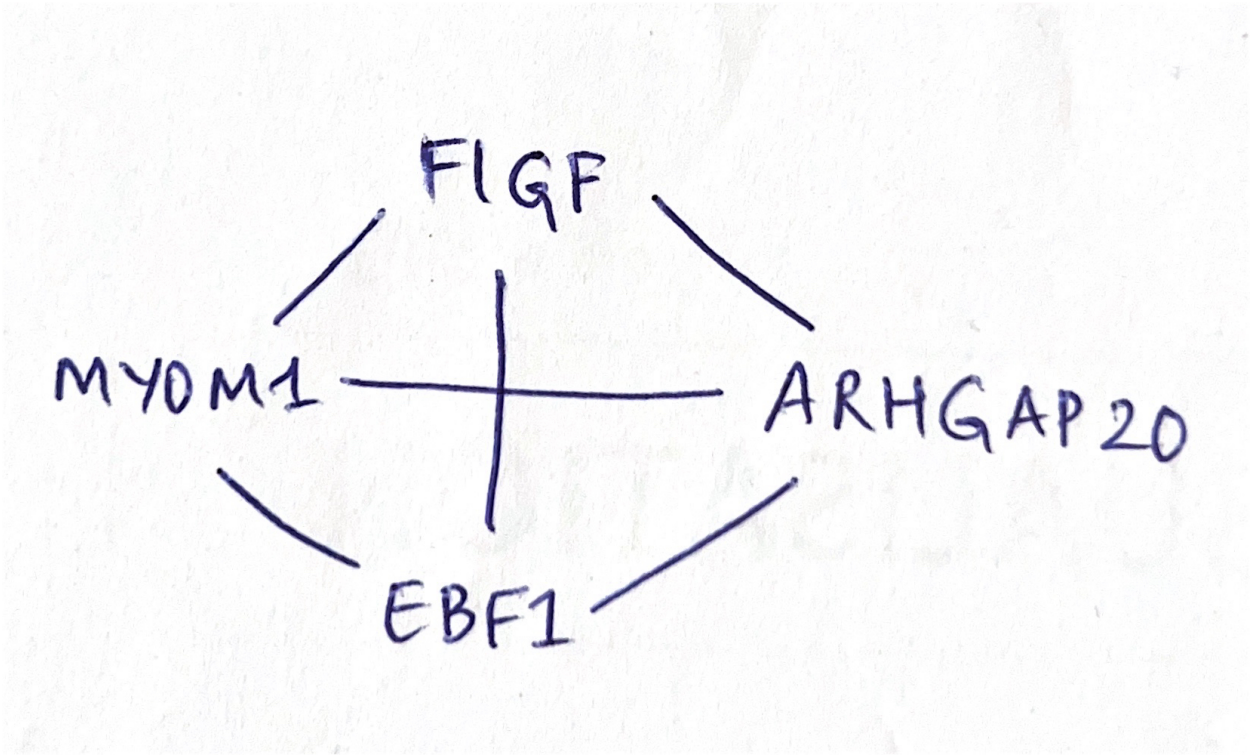
Input: **FIGF**, **MYOM1**, **ARHGAP20** and **EBF1**, connected graph.

Output: Undirected graph with a set of edges.

depth d = 0

Step 1: Create an Initial Network

The PC algorithm starts by assuming that depth **d=0** and all genes are connected to each other. This is called the initial network. At this point, the network is fully connected, meaning that every gene is thought to potentially influence every other gene.



Step 2: Find Conditional Independence

The key idea of the PC algorithm is to test whether the relationship between two genes (**X & Y**) is influenced by other genes. This is called conditional independence. In simple terms, if the expression of gene A does not affect the expression of gene B once you account for gene C (**Z**), then A and B are not directly connected in the final network. **Z** is equal to **d** so in this iteration, $\mathbf{Z = 0}$.

```
cor(brcaData[, c("FIGF", "EBF1", "MYOM1", "ARHGAP20")])
```

```
##                FIGF      EBF1      MYOM1   ARHGAP20
## FIGF     1.00000000 0.7437452 0.09350064 0.76007322
## EBF1     0.74374521 1.0000000 0.11449829 0.84423111
## MYOM1    0.09350064 0.1144983 1.00000000 0.09713995
## ARHGAP20 0.76007322 0.8442311 0.09713995 1.00000000
```

FIGF & *EBF1* : Strong relation

FIGF & *ARHGAP20*: Strong relation

FIGF & *MYOM1*: Weak relation

EBF1 & *MYOM1*: Weak relation

EBF1 & *ARHGAP20*: Strong relation

ARHGAP20 & *MYOM1*: Weak relation

Step 3: Remove Unnecessary Connections

Based on the tests of conditional independence, the algorithm removes edges (connections) between genes (**X & Y**) that are not directly influencing each other. Here, gene FIGF does not affect gene MYOM1, the direct link between FIGF and MYOM1 is removed. Gene EBF1 does not affect gene MYOM1, the direct link between EBF1 and MYOM1 is removed. Gene ARHGAP20 does not affect gene MYOM1, the direct

link between ARHGAP20 and MYOM1 is removed. Rest of the genes have an influence so the direct links are maintained.

I(FIGF, EBF1) ? No

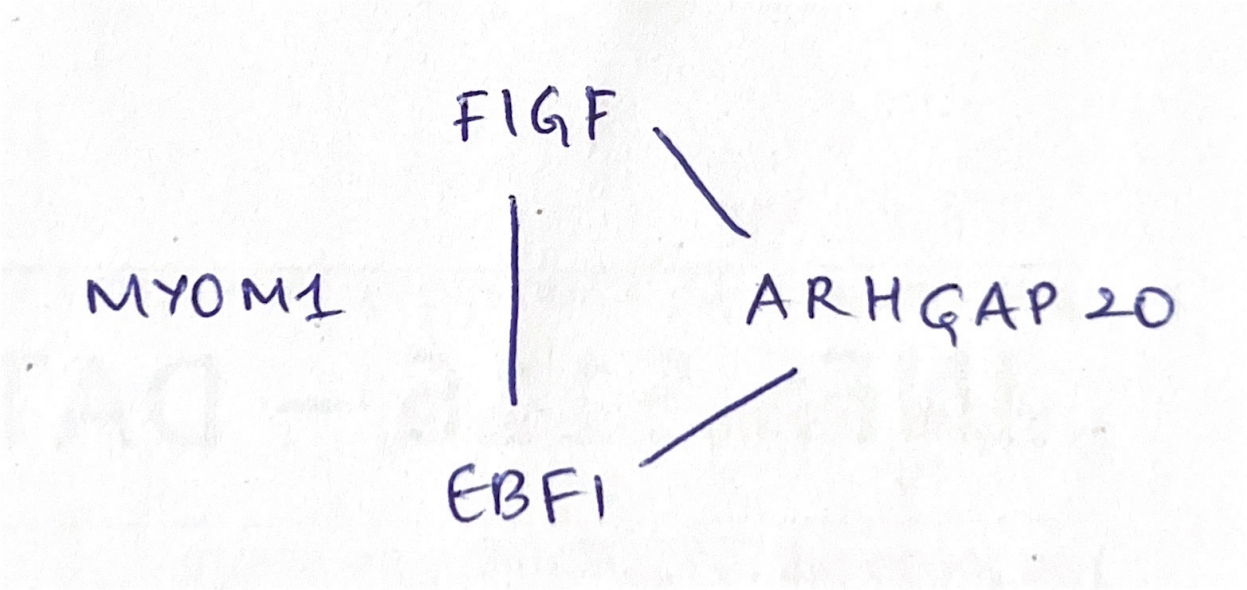
I(FIGF, ARHGAP20) ? No

I(FIGF, MYOM1) ? Yes

I(EBF1, MYOM1) ? Yes

I(EBF1, ARHGAP20) ? No

I(ARHGAP20, MYOM1) ? Yes



We complete a single depth level until every pair of adjacent vertices X and Y in G have been considered.

depth d = 1

Z = d = 1 so repeat the conditional independence test, this time for two genes, controlling the third gene.

```
# Calculate partial correlation
# I(FIGF, EBF1|ARHGAP20) - low
pcor.test(brcaData$FIGF, brcaData$EBF1, brcaData$ARHGAP20)
```

```
##   estimate      p.value statistic    n gp  Method
## 1 0.293046 2.077626e-25  10.65728 1212  1 pearson
```

The estimate shows a weak relation so independence **I(FIGF, EBF1|ARHGAP20) = Yes** so remove the link.

```
# I(FIGF, ARHGAP20|EBF1) - low
pcor.test(brcaData$FIGF, brcaData$ARHGAP20, brcaData$EBF1)
```

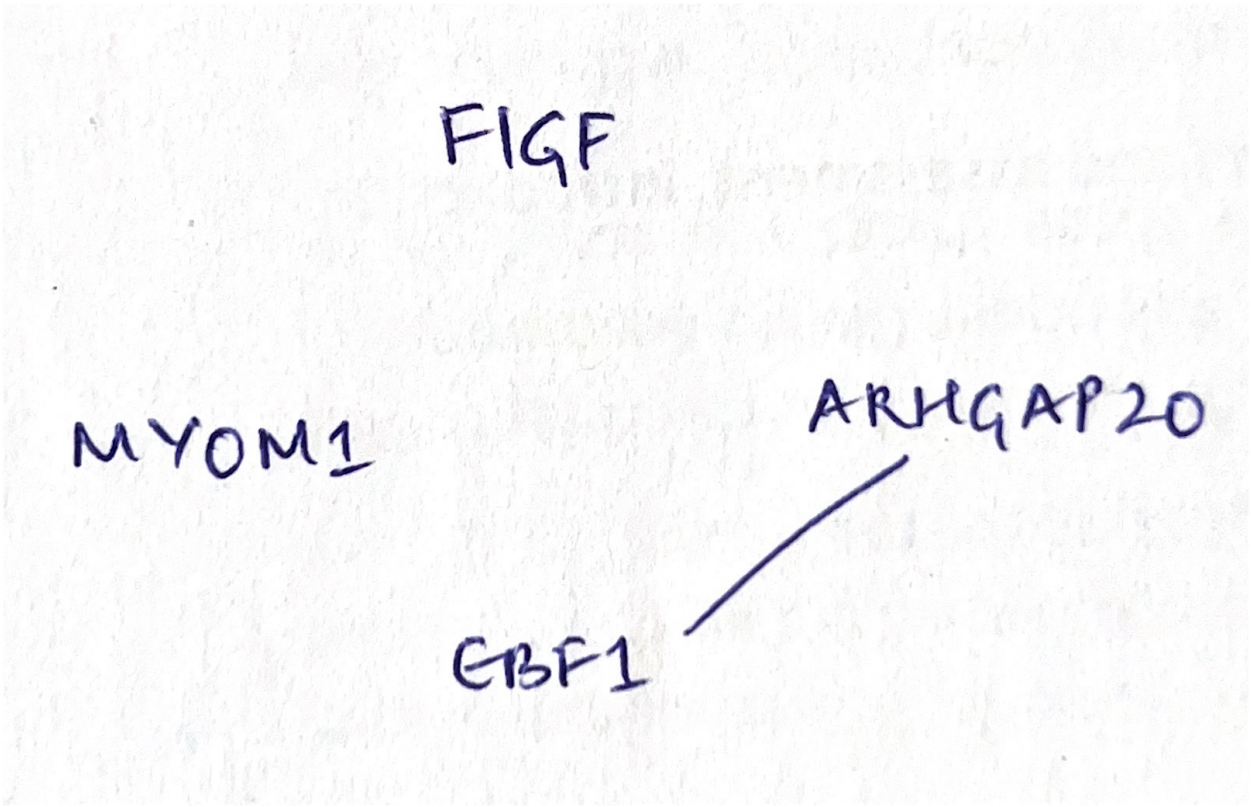
```
##    estimate      p.value statistic    n gp  Method
## 1 0.3689278 2.405816e-40  13.80145 1212  1 pearson
```

The estimate shows a weak relation so independence **I(FIGF, ARHGAP20|EBF1) = Yes** so remove the link.

```
# I(EBF1, ARHGAP20|FIGF) - high
pcor.test(brcaData$EBF1, brcaData$ARHGAP20, brcaData$FIGF)
```

```
##    estimate       p.value statistic    n gp  Method
## 1 0.6421159 1.003186e-141  29.12417 1212  1 pearson
```

The estimate shows a strong relation so independence **I(EBF1, ARHGAP20|FIGF) = No** so keep the link.

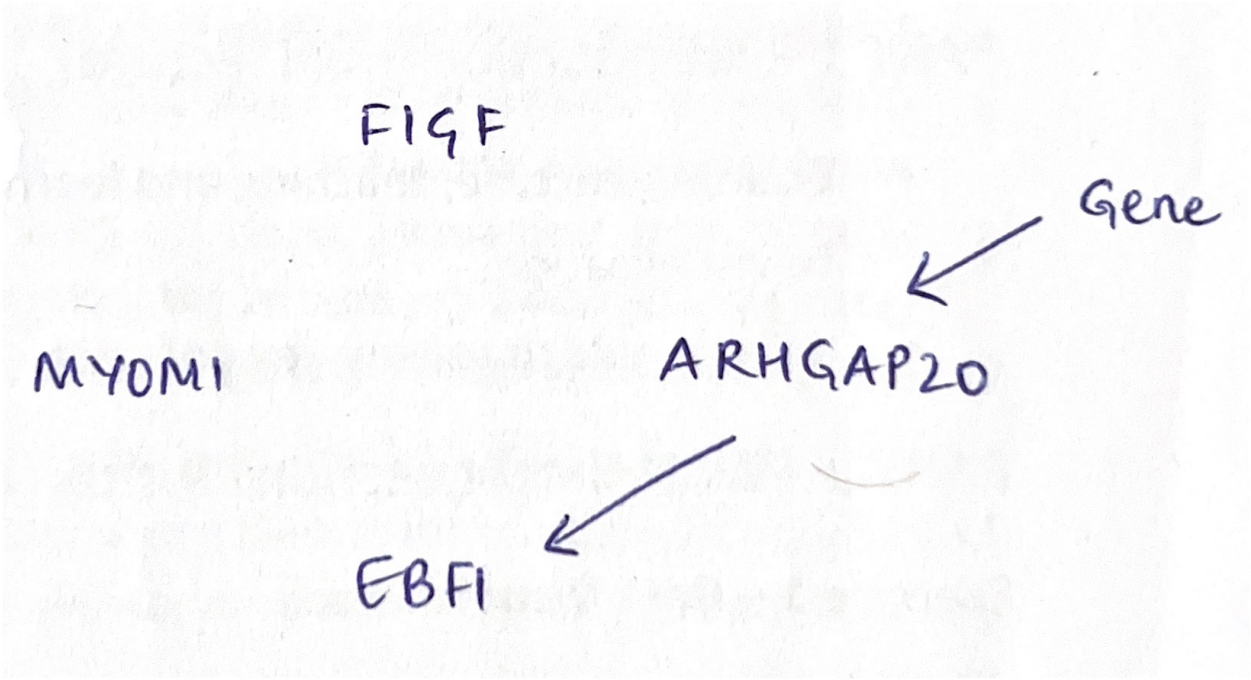


depth d = 2

Z = d = 2 so repeat the conditional independence test, this time for three genes, controlling the fourth gene. But we only have existing direct link between *two* nodes. There is no subset with 2 nodes so we have to stop the iterations now.

After all iterations, we end up with a simplified network showing the direct interactions between genes. This network represents how genes regulate each other based on the data.

To orientate the edges, assume there is a gene with an edge directed to **ARHGAP20**. If there is a chain X_i -> X_j and X_i and X_k are non-adjacent, we can orient X_j - X_k to X_j -> X_k. Likewise, we can orientate the edge as below.



Implementation:

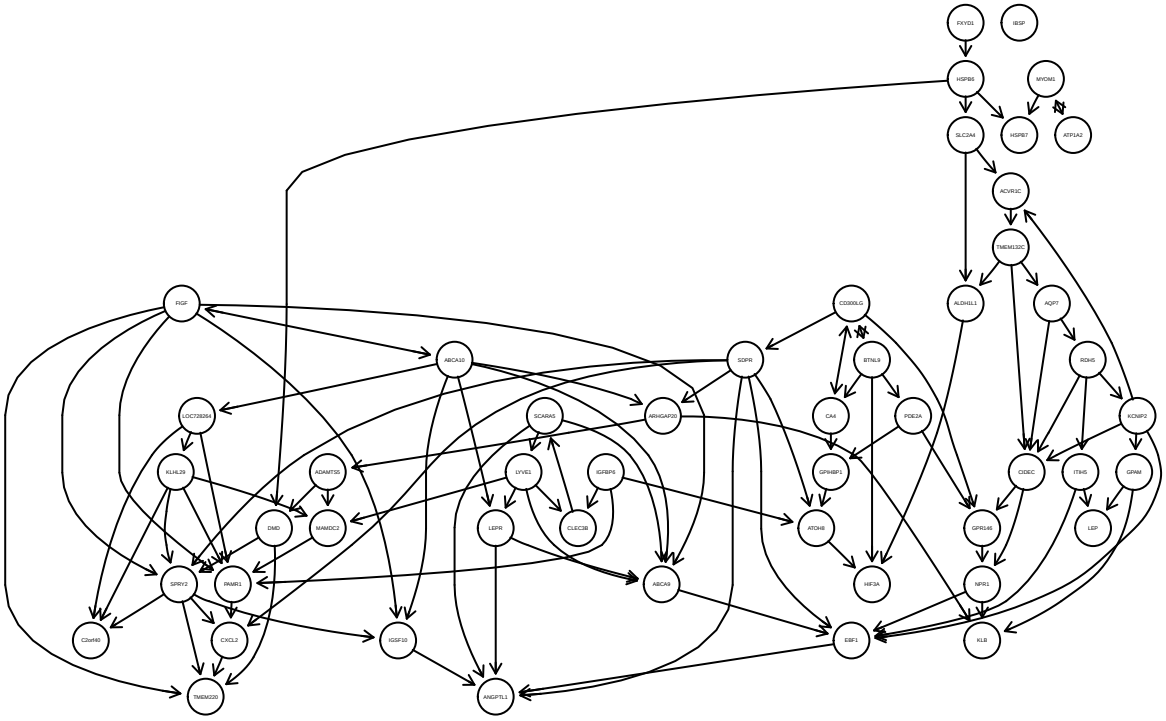
```
# Remove the column named "class"
brcaDataQ1 <- brcaData[, !names(brcaData) %in% "class"]

# Get number of rows and list of column names
n <- nrow(brcaDataQ1)
V <- colnames(brcaDataQ1)
```

`pc.fit` variable will store the result of running the PC algorithm, which identifies the structure of a causal graph, where nodes (representing variables or genes) are connected by edges if there's a direct relationship between them. `C` is the correlation matrix of the gene expression data (brcaDataQ1). It shows how strongly the expression of each gene is related to the others. `gaussCItest` is used to determine if two genes are conditionally independent given a set of other genes.

```
# Estimate CPDAG
pc.fit <- pc(suffStat = list(C = cor(brcaDataQ1), n = n),
indepTest = gaussCItest, alpha=0.01, labels = V)

if (require(Rgraphviz)) {
  # Plot the estimated CPDAG with customized node attributes
  plot(pc.fit, main = "Estimated CPDAG - Gene Regulatory Network")
}
```

```
## Loading required package: Rgraphviz

## Loading required package: graph

## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'

## The following object is masked from 'package:bnlearn':
##
##     score

## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min

##
## Attaching package: 'graph'

## The following objects are masked from 'package:gRbase':
##
##     addEdge, adj, connComp, edges, nodes, removeEdge, subGraph

## The following objects are masked from 'package:bnlearn':
##
##     degree, nodes, nodes<-

## Loading required package: grid
```



Question 2

EBF1 is an important gene that is involved in many biological processes leading to cancer. Find the top 10 other genes that have strong causal effects on EBF1 using a causal inference algorithm.

Answer:

idaFast (Intervention Discalculation Algorithm - Fast) is a method used to estimate the causal effects of variables on a target variable based on observational data. The goal of idaFast is to estimate the causal effects of multiple variables (genes, in gene expression data) on a given target variable.

Before idaFast can be applied, the PC algorithm (Peter-Clark algorithm) is often used to generate a graphical model from the data. This graphical model is a Directed Acyclic Graph (DAG) where the nodes represent variables (genes) and the edges represent potential causal relationships between them.

To estimate causal effects, idaFast makes use of the covariance matrix of the data, which represents how the different variables in the dataset vary together. The covariance matrix is essential for calculating causal effects in linear Gaussian models.

In observational data, we cannot directly intervene on the variables. However, idaFast simulates interventions by estimating how a change in one variable would propagate through the causal network (DAG) and affect other variables. This helps estimate the causal effect of each variable on the target. idaFast considers all possible paths and returns the causal effect (or another measure of choice) for each variable on the target EBF1 for this question. We can see the top ten genes that have strong causal effects on EBF1 below.

```r
# Assigning same dataset in Q1 to Q2
brcaDataQ2 <- brcaDataQ1

# Specify the target gene
targetGene <- "EBF1"

# Identify the position of the EBF1 gene
targetGeneIndex <- which(colnames(brcaDataQ2) == "EBF1")

# Exclude the EBF1 column for calculating causal effects
reqIndices <- setdiff(1:ncol(brcaDataQ2), targetGeneIndex)

# Calculate the causal effects of all genes on EBF1
causalEffects <- sapply(reqIndices, function(gene) {
  effects <- idaFast(gene, targetGeneIndex, cov(brcaDataQ2), pc.fit@graph)
  # Take the minimum of the absolute values of the causal effects
  min(abs(effects))
})

# Convert to named vector with gene names for clarity
names(causalEffects) <- colnames(brcaDataQ2)[reqIndices]

# Rank genes by the absolute value of the causal effects and select the top 10
topTenGenes <- sort(causalEffects, decreasing = TRUE)[1:10]
topTenGenes
```

```
##     FXYD1    ABCA10   TMEM220  ARHGAP20      FIGF    KLHL29   GPIHBP1  TMEM132C
## 1.9760641 1.8314912 1.8291653 1.7366893 1.3455557 1.3398860 1.2125681 1.0699450
##      RDH5     ABCA9
## 0.9486133 0.7651394
```

Question 3

Use a local causal structure learning algorithm to find genes in the Markov blanket of ABCA9 from data. Explain how the algorithm works.

Answer:

HITON-PC is a local causal structure learning algorithm designed to efficiently discover the Markov blanket (MB) of a target variable. The Markov blanket of a target variable T is the smallest set of variables that makes T conditionally independent of all other variables in the dataset. The MB typically consists of: **Parents:** Variables that directly influence the target. **Children:** Variables that are directly influenced by the target. **Spouses:** Parents of the target's children (other variables that indirectly influence the target via a child).

The goal is to identify the MB of a target variable T efficiently. Finding the MB is important because it is the minimal set of variables required to predict T without needing the rest of the variables. HITON-PC builds on the PC algorithm, which is a constraint-based causal discovery method that finds conditional independencies between variables using conditional independence tests (e.g., using partial correlations or mutual information).

Input: X = **{Gene A, Gene B, Gene C, Gene D}**, Z = **Gene Z**

Output: **PC**, the subset of X that comprises the parents and children of Z

We have a dataset of genes, and we want to find out which genes (let's call them X genes) are direct causes or effects (parents and children) of a particular gene of interest, say Gene Z.

Step 1: Initialization:

PC = : Initially, the set of parents and children (PC) of Gene Z is empty. **OPEN:** All the genes in the dataset except Gene Z are placed in an **OPEN** list, which contains potential candidate genes for the PC set. These genes are sorted in descending order based on how strongly they are associated with Gene Z (e.g., through correlation or mutual information). Suppose the genes are initially ordered in the OPEN list based on their correlation with Gene Z: {GeneA,GeneB,GeneC,GeneD}.

Step 2: Main Loop (Finding Candidate Parents/Children):

While OPEN : As long as the OPEN list is not empty, we continue the process.

Remove Gene A: It is the most strongly associated with Gene Z, so we remove it from OPEN and add it to PC. We test if **Gene A** is conditionally independent of Gene Z given subsets of the PC set (which is empty at this point). If it is not independent, we keep it in PC.

Remove Gene B: Next, we remove **Gene B** from OPEN and test if it is conditionally independent of **Gene Z** given any genes in the PC set (which currently has **Gene A**). If **Gene B** is not independent, we add it to PC.

Step 3: Second Loop (Refining the PC Set):

After we go through the OPEN list and identify some potential parents and children, we perform a second loop over the PC set to refine it. **For each gene in PC**, we again test whether it is conditionally independent of Gene Z given subsets of the other genes in PC that were added before it. This ensures that we remove any genes that may have been incorrectly added to PC in the first pass.

Step 4: Output:

After performing the tests, the remaining genes in PC are the direct parents or children of Gene Z. In this case, suppose the final **PC set** is {GeneA,GeneC}, which means that Gene A and Gene C are either direct causes or direct effects of Gene Z.

Implementation:

learn.nbr function learns the neighborhood of the gene ABCA9. The neighborhood refers to the set of variables (or genes) that are directly connected to the target, either as causes or effects. An *alpha of 0.01* means that a relationship between genes will only be considered significant if the probability of it occurring by random chance is less than 1%. The below genes are the parents and children of ABCA9.

```
# Assigning same dataset in Q1 to Q3
brcaDataQ3 <- brcaDataQ1

#  Learn the parents and children set of ABCA9
HITON.PC.ABCA9 <- learn.nbr(brcaDataQ3, "ABCA9", method = "si.hiton.pc", alpha = 0.01)
HITON.PC.ABCA9
```

```
## [1] "SCARA5" "LYVE1"  "FIGF"   "ABCA10" "LEPR"   "ACVR1C"
```

learn.mb function is used to learn the Markov Blanket of ABCA9. The Markov Blanket consists of the most important genes that directly influence or are influenced by the target gene. *IAMB* is an algorithm that incrementally adds variables (genes) to the Markov Blanket of the target gene based on their statistical association with the target. Below includes the parents and children from the step but adds the spouses of ABCA9 as well.

```r
# Learn the markov blanket
MB.ABCA9=learn.mb(brcaDataQ3, "ABCA9", method="iamb", alpha=0.01)
MB.ABCA9
```

```
##  [1] "EBF1"      "ABCA10"    "SCARA5"    "ACVR1C"    "CD300LG"   "LYVE1"
##  [7] "GPAM"      "FIGF"      "LEPR"      "LOC728264" "TMEM132C"  "HIF3A"
## [13] "LEP"       "ANGPTL1"   "PAMR1"     "CLEC3B"    "GPIHBP1"   "KLB"
## [19] "ATOH8"     "RDH5"      "NPR1"      "CIDEC"
```

Discretise the dataset to binary using the average expression of ALL genes as the threshold.

```r
# Finding average expression of ALL genes
averageGeneExpression <- mean(as.matrix(brcaDataQ1))

# Discretize the data
brcaDataBinary <- as.data.frame(ifelse(as.matrix(brcaDataQ1) >= averageGeneExpression, 1, 0))

head(brcaDataBinary)
```

```
##   FIGF LYVE1 CD300LG SCARA5 PAMR1 SDPR MYOM1 BTNL9 KCNIP2 SLC2A4 PDE2A LEP
## 1    1     1       1      1     1    1     0     1      0      0     1   1
## 2    1     1       1      1     1    1     1     1      1      1     1   1
## 3    1     1       1      1     1    1     1     1      1      1     1   1
## 4    1     1       1      1     1    1     1     1      1      1     1   1
## 5    1     1       1      1     1    1     1     1      1      1     1   1
## 6    1     1       1      1     1    1     1     1      1      1     1   1
##   ACVR1C ABCA10 AQP7 GPR146 ATP1A2 FXYD1 ARHGAP20 NPR1 ATOH8 ABCA9 ALDH1L1
## 1      0      0    0      0      0     0        0    1     1     1       0
## 2      1      0    1      1      1     1        0    1     1     1       1
## 3      1      0    1      1      1     1        1    1     1     1       1
## 4      1      1    1      1      1     1        1    1     1     1       1
## 5      1      0    1      1      1     1        1    1     1     1       1
## 6      1      1    1      1      1     1        1    1     1     1       1
##   ADAMTS5 RDH5 GPAM CA4 KLHL29 GPIHBP1 LOC728264 MAMDC2 TMEM132C ITIH5 HSPB7
## 1       1    0    1   0      1       1         1      1        0     1     0
## 2       1    1    1   1      1       1         1      1        1     1     1
## 3       1    1    1   1      1       1         1      1        1     1     1
## 4       1    1    1   1      1       1         1      1        1     1     1
## 5       1    1    1   1      1       1         1      1        1     1     1
## 6       1    1    1   1      1       1         1      1        1     1     1
##   HSPB6 DMD SPRY2 IGFBP6 CXCL2 EBF1 KLB CLEC3B TMEM220 IBSP HIF3A IGSF10 CIDEC
## 1     1   1     1      1     1    1   0      1       0    0     0      1     1
## 2     1   1     1      1     0    1   1      1       1    0     1      1     1
## 3     1   1     1      1     1    1   1      1       1    0     1      1     1
## 4     1   1     1      1     0    1   1      1       0    0     1      1     1
## 5     1   1     1      1     1    1   1      1       0    0     0      1     1
## 6     1   1     1      1     1    1   1      1       0    0     1      1     1
##   C2orf40 LEPR ANGPTL1
## 1       1    1       0
## 2       0    1       1
## 3       0    1       1
## 4       0    1       1
## 5       0    1       1
## 6       1    1       1
```

Question 4

Use PC-simple algorithm (pcSelect) to find the parent and children set of the class variable. Explain how PC-simple works.

Evaluate the accuracy of the Naïve Bayes classification on the dataset in the following cases: a) Use all features (genes) in the dataset b) Use only the features (genes) in the parent and children set of the class variable

Compare the accuracy of the models in the two cases using 10-fold cross validation.

Answer:

The **PC-simple algorithm** is used to identify the set of parents and children (PC) of a target variable Z, given a dataset with a set of predictor variables $\{X_1, X_2, ..., X_m\}$.

Input: **Predictor variables (`X~1~, X~2~, ..., X~m~`)** and **Gene_Z**

Output: **PC**, the subset with parents and children of Z

Step 1: Initialization

Start with k = 0, representing the current level of conditional independence testing. Set $PC^k = \{ X_1, X_2, ..., X_m \}$, initially including all genes except the target Gene_Z.

Step 2: Iterative Pruning of Independent Genes

The algorithm proceeds by testing conditional independence between each gene X in $\{PC\}^{\{k-1\}}$ and the target gene `Z` given smaller subsets of the other genes in $\{PC\}^{\{k-1\}}$. For each gene X and each subset S of size |S| = k - 1 from the remaining genes, the algorithm performs a conditional independence test.

Step 3: Testing Conditional Independence

If gene X and target gene Z are found to be conditionally **independent** given a subset S, it implies that X is **not** directly associated with `Z` (i.e., it is neither a parent nor a child of `Z`). In this case, X is removed from the current PC set. After the first iteration, Gene_3 may be removed because it is found to be independent of Gene_1 given Gene_2. In the next iteration, Gene_4 may be removed because it is independent of Gene_1 given a subset of the remaining genes.

Step 4: Incremental Updates

Increase k by 1 and repeat the process of testing conditional independence for higher-order subsets. At each step, the size of the conditioning subsets increases, allowing the algorithm to capture more complex dependencies between the genes.

Step 5: Termination

The algorithm stops when no more variables can be pruned, meaning that for all genes in the current PC set, none can be conditionally independent of the target gene `Z` given any subset of the remaining genes. The output is the final PC set $\{PC\}^k$, which consists of genes that are either direct **parents** or **children** of the target gene `Z`.

Implementation:

The pcSelect function is used to find which genes are most important for predicting the class variable (`brcaDataBinary$Class`). `brcaDataBinary$Class` specifies the target variable (in this case, the cancer classification). `brcaDataQ4` is the gene expression data (the predictor matrix), where each column is a gene, and each row is a sample. Below are the genes that are either directly influencing the class or are influenced by the class.

```r
# Ensure the class variable is converted to a numeric binary variable (1 for "C", 0 for "N")
brcaDataBinary$Class <- ifelse(brcaData$class == "C", 1, 0)

# Exclude the class column for the predictor data matrix
brcaDataQ4  <- brcaDataBinary[, -which(names(brcaDataBinary) == "Class")]

# Set up the sufficient statistics using the correlation matrix of the numeric data
suffStat <- list(C = cor(brcaDataQ4), n = nrow(brcaDataQ4))

# Use the class as the target variable and the gene expression data as the predictor matrix
pcSelResults <- pcSelect(y = brcaDataBinary$Class, dm = brcaDataQ4, alpha = 0.01)

# Display the results
pcSelResults
```

```
## $G
##      FIGF     LYVE1   CD300LG    SCARA5     PAMR1      SDPR     MYOM1     BTNL9
##      TRUE     FALSE      TRUE      TRUE     FALSE     FALSE     FALSE     FALSE
##    KCNIP2    SLC2A4     PDE2A       LEP    ACVR1C    ABCA10      AQP7    GPR146
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
##    ATP1A2     FXYD1  ARHGAP20      NPR1     ATOH8     ABCA9   ALDH1L1   ADAMTS5
##      TRUE     FALSE      TRUE     FALSE      TRUE     FALSE     FALSE     FALSE
##      RDH5      GPAM       CA4    KLHL29   GPIHBP1 LOC728264    MAMDC2  TMEM132C
##     FALSE     FALSE     FALSE      TRUE     FALSE     FALSE      TRUE     FALSE
##     ITIH5     HSPB7     HSPB6       DMD     SPRY2    IGFBP6     CXCL2      EBF1
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE      TRUE     FALSE
##       KLB    CLEC3B   TMEM220      IBSP     HIF3A    IGSF10     CIDEC   C2orf40
##     FALSE     FALSE      TRUE     FALSE     FALSE     FALSE     FALSE     FALSE
##      LEPR   ANGPTL1
##     FALSE     FALSE
##
## $zMin
##  [1] 12.4322070  1.7534317  8.4777819  2.7931230  2.3164672  1.2653223
##  [7]  1.5646010  2.4111944  0.1763132  2.3741410  2.0819730  2.0447520
## [13]  2.2876307  1.1887476  1.2882622  1.1976424  5.0732814  2.3416787
## [19] 10.7229683  2.1068818  2.6613375  1.8493220  1.9299658  1.7541644
## [25]  0.8571422  2.2069921  2.1545528  8.0283830  1.8097174  2.1798232
## [31]  5.0653144  2.4455780  0.7277827  2.4673648  0.4879938  2.2280258
## [37]  1.9153599  1.9897989  7.6465681  1.3002320  1.9057004  1.9714550
## [43]  4.3555226  1.7075677  0.8967489  1.5452763  1.2643481  2.5619648
## [49]  1.2056382  1.1223886
```

Next, extraction of parent and children set of the class variable is done.

```r
# Initialize an empty list to store the results
allPCFeatures <- list()

# Iterate from 1 to 10
for (i in 1:length(pcSelResults$G)) {
  # Check if index i is within the range of the vector length
  if (i <= length(pcSelResults$G)) {
    # Extract the logical value at index i
    selectedLogical <- pcSelResults$G[i]
```

```
    # Extract the names and TRUE values if applicable
    if (selectedLogical) {
      featureName <- names(pcSelResults$G)[i]
      allPCFeatures[[featureName]] <- TRUE
    }
  }
}

# Extract the names of the features that are TRUE
truePCNames <- c(names(allPCFeatures), "Class")

# Extract the columns from binary_data using true_feature_names
PCData <- brcaDataBinary[, truePCNames]
PCData$Class <- brcaDataBinary$Class

# Print the first few rows of the extracted data to verify
head(PCData)
```

```
##   FIGF CD300LG SCARA5 ATP1A2 ARHGAP20 ATOH8 KLHL29 MAMDC2 CXCL2 TMEM220 Class
## 1    1       1      1      0        0     1      1      1     1       0     0
## 2    1       1      1      1        0     1      1      1     0       1     0
## 3    1       1      1      1        1     1      1      1     1       1     0
## 4    1       1      1      1        1     1      1      1     0       0     0
## 5    1       1      1      1        1     1      1      1     1       0     0
## 6    1       1      1      1        1     1      1      1     1       0     0
```

```
# Factorising the Class column to 2 levels
brcaDataBinary$Class <- factor(brcaDataBinary$Class)
PCData$Class <- factor(PCData$Class)

### Case A: Using All Features

# Set up 10-fold cross-validation on the training set
trainOnControl <- trainControl(method = "cv", number = 10)

# Train Naive Bayes classifier using all features with 10-fold cross-validation on training set
modelAllGenes <- suppressWarnings(train(Class ~ .,
                      data = brcaDataBinary,
                      method = "nb",  # Naive Bayes
                      trControl = trainOnControl))

# Get the cross-validation accuracy from the training set
accuracyAllGenes <- modelAllGenes$results$Accuracy
print(paste("Cross-validation accuracy using all features:", accuracyAllGenes[2]))
```

```
## [1] "Cross-validation accuracy using all features: 0.976053380300772"
```

```
### Case B: Using Parent and Children Features

# Train Naive Bayes classifier with 10-fold cross-validation on training set (parent and children featu
modelPCCv <- suppressWarnings(train(Class ~ .,
                    data = PCData,
```

```
                      method = "nb",  # Naive Bayes
                      trControl = trainOnControl))

# Get the cross-validation accuracy from the training set
accuracyPCCv <- modelPCCv$results$Accuracy
print(paste("Cross-validation accuracy using parent and children features:", accuracyPCCv[2]))
```

```
## [1] "Cross-validation accuracy using parent and children features: 0.984331391410378"
```

Model with All Features:

The accuracy achieved by the Naive Bayes model when using all features from the dataset is **0.9768 (about 97.6%)**. This means that the model correctly predicted the class label of the test data in 97.6% of the cases. This model uses every feature available in the dataset to make predictions.

Model with Parent and Children Features:

The accuracy achieved by the Naive Bayes model when using only the parent and children features (the key variables most directly associated with the target variable) is **0.9843 (about 98.4%)**. This shows that the model can predict class labels slightly better using only these important variables, improving accuracy by about 1% compared to using all features.

Question 5(a)

Construct the conditional probability tables for the Bayesian network based on data.

Answer

To construct probability tables, also known as conditional probability tables (CPTs), the goal is to summarize the probabilities of different outcomes for a variable based on the data. Create a contingency table for the variable. A contingency table simply counts the frequency of each unique value (or combination of values in the case of conditional probabilities) for the variable.Find the total number of observations (e.g., number of rows in the dataset). For each level or value of the variable, calculate its probability by dividing the count for that level by the total number of observations. This gives the likelihood of each outcome for the variable. Use the calculated probabilities and levels to construct the probability table.

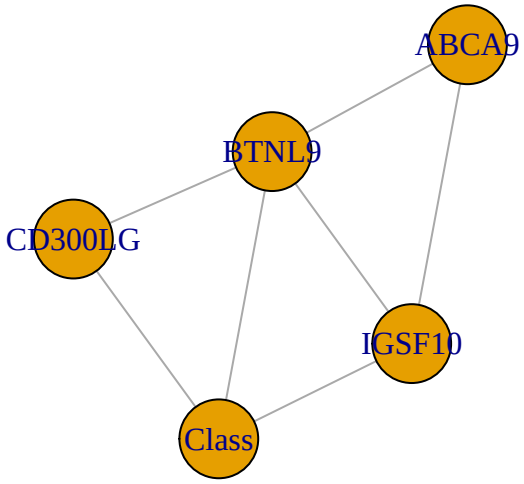
```
# Select specific columns by name
brcaDataQ5 <- brcaDataBinary[, c("BTNL9", "CD300LG", "Class", "IGSF10", "ABCA9")]

# Create a contingency table for BTNL9
counts <- table(brcaDataQ5$BTNL9)

# Calculate total number of observations
n <- sum(counts)

# Calculate probabilities for each level of BTNL9
prob_0 <- counts["0"] / n
prob_1 <- counts["1"] / n

# Define levels for BTNL9
levels <- c("0", "1")
```

```r
# Create the CPT
a <- cptable(~BTNL9,
               values = c(prob_0, prob_1),
               levels = levels)

# Get the contingency table
counts <- table(brcaDataQ5$BTNL9, brcaDataQ5$CD300LG)

# Calculate probabilities
prob_00 <- counts[1, 1] / n
prob_01 <- counts[1, 2] / n
prob_10 <- counts[2, 1] / n
prob_11 <- counts[2, 2] / n


# Create the CPT
b.a <- cptable(~CD300LG | BTNL9,
               values = c(prob_00, prob_01, prob_10, prob_11),
               levels = levels)

# Get the contingency table
counts <- table(brcaDataQ5$CD300LG, brcaDataQ5$Class)

# Calculate probabilities
prob_00 <- counts[1, 1] / n
prob_01 <- counts[1, 2] / n
prob_10 <- counts[2, 1] / n
prob_11 <- counts[2, 2] / n

c.b <- cptable(~ Class|CD300LG,
               values = c(prob_00, prob_01, prob_10, prob_11),
               levels = levels)

# Get the contingency table
counts <- table(brcaDataQ5$Class, brcaDataQ5$IGSF10)

# Calculate probabilities
prob_00 <- counts[1, 1] / n
prob_01 <- counts[1, 2] / n
prob_10 <- counts[2, 1] / n
prob_11 <- counts[2, 2] / n

d.c <- cptable(~ IGSF10|Class,
               values = c(prob_00, prob_01, prob_10, prob_11),
               levels = levels)

# Create contingency tables
table_000 <- table(brcaDataQ5$ABCA9[brcaDataQ5$IGSF10 == 0 & brcaDataQ5$BTNL9 == 0])
table_010 <- table(brcaDataQ5$ABCA9[brcaDataQ5$IGSF10 == 1 & brcaDataQ5$BTNL9 == 0])
table_001 <- table(brcaDataQ5$ABCA9[brcaDataQ5$IGSF10 == 0 & brcaDataQ5$BTNL9 == 1])
table_011 <- table(brcaDataQ5$ABCA9[brcaDataQ5$IGSF10 == 1 & brcaDataQ5$BTNL9 == 1])

# Calculate probabilities
```

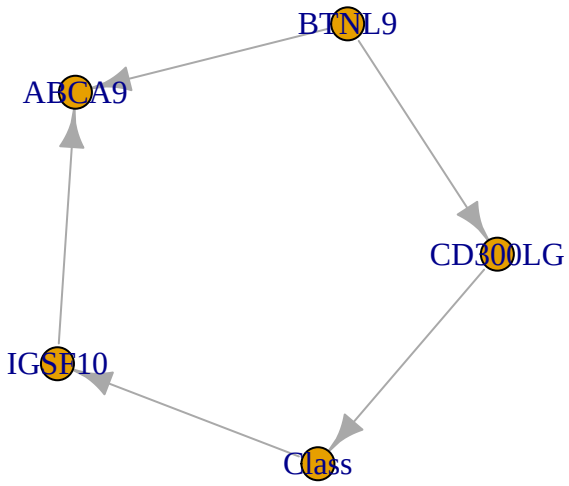
```r
n_000 <- sum(table_000)
n_010 <- sum(table_010)
n_001 <- sum(table_001)
n_011 <- sum(table_011)

prob_000 <- c(table_000[1] / n_000, table_000[2] / n_000)
prob_010 <- c(table_010[1] / n_010, table_010[2] / n_010)
prob_001 <- c(table_001[1] / n_001, table_001[2] / n_001)
prob_011 <- c(table_011[1] / n_011, table_011[2] / n_011)

# Combine probabilities into a vector
prob_values <- c(prob_000, prob_010, prob_001, prob_011)

# Create the CPT
e.da <- cptable(~ABCA9 | IGSF10:BTNL9,
                values = prob_values,
                levels = levels)

plist <- compileCPT(list(a, b.a, c.b, d.c, e.da))
plist
```

```
##  P( BTNL9 )
##  P( CD300LG | BTNL9 )
##  P( Class | CD300LG )
##  P( IGSF10 | Class )
##  P( ABCA9 | IGSF10 BTNL9 )
```

```r
# Draw the undirected network
net1=grain(plist)
plot(net1)
```




```
# Draw the directed network
plot(net1$dag)
```



Conditional Probability Table P(BTNL9)

```
# Table 1:
# BTNL9 P(BTNL9)
plist$BTNL9
```

```
## BTNL9
##         0         1
## 0.8209571 0.1790429
```

Conditional Probability Table P(CD300LG|BTNL9)

```
# Table 2:
# BTNL9 CD300LG P(CD300LG|BTNL9)
plist$CD300LG
```

```
##         BTNL9
## CD300LG           0        1
##       0 0.991959799 0.359447
##       1 0.008040201 0.640553
```

Conditional Probability Table P(Class|CD300LG)

```
# Table 3:
# CD300LG Class P(Class|CD300LG)
plist$Class
```

```
##      CD300LG
## Class           0         1
##     0 0.002816901 0.7414966
##     1 0.997183099 0.2585034
```

Conditional Probability Table P(IGSF10|Class)

```
# Table 4:
# BinaryClass IGSF10 P(IGSF10|Class)
plist$IGSF10
```

```
##       Class
## IGSF10     0          1
##      0 0.125 0.93545455
##      1 0.875 0.06454545
```

Conditional Probability Table P(ABC9|IGSF10,BTNL9)

```
# Table 5:
# BTNL9 IGSF10 P(ABC9|IGSF10,BTNL9)
plist$ABCA9
```

```
## , , BTNL9 = 0
##
##      IGSF10
## ABCA9          0         1
##     0 0.98305085 0.7843137
##     1 0.01694915 0.2156863
##
## , , BTNL9 = 1
##
##      IGSF10
## ABCA9         0          1
##     0 0.7373737 0.03389831
##     1 0.2626263 0.96610169
```

Question 5(b)

Estimate the probability of the four genes in the network having high expression levels.

Answer

A joint probability refers to the probability of multiple events happening simultaneously. In this case, the query involves computing the probability that four genes (BTNL9, CD300LG, IGSF10, and ABCA9) all have a high expression level. The joint probability is expressed as P(BTNL9 = high, CD300LG = high, IGSF10 = high, ABCA9 = high). The querygrain() function is used to query the **joint probability** for the genes BTNL9, CD300LG, IGSF10, and ABCA9. The type of query is set to "joint", indicating that the goal is to compute the joint probability for these four variables being in specific states.

The index ["1", "1", "1", "1"] refers to the event where each of the four genes (BTNL9, CD300LG, IGSF10, and ABCA9) is in the high expression state (1). Below is the probability of the four genes in the network having high expression levels.

```
# Query the joint probability of all expressions high
allGenesHigh <- querygrain(net1, nodes=c("BTNL9", "CD300LG", "IGSF10", "ABCA9"), type="joint")

#cSet evidence: BTNL9 = high, CD300LG = high, IGSF10 = high, ABCA9 = high
PAllYes <- allGenesHigh["1", "1", "1", "1"]
PAllYes
```

```
## [1] 0.07373601
```

There is only a probability of **7%** for all the four genes in the network to have high expression levels.

Question 5(c)

Estimate the probability of having cancer when the expression level of CD300LG is high and the expression level of BTNL9 is low.

Answer

This code helps us estimate the probability of having cancer based on the expression levels of two genes: CD300LG and BTNL9.

CD300LG is highly expressed (set to "1"). BTNL9 is lowly expressed (set to "0"). This information is called evidence, which means we are fixing these genes to certain values and asking the system to calculate the cancer probability based on these conditions. Below is the probability of having cancer when the expression level of CD300LG is high and the expression level of BTNL9 is low

```
# Set evidence: CD300LG = high, BTNL9 = low
evidence <- setEvidence(net1, evidence = list(CD300LG = "1", BTNL9 = "0"))

# Query the conditional probability of having cancer
queryResult <- querygrain(evidence, nodes = "Class", type = "marginal")

# Extract and print the probability of having cancer
cancerProb <- queryResult$Class["1"]
cancerProb
```

```
##         1
## 0.2585034
```

The probability of having cancer when the expression level of CD300LG is high and the expression level of BTNL9 is low is **25%**.

Question 5(d)

Prove the result in c) mathematically.

Answer

$$P(\text{cancer} = 1 \mid \text{CD300LG} = 1, \text{BTNL9} = 0)$$

$$= \frac{P(\text{BTNL9}', \text{CD300LG} \mid \text{cancer}) * P(\text{cancer})}{P(\text{BTNL9}', \text{CD300LG})}$$

→ (by Naive Bayes classifier rule)

$$= \frac{P(\text{cancer}, \text{BTNL9}', \text{CD300LG})}{P(\text{BTNL9}', \text{CD300LG})}$$ → Numerator / → Denominator

→ $(P(X|E) = P(E|X)P(X)/P(E) = P(E,X)/P(E))$

Numerator

$$P(\text{cancer}, \text{BTNL9}', \text{CD300LG}) = P(\text{cancer} \mid \text{CD300LG}, \text{BTNL9}') \times P(\text{CD300LG}, \text{BTNL9}'$$

(Relationship: $P(X_1X_2) = P(X_2 \mid X_1)P(X_1)$

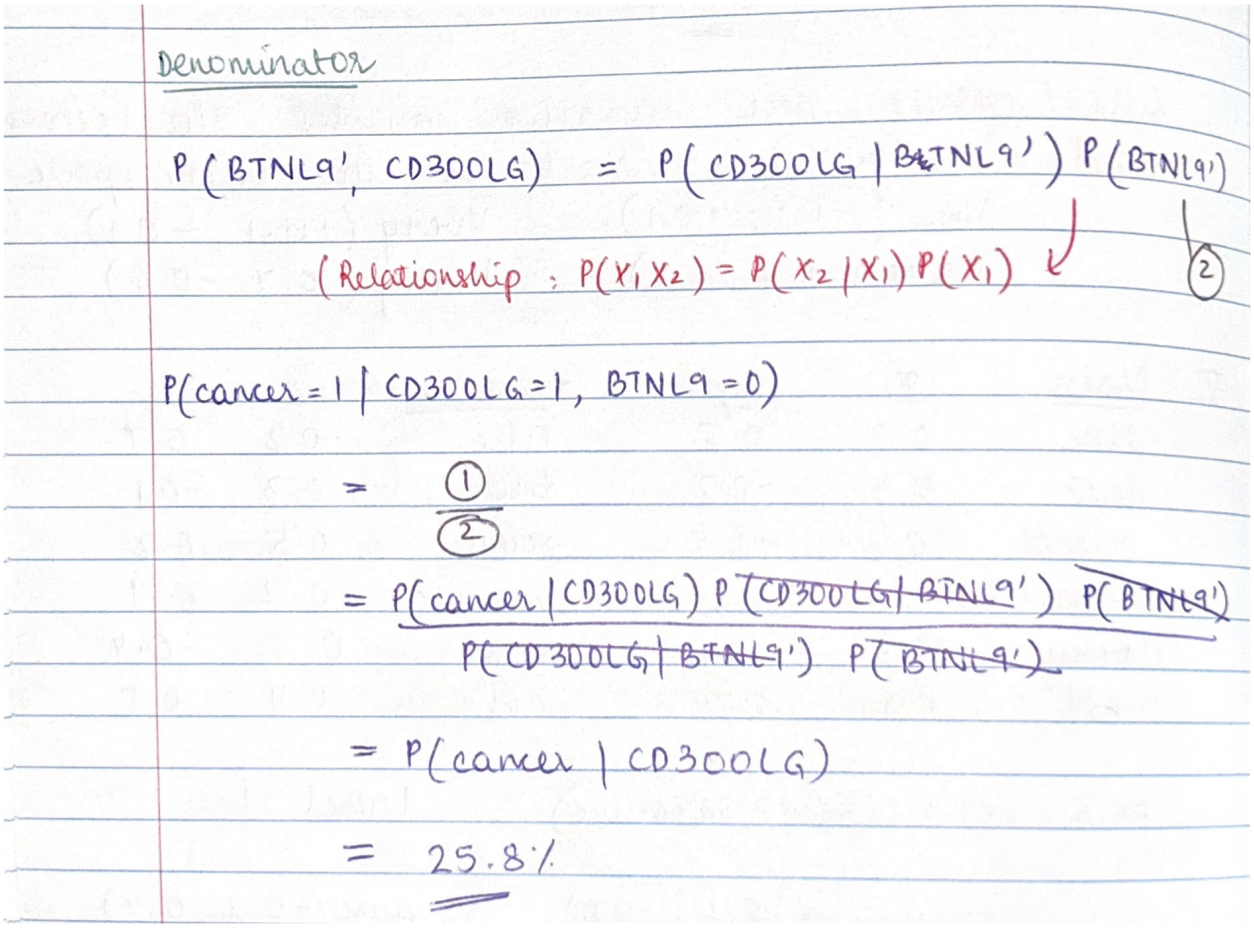
$$= P(\text{cancer} \mid \text{CD300LG}, \text{BTNL9}')P(\text{CD300LG} \mid \text{BTNL9}')P(\text{BTNL9}')$$

(Relationship: $P(X_1X_2) = P(X_2 \mid X_1)P(X_1)$

$$= P(\text{cancer} \mid \text{CD300LG})\ P(\text{CD300LG} \mid \text{BTNL9}')\ P(\text{BTNL9}')$$

↓

(Markov condition : $I(X, ND_X \mid Parent(X))$ ①



Question 5(e)

Given we know the value of CD300LG, is the "class" conditionally independent of ABCA9? And why?

Answer

Yes, due to the **Markov condition**.

The Markov condition is a fundamental concept in probabilistic graphical models, particularly in Bayesian networks. It states that in a directed acyclic graph (DAG), a variable is conditionally independent of its non-descendants given its parents. This means that once we know the value of a variable's direct causes (its parent nodes), it becomes independent of any other variables that are not directly influenced by it (its non-descendants).

In this question,

class depends on **CD300LG**.

CD300LG is influenced by **BTNL9**.

ABCA9 is also influenced by **BTNL9**.

Applying the Markov Condition, **class** is only directly influenced by **CD300LG** and not by **BTNL9** or **ABCA9**. According to the Markov condition, **class** should be conditionally independent of any variable that is not its descendant, given its parent (**CD300LG**). **ABCA9** is also indirectly influenced by **BTNL9**, but it is not directly related to **class**.

Therefore, **class** is conditionally independent of **ABCA9**.