# Data Mining 2013
# Assignment 2: Graphical Models

## Instructions

This assignment should be made in groups of two or three students. Solutions should be handed in by Friday, November 1. Send your solution by e-mail to `A.J.Feelders@uu.nl`. Your solution consists of

1. An R workspace containing the functions you have written.
2. A flat text file containing the documented program code, and
3. A PDF file containing the answers to Part 2.

Always put name and student number on your work.

## Part 1: Programming

Write a function to learn an undirected graphical model from data. To fit a single model, use the function `loglin` for fitting hierarchical log-linear models. You have to pass as arguments to `loglin` a table with the observed counts, and a list containing the margins to be fitted. The margins to be fitted can simply be indicated by giving the appropriate column numbers, for example to indicate that the margin containing the first, third and sixth column (variable) should be fitted, simply include the vector `c(1,3,6)` in the list of margins that you pass to `loglin`. You don't have to use variable names or value names. Note that in graphical models, the margins to be fitted correspond to the cliques in the graph.

Study the documentation of `loglin` for the relevant details.

The function to be written should be named `gm.search`. The first argument is the table of observed counts (to be passed to `loglin`), and the second argument is the initial graph from which the search process starts. The table of counts can be obtained by applying the `table` function to the data frame. In the call to `gm.search` the initial graph is represented as a matrix called `graph.init`, where `graph.init[i,j]=1` if there is an edge

between node $i$ and node $j$, and `graph.init[i,j]=0` otherwise. Obviously, this matrix is symmetric.

The algorithm should perform a local search starting at `graph.init`, where the neighbors of the current model are those models that can be obtained by

1. adding an edge to the current model, or
2. removing an edge from the current model.

The third argument of `gm.search` is the boolean `forward` indicating whether adding edges is allowed, and the fourth argument is the boolean `backward` indicating whether removing edges is allowed. Models are scored by AIC or BIC (the user should have a choice), where

$$\mathrm{AIC}(M) = \mathrm{dev}(M) + 2 \times \dim(M),$$

and

$$\mathrm{BIC}(M) = \mathrm{dev}(M) + \ln N \times \dim(M).$$

Here $M$ denotes an arbitrary model, $\mathrm{dev}(M)$ denotes the deviance of model $M$, and $\dim(M)$ denotes the number of parameters ($u$-terms) of $M$. This last term is included in the score in order to penalize complex models so as to avoid overfitting. The fifth argument is the string `score` which has the value "aic" if AIC scoring is wanted, and "bic" if BIC scoring is wanted.

The function `gm.search` should return a list containing the following named components:

1. `$model`: A list containing the cliques of the final model. Each clique is a vector containing the numbers of the variables in the clique.
2. `$score`: The AIC or BIC score of the final model.
3. `$trace`: A data frame providing a trace of the search process. Row $i$ of the data frame provides relevant information on step $i$ of the search.
4. `$call`: The call to the function `gm.search` that produced this result. Use the function `match.call` for this.

Here's an example call and the correct result. The `$trace` component is not shown, but I included some print statements in the function to provide information about the search process. The file `coronary.txt` can be downloaded from the course web page.

```
# Read in the data (assuming you saved it in "U:/Data Mining/coronary.txt").

> coronary.dat <- read.table("U:/Data Mining/coronary.txt",header=T)

# Perform a forward-backward search starting from the empty graph.
# Use BIC to score the models.
```

```
> coronary.graphmod <- gm.search(table(coronary.dat),matrix(0,6,6),
                                  forward=T,backward=T,score="bic")
Added: 2 - 3 (score= 457.58 )
Added: 2 - 5 (score= 415.52 )
Added: 1 - 2 (score= 380.46 )
Added: 1 - 5 (score= 351.56 )
Added: 1 - 3 (score= 331.61 )
Added: 4 - 5 (score= 326.32 )
Added: 2 - 6 (score= 321.71 )
Added: 1 - 4 (score= 320.2 )
Added: 2 - 4 (score= 315.15 )
Removed: 4 - 5 (score= 311.79 )

# Print the resulting object.

> coronary.graphmod
$cliques
$cliques[[1]]
[1] 1 2 3

$cliques[[2]]
[1] 1 2 4

$cliques[[3]]
[1] 1 2 5

$cliques[[4]]
[1] 2 6


$score
[1] 311.7889

$call
gm.search(data = table(coronary.dat), graph.init = matrix(0,6,6),
    forward = T, backward = T, score = "bic")
```

A simple hill-climbing search can get stuck in (bad) local minima. A straightforward way to try to mitigate this problem is to use random restarts from different initial models and keeping the best solution found. Write a function `gm.restart` with arguments `nstart` (the number of restarts to be performed) and `prob`, which specifies the probability of an edge between any pair of nodes (to be used in generating the random graph). So if you

pick a value close to 0 for `prob` you will get a sparse graph, and if you pick a value close to 1 you will get a dense graph. The third argument `seed` allows the user to specify a random seed so the results can be reproduced. The random seed can be set with the function `set.seed`. The remaining arguments to `gm.restart` are those that have to be passed to `gm.search` (all arguments of `gm.search` except `graph.init` which is of course determined within `gm.restart`). The function returns the output of `gm.search` for the best model found, and the call to `gm.restart`.

# Part 2: Data Analysis

In the second part you are going to analyze a data set with the functions you have written. The data set contains data from a medical study on adult patients admitted to an intensive care unit (see the course web page for the data set and documentation). It has 10 columns and 5,735 rows (check this after reading in the data in R). If this checks out, apply the `summary` function to the data frame to get some descriptive statistics of the data.

Give answers to the following questions in your report:

(a) How many different graphical models are there for this data set?

(b) How many cells does the table of counts for this data set have? How many parameters does the saturated model have?

(c) Perform a forward-backward search on this data set, starting from the empty graph (independence model). Use the BIC scoring function. List the cliques of the resulting model and draw the independence graph.

(d) Using the independence graph you found under (c), what can you say about the pairwise conditional independence between income and gender? If we are interested in predicting whether or not someone survives, looking at the independence graph, which variables appear to be sufficient for that purpose?

(e) Perform a forward-backward search on this data set, starting from the complete graph (saturated model). Use the BIC scoring function. List the cliques of the resulting model and draw the independence graph. How does it compare to the model you found under (c)?

(f) Perform a forward-backward search with AIC scoring on this data set, starting from the complete graph and empty graph. Give the cliques of the models you find, and the score of the models. Are they the same?

Compare the complexity of the models you found with BIC to those you found with AIC. Can you explain the difference?

(g) Use random restarts to see whether you can find better scoring models than you have found so far (both for AIC and for BIC scoring). Report your findings.

# Handing in the assignment

The R workspace (file with extension `.Rdata`) that you hand in should be tested to work on the Windows machines in the computer labs of the University. This file will be used to test the functions you have written.

Also put the functions you have written in a flat ascii file. Before you give the code of a function, provide the following information (start each line containing this information with the symbol #):

1. Name of the function.
2. Names and types of input arguments.
3. The result returned by the function.
4. A short description of what it does.

The two main functions should be called `gm.search` and `gm.restart`, and should be the first two functions in the file you send me. Any other functions you have written that are needed should be listed below that. Your report on the analysis of the data set (containing the answers to part II) should be handed in in PDF format.

# Remarks and Hints

- To score models you need their deviance and complexity (number of parameters). The deviance is returned by `loglin` in the component named `$lrt`. This is short for likelihood ratio test, the name of the test that uses the deviance as the test statistic (see the lecture slides and lecture notes). `loglin` also returns the degrees of freedom for the likelihood ratio test, which is the number of parameters ($u$-terms) excluded from the model compared to the saturated model. You can use this information to determine the value for $\dim(M)$ in computing AIC or BIC.
- As we mentioned, in graphical models, the margins to be fitted correspond to the cliques in the graph. Hence, you need to implement an algorithm for finding the cliques of a graph. The Bron-Kerbosch algorithm with pivot selection is quite easy to implement for example.
- Note that we cannot handle data sets with too many variables, because the `table` function will start to complain that we try to create a table with too many cells. Since the data has to be passed to `loglin` as a table, we cannot avoid this problem, unless we write our own IPF implementation.