Project 1: Bayesian Structure Learning

Yu-Wei Lin ywlin@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

This code implements a Bayesian network structure learning algorithm using the K2 method. It starts by reading data from a CSV file and processing it to convert categorical values into numeric indices that can be used for statistical calculations. Each variable in the dataset is tracked with its name and the number of possible values it can take.

The heart of the code is the K2 algorithm, which learns the structure of a Bayesian network. It works with a predetermined ordering of variables and systematically builds a directed acyclic graph by adding parent nodes that maximize a Bayesian score. This score is calculated using count statistics from the data and Dirichlet priors, incorporating gamma functions for probability computations. The final output is written to a .gph file as a list of edges that represent the learned network structure.

The code is structured to be modular and flexible, with separate functions handling data reading, variable preparation, statistical calculations, and the core K2 search algorithm. This makes it easier to modify or extend different parts of the implementation as needed.

2. Graphs

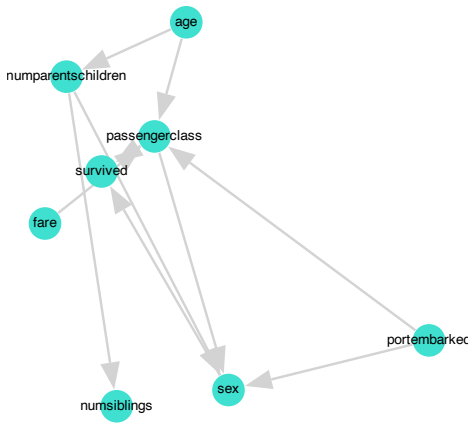

Figure 1: small.pdf (runtime: 1.44 sec)

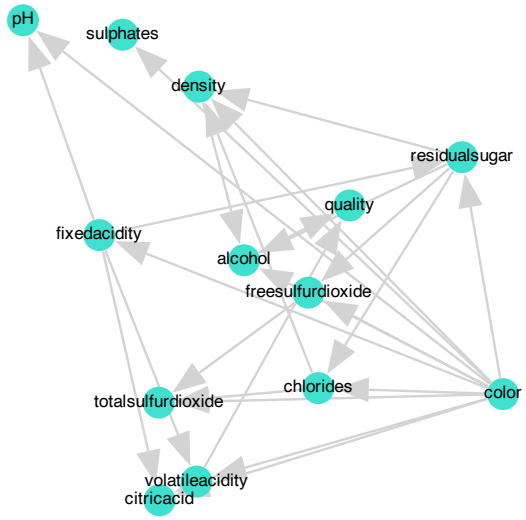

Figure 2: medium.pdf (runtime: 2.66 sec)

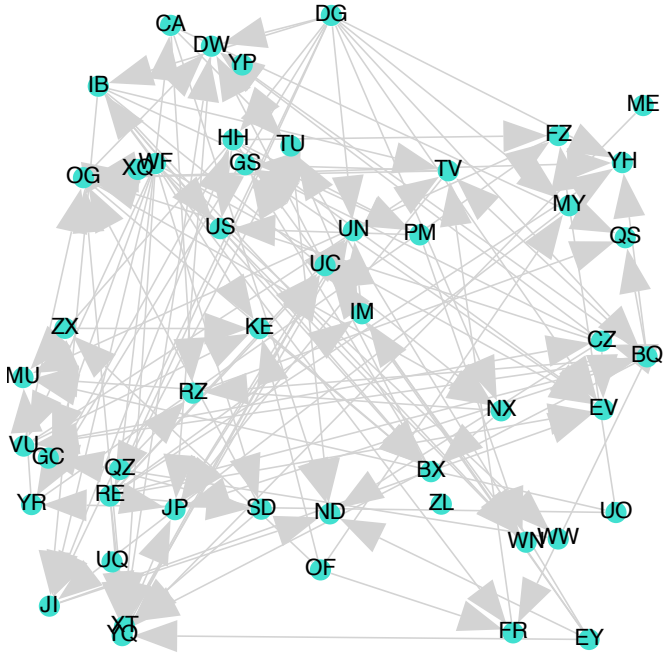

Figure 3: large.pdf (runtime: 154.24 sec)

3. Code

```
import Pkg
Pkg.add("Graphs")
Pkg.add("Printf")
Pkg.add("CSV")
Pkg.add("DataFrames")
Pkg.add("SpecialFunctions")
Pkg.add("LinearAlgebra")
Pkg.add("GraphPlot")
Pkg.add("Compose")
Pkg.add("Cairo")

using Graphs
using Printf
using CSV
using DataFrames
using SpecialFunctions
using LinearAlgebra
using GraphPlot
using Compose
using Cairo

# Structure to hold variable information
struct Variable
    name::String
    r::Int  # number of possible values
end

"""
    read_data(filename)

Reads CSV data and returns a DataFrame and a mapping of variable names to
    indices.
"""
function read_data(filename)
    df = CSV.read(filename, DataFrame)
    vars = names(df)
    idx2name = Dict(i => name for (i, name) in enumerate(vars))
    return df, vars, idx2name
end

"""
    prepare_variables(df)

Prepares Variable structures for each column in the dataset.
"""
function prepare_variables(df)
    variables = Variable[]
    for col in names(df)
        # # Get unique values to determine possible values count
```

```
        # r = length(unique(df[!, col]))
        # push!(variables, Variable(String(col), r))

        # Get unique values to determine possible values count
        unique_vals = unique(df[!, col])
        # Remap values to 1:r for consistent indexing
        value_map = Dict(val => i for (i, val) in enumerate(unique_vals))
        df[!, col] = map(x -> value_map[x], df[!, col])
        r = length(unique_vals)
        push!(variables, Variable(String(col), r))
    end
    return variables, df
end

function sub2ind(siz, x)
    k = vcat(1, cumprod(siz[1:end-1]))
    return dot(k, x .- 1) + 1
end

function statistics(vars, G, D::Matrix{Int})
    n = size(D, 2)
    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(Int, q[i], r[i]) for i in 1:n]
    for o in eachrow(D)
        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 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

"""
    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 plot_gph(dag::DiGraph, node_names, outplot)
    draw(PDF(outplot, 20cm, 20cm),
    gplot(dag,
         nodelabel=node_names,
         layout=random_layout,
         NODELABELSIZE=6.0,
         arrowlengthfrac=0.15))
end

function bayesian_score_component(M, a)
    p = sum(loggamma.(a + M))
    p -= sum(loggamma.(a))
    p += sum(loggamma.(sum(a,dims=2)))
    p -= sum(loggamma.(sum(a,dims=2) + sum(M,dims=2)))
    return p
end

function bayesian_score(vars, G, D)
    n = length(vars)
    M = statistics(vars, G, D)
    a = prior(vars, G)
    return sum(bayesian_score_component(M[i], a[i]) for i in 1:n)
end

struct K2Search
    ordering::Vector{Int} # variable ordering
end

function fit(method::K2Search, vars, D)
    G = SimpleDiGraph(length(vars))
    for (k,i) in enumerate(method.ordering[2:end])
        y = bayesian_score(vars, G, D)
        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' = bayesian_score(vars, G, D)
                    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 compute(infile, outfile, outplot)
    runtime = @elapsed begin
    # 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
    # Read the data
    df, node_names, idx2name = read_data(infile)

    # Prepare variables
    vars, processed_df = prepare_variables(df)

    # Convert DataFrame to Matrix{Int}
    # data_matrix = Matrix{Int}(df)
    data_matrix = Matrix{Int}(processed_df)  # Transpose to get variables as
    rows

    # Define variable ordering for K2 search
    # Here we use the original order from the DataFrame
    ordering = collect(1:length(vars))

    # Create K2Search instance
    k2 = K2Search(ordering)

    # Fit the Bayesian network
    dag = fit(k2, vars, data_matrix)

    # Write the resulting graph to file
    write_gph(dag, idx2name, outfile)

    #plot
    plot_gph(dag, node_names, outplot)
    end
    println(runtime)
end
```

```
if length(ARGS) != 3
    error("usage: julia project1.jl <infile>.csv <outfile>.gph")
end

inputfilename = ARGS[1]
outputfilename = ARGS[2]
outputplotname = ARGS[3]

compute(inputfilename, outputfilename, outputplotname)
```