Project 1: Bayesian Structure Learning

Lisa Fung lisafung@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

I used the K2 Search Algorithm with hyperparameter `max_parents` and random initial orderings of the nodes.

Dataset	Max Parents	Random Orderings	Bayesian Score	Running Time (sec)
Small	3	5	-3,814.082	0.103
Medium	3	5	-97,006.101	2.651
Large	4	2	-428,847.924	287.05

For small and medium datasets, I performed K2 Search with `max_parents` $= 1, 2, 3$ with the default CSV file ordering of nodes as well as 5 random initial permutations of nodes. For both datasets, one of the 5 random permutations and `max_parents` $= 3$ yielded the greatest Bayesian score. For the large dataset, I performed K2 Search with `max_parents` $= 1, 2, 3, 4, 5$ with the default CSV file ordering of nodes and 2 random initial permutations of nodes with `max_parents` $= 4$, one of which yielded the greatest Bayesian score on the large dataset.

2. Graphs

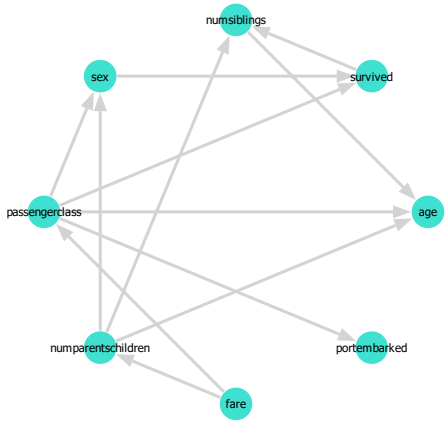

Figure 1: Small Graph

3. Code

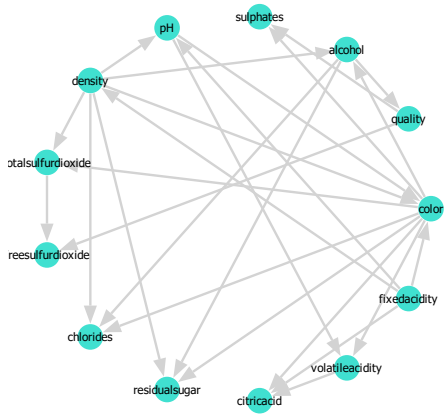

Figure 2: Medium Graph

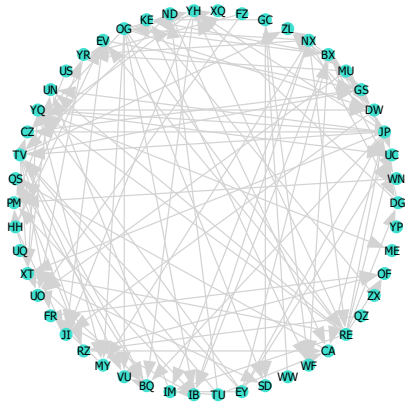

Figure 3: Large Graph

```
"""
Project 1: K2 Search
Lisa Fung
Last updated: Saturday 10/19/2024
"""

using Graphs
using Printf
using CSV
using DataFrames

using LinearAlgebra     # for dot product
using SpecialFunctions  # for loggamma
```

```
using GraphPlot         # for plotting graphs
using Compose, Cairo, Fontconfig    # for saving graphs

using Random            # for random node orderings in K2 Search

"""
Part 1: Read in dataset D and extract statistics (counts) from D.
"""

# Variable structure for all variables in Bayesian network
struct Var
    name::String
    r::Int # number of possible assignments to variable
end

"""
Read in CSV data file as DataFrame.
Returns:
    D : n x m matrix of dataset.
        n = number of variables
        m = number of observations (data points)
    vars : array of length n with variables
        Vector{Variable}
"""
function read_data(datafile)
    df = DataFrame(CSV.File(datafile))

    # Maximum value of each variable
    max_values = vec(Matrix(combine(df, All() .=> maximum)))

    # Create variables with name and number of possible assignments
    vars = [Var(name, r) for (name, r) in zip(names(df), max_values)]

    D = transpose(Matrix(df))   # Dataset

    return D, vars
end

"""
Returns linear index given subscript index.
Inputs:
    siz : size of multi-dimensional array
    subscript : subscript/Cartesian coordinates of element
Output:
    corresponding integer linear index
"""
function sub2ind(siz, subscript)
    # Given siz (n1, n2, ..., nt) and subscript (x1, x2, ..., xt)
```

```
    # Position in serialized array is (x1 + x2*n1 + x3*n1*n2 + ... + xt*n1
    *...*n{t-1})
    mult = [1; cumprod(siz[1:end-1])]     # [1, n1, n1*n2, ..., n1*...*n{t-1}
    ]

    # Must convert subscript to be 0-indexed, dot product, then convert to 1-
    indexed
    return dot(mult, subscript .- 1) + 1
end

"""
Extract statistics (counts) from dataset.
Inputs:
    D : n x m matrix of dataset.
        n = number of variables
        m = number of observations (data points)
    vars : array of length n with variables
        Vector{Variable}
    G : directed graph with nodes and edges

Returns:
    M : array of length n
        M[i] : q[i] x r[i] matrix of counts for each parental instantiation
                and assignment combination of vars[i]
"""
function statistics(D, vars, G)
    n = length(vars)    # Number of variables
    r = [vars[i].r for i in 1:n]    # Number of assignments for each variable
    # Number of parental instantiations for each variable
    q = [prod([vars[parent].r for parent in inneighbors(G, i)]) for i in 1:n]

    M = [zeros(q[i], r[i]) for i in 1:n]

    # Linear indices from Subscript indices
    # linear_indices = LinearIndices()
    # println(linear_indices)

    for o in eachcol(D)     # Iterate thru each observation
        for i in 1:n        # Iterate thru each variable
            k = o[i]        # Assignment to vars[i]
            parents = inneighbors(G, i)
            j = 1           # Parental instantiation, default q[i] is 1
            if !isempty(parents)
                # Get linear index of parental assignments, o[parents], from
    subscript
                j = sub2ind(r[parents], o[parents])
            end
            M[i][j, k] += 1.0
        end
    end
```

```
    return M
end

"""
Part 2: Compute Bayesian Score for a Graphs
"""

"""
Compute uniform prior for pseudocounts (alpha's).
"""
function prior(D, vars, G)
    n = length(vars)
    r = [vars[i].r for i in 1:n]
    q = [prod([r[parent] for parent in inneighbors(G, i)]) for i in 1:n]
    return [ones(q[i], r[i]) for i in 1:n]
end

"""
Compute Bayesian score component for each variable
"""
function bayesian_score_component(M_i, alpha_i)
    # Sum over all parental instantiations q[i]
    p = sum(loggamma.( sum(alpha_i, dims=2) ))
    p -= sum(loggamma.( sum(alpha_i, dims=2) + sum(M_i, dims=2) ))

    # Sum over all elements
    p += sum(loggamma.(alpha_i + M_i))
    p -= sum(loggamma.(alpha_i))    # Equals 0 due to uniform prior, loggamma
    (1) = 0

    return p
end

"""
Compute Bayesian Score given graph, data, vars
Inputs:
    D : n x m matrix of dataset.
        n = number of variables
        m = number of observations (data points)
    vars : array of length n with variables
        Vector{Variable}
    G : directed graph with nodes and edges

Output:
    Returns Bayesian score (log version)
"""
function bayesian_score(D, vars, G)
    n = length(vars)
    M = statistics(D, vars, G)
    alpha = prior(D, vars, G)
```

```
    return sum([bayesian_score_component(M[i], alpha[i]) for i in 1:n])
end


"""
Part 3: K2 Search with limited number of parents (2)
"""

"""
Perform K2 Search on available graphs with prespecified ordering.
K2 Search only adds edges from earlier parent nodes to later child nodes in
    ordering
"""
struct K2Search
    order::Vector{Int}  # Ordering of nodes
end

"""
Fit function for K2 Search.
Inputs:
    method::K2Search : method structure with ordering of nodes
    vars::Vector{Variable} : array of length n with variables
    D : n x m matrix of dataset.
        n = number of variables
        m = number of observations (data points)
    max_parents : maximum number of parents any node can have

Output:
    G : best graph by Bayesian Score given K2 Search ordering
"""
function fit(method::K2Search, vars, D, max_parents=2)
    # Initialize graph with only nodes, no edges
    G = SimpleDiGraph(length(vars))

    score = bayesian_score(D, vars, G)
    # Iterate through all nodes in K2Search ordering to add parents
    for (node_idx, i) in enumerate(method.order[2:end])
        for _ in 1:max_parents
            # Determine which node as parent j is best
            score_best, parent_best = -Inf, 0
            for j in method.order[1:node_idx]
                if !has_edge(G, j, i)
                    add_edge!(G, j, i)
                    score_j = bayesian_score(D, vars, G)
                    if score_j > score_best
                        score_best = score_j
                        parent_best = j
                    end
                    rem_edge!(G, j, i)
                end
```

```
            end

            # Add edge from best parent if Bayesian Score improves
            if score_best > score
                score = score_best
                add_edge!(G, parent_best, i)
            else
                # No improvement by parents, move onto next node in ordering
                break
            end
        end
    end

    return G
end

"""
Part 4: Save graph
"""

"""
    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


"""
Main code
"""
dataset = "small"

inputfilename = string("project1/data/", dataset, ".csv")
D, vars = read_data(inputfilename)
node_names = [var.name for var in vars]
G_baseline = SimpleDiGraph(length(vars))    # Baseline graph with no edges
println("Baseline Bayes Score: ", bayesian_score(D, vars, G_baseline))

default_order = [i for i in 1:length(vars)]     # Default ordering
# Seeded random orderings
random_orders = unique([randperm(Xoshiro(i+137), length(vars)) for i in 1:5])
```

```
"""
Test output of different max_parents with K2 Search.
1. Default ordering of nodes in dataset
2. Random permutations of nodes for ordering
    a. Test 5 random permutations, max_parents (1:3) for small/medium dataset
    , record and compare performance
    b. Increase max_parents until level out performance for large dataset
"""

function K2_eval(order_list, do_plot=false)
    K2_graphs = []
    K2_score_best = -Inf
    K2_G_best = G_baseline

    for (i, order) in enumerate(order_list)
        for max_parents in 1:3
            println("Order: ", i, ", max_parents: ", max_parents)
            @time begin
                G_curr = fit(K2Search(order), vars, D, max_parents)
            end
            score_curr = bayesian_score(D, vars, G_curr)
            push!(K2_graphs, G_curr)
            println("Bayes Score (order: ", i, ", max_parents: ", max_parents
    , "): ", score_curr)
            if score_curr > K2_score_best
                K2_score_best = score_curr
                K2_G_best = G_curr
            end

            if do_plot
                plot = gplot(G_curr, layout=circular_layout, nodelabel=
    node_names)
                draw(PDF(string("project1/outputs_", dataset, "/
    K2_plot_G_order_random", i, "_max_parent_", max_parents, ".pdf"), 16cm, 16
    cm), plot)
                write_gph(G_curr, node_names, string("project1/outputs_",
    dataset, "/K2_G_order_random_max_parent_", max_parents, ".gph"))
            end
        end
        println()
    end

    return K2_graphs, K2_G_best, K2_score_best
end

K2_graphs, K2_G_best, K2_score_best = K2_eval(random_orders[1], true)

# Plot and write best graph
plot = gplot(K2_G_best, layout=circular_layout, nodelabel=node_names)
```

```
draw(PDF(string("project1/outputs_", dataset, "/K2_best_plot_G_order_random.
    pdf"), 16cm, 16cm), plot)
write_gph(K2_G_best, node_names, string("project1/outputs_", dataset, "/
    K2_best_G_order_random.gph"))
println("Best Bayesian Score from K2 Random: ", K2_score_best)
```