Project 1: Bayesian Structure Learning

Karthik Pythireddi karthik9@stanford.edu
AA228/CS238, Stanford University

1. Algorithm

When we think of learning the structure of the models from data, we prefer to maximize the probability of a graphical structure and also find ways to search their space in an efficient manner given their too large space. Firstly, we want to be able to calculate the score on a network structure G which involves the maximum posteriori approach. In this approach, we find the G which maximizes the probability i.e $P(G|D)$. We use the Bayesian Score to measure how well the graph (G) models the data (D).

$$P(G|D) = P(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})} \tag{1}$$

The values of the α_{ijk} are considered as pseudo-counts and m_{ijk} are the counts. We have the equations as: $\alpha_{ij0} = \sum_{k=1}^{r_i} \alpha_{ijk}$, $m_{ij0} = \sum_{k=1}^{r_i} m_{ijk}$. In this approach, finding the G which maximizes is called as *Bayesian Score*:

$$\log P(G|D) = \log P(G) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} (\log(\frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})}) + \sum_{k=1}^{r_i} (\log \frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})})) \tag{2}$$

Here the Bayesian score is computed numerically as adding the log values in small numbers is easy than the product of all these numbers. We often use the Uniform Prior in our calculations and that makes the $P(G)$ to be dropped out of the Bayesian score computations. The benefit of using the Bayesian score to optimize the structure is that we can have a right balance between the model complexity, given the available data. Usually, larger data sets tend to improve the bayesian scores significantly by providing better estimates of the network parameters. We used Julia as a programming language and referenced the algorithmic code from the text book. We used the functions prior, statistics from the earlier lectures to supplement the function bayesian score.

In our implementation we search the space of directed acyclic graphs for the one that maximizes the bayesian score. Bayesian network structure space grows exponentially with increase in the number of nodes. We used a search strategy as we cannot enumerate the all possible structures with highest scoring network.In the current implementation we used a K2 Search which is a most commonly used search algorithm. Some context about K2 Search, it runs in polynomial time and it can't guarantee finding a globally optimal network structure. K2 is often used with the bayesian score as it can balance the complexity of the graph/model with the available data or provided data. It starts out as an graph with no edges and iteratively adds the parents to the nodes greedily. The addition of the parents to the nodes is done such that the score increases to maximum. We can notice from the section 2 of this write up, where the computation time increases with the increase in the size of the data set. As the size of the dataset increases the reliability of the bayesian score increases as the network has more data points to learn from which can help in modeling the network structures more accurately. Detailed implementation of the algorithm can be found in the code section. We used the K2 search as the algorithm for the Directed Graph Search, this project can further be extended by implementing the local search algorithms like Simulated annealing, Genetic Algorithms and Tabu Search etc.

2. Bayesian Score and Run Time

DataSet	Bayesian Score	Run time
small.csv	-3835.67942521279	7.534369 seconds
medium.csv	-42060.47640776388	9.882106 seconds
large.csv	-408087.15267893655	570.204144 seconds

Table 1: Caption

3. Graphs

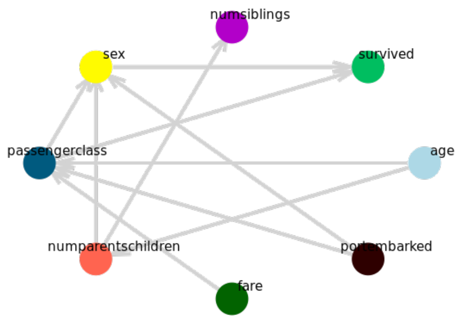

Figure 1: Graph "G" for the dataset "small.csv".

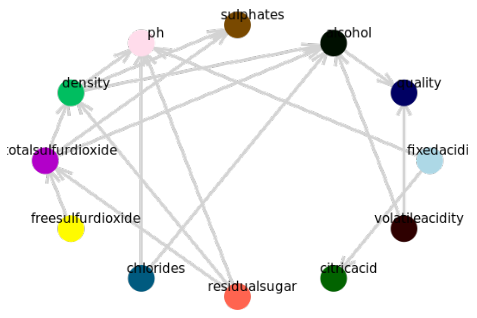

Figure 2: Graph "G" for the dataset "medium.csv".

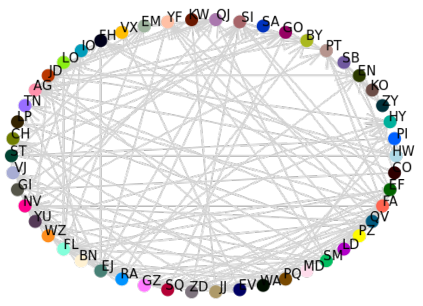

Figure 3: Graph "G" for the dataset "large.csv".

4. Code

```
"""
Many of the functions used in this project have been referred from the
    Decision Making Under Uncertainty by Mykel J. Kochenderfer, Tim A. Wheeler
    , Kyle H. Wray
"""


#import the libraries
using CSV
using DataFrames
using GraphPlot, Graphs
using GraphRecipes
using Printf
using Compose
using Cairo
using LinearAlgebra
using SpecialFunctions
using Colors

#struct to store the variable information
struct Variable
    name::Symbol
    r::Int # number of possible values
end

#struct to store the ordering information
struct K2Search
    ordering::Vector{Int}
end


"""
    write_gph(dag::DiGraph, idx2names, filename)
```

```
Takes a DiGraph, a Dict of index to names and a output filename to write the
    graph in `gph` format.
"""
function write_gph(dag::DiGraph, idx2names, filename)
    open(filename, "w") do io
        for edge in edges(dag)
            @printf(io, "%s,%s\n", idx2names[src(edge)], idx2names[dst(edge)
    ])
        end
    end
end

#function to calculate the prior probability
function prior(vars, G)
    n = length(vars)
    r = [vars[i].r for i in 1:n]
    q = [prod([r[j] for j in inneighbors(G, i)]) for i in 1:n]

    return [ones(q[i], r[i]) for i in 1:n]
end

#function to convert the subscripts to indices
function sub2ind(siz, x)
    k = vcat(1, cumprod(siz[1:end-1]))
    return dot(k, x .- 1) .+ 1
end

#function to calculate the statistics of the data
function statistics(vars, G, data_T)
    n = size(data_T, 1)
    r = [vars[i].r for i in 1:n]
    q = [prod([r[j] for j in inneighbors(G, i)]) for i in 1:n]
    M = [zeros(q[i], r[i]) for i in 1:n]
    for o in eachcol(data_T)
        for i in 1:n
            k = o[i]
            parents = inneighbors(G, i)
            j = 1
            if !isempty(parents)
                j = sub2ind(r[parents], o[parents])
            end
            M[i][j, k] += 1.0
        end
    end
    return M
end


#function to calculate the bayesianscore component
```

```
function bayesianscore_component(M,alpha)

    p = sum(loggamma.(alpha + M))
    q = sum(loggamma.(alpha))
    r = sum(loggamma.(sum(alpha,dims=2)))
    s = sum(loggamma.(sum(alpha,dims=2) + sum(M,dims=2)))

    return p - q + r - s
end

#function to calculate the bayesian score
function bayesianscore(vars, G, data_T)
    n = length(vars)
    M = statistics(vars, G, data_T)
    alpha = prior(vars, G)
    return sum(bayesianscore_component(M[i], alpha[i]) for i in 1:n)
end

#function to fit the graph  using K2 search
function fit(method::K2Search, vars, data_T)
    G = SimpleDiGraph(length(vars))
    for (k,i) in enumerate(method.ordering[2:end])
        y = bayesianscore(vars, G, data_T)
        while true
            y_best, j_best = -Inf, 0
            for j in method.ordering[1:k]
                if !has_edge(G, j, i)
                    add_edge!(G, j, i)
                    y_ = bayesianscore(vars, G, data_T)
                    if y_ > y_best
                        y_best, j_best = y_, j
                    end
                    rem_edge!(G, j, i)
                end
            end
            if y_best > y
                y = y_best
                add_edge!(G, j_best, i)
            else
                break
            end
        end
    end
    return G
end

#function to read the csv file and return the data frame
function compute(infile, outfile)

    # WRITE YOUR CODE HERE
    # FEEL FREE TO CHANGE ANYTHING ANYWHERE IN THE CODE
    # THIS INCLUDES CHANGING THE FUNCTION NAMES, MAKING THE CODE MODULAR,
    BASICALLY ANYTHING
    csv_file = CSV.File(infile)
    df = DataFrame(csv_file)
    variable_names = names(df)
    data = Matrix(df)
    data_T = transpose(data)

    #initalize the vars array with the variable information
    vars = []
    for (idx, variable_name) in enumerate(variable_names)
        r = maximum(data_T[idx, :])  # Access the column by its integer index
     in the transposed matrix
        push!(vars, Variable(Symbol(variable_name), r))
    end

    #initalize the node_dict and num_nodes
    node_dict = Dict(zip(1:length(variable_names), variable_names))
    num_nodes = length(variable_names)

    #calculate the G using K2 search and plot using the gplot
    G = fit(K2Search(1:num_nodes), vars, data_T)
    nodefillc = distinguishable_colors(num_nodes, colorant"lightblue")
    p = gplot(G; nodelabel=[node_dict[i] for i in 1:length(variable_names)],
    nodefillc=nodefillc, nodelabeldist=1.5, layout=circular_layout)
    draw(PNG("small_graph.png"), p)

    #bayesian score for the above graph
    println(bayesianscore(vars, G, data_T))

    # node_dict = Dict(zip(1:num_nodes, variable_names))
    write_gph(G, node_dict, outfile)

end

inputfilename = "/home/karthikpythireddi/Stanford_Classes/AA228/AA228-CS238-
    Student/project1/data/small.csv"
outputfilename = "/home/karthikpythireddi/Stanford_Classes/AA228/AA228-CS238-
    Student/project1/small.gph"

@time compute(inputfilename, outputfilename) # compute function and the time
    taken to run it
```

References: Decision Making Under Uncertainty by Mykel J. Kochenderfer, Tim A. Wheeler, Kyle H. Wray