Project 1: Bayesian Structure Learning

Neha Balamurugan nbalamur@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

The algorithm used here is a simple K2 search. It starts by creating a graph with nodes and no edges. Then, it calculates the Bayesian Score for each node, which tells how well that node's current connections explain the data. The algorithm then adds edges between nodes using the K2 method, which looks for the best parent nodes for each variable. It adds connections between nodes in a way that improves the score without creating cycles in the graph. After this, the algorithm fine-tunes the graph by switching the direction of some edges to see if it improves the score further, while still ensuring the graph stays acyclic. It keeps adjusting the graph until no more improvements can be made, then saves and visualizes the final graph, which reveals the relationships between variables in the dataset.

2. Graphs

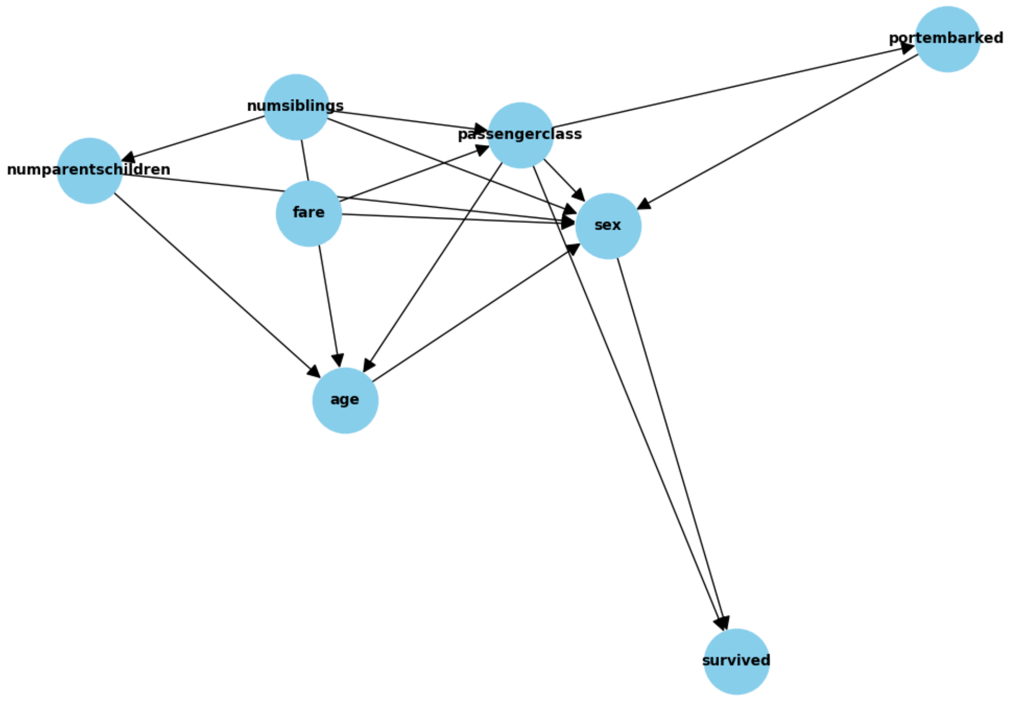

Figure 1: Structure of small.csv (Compute time: 5s)

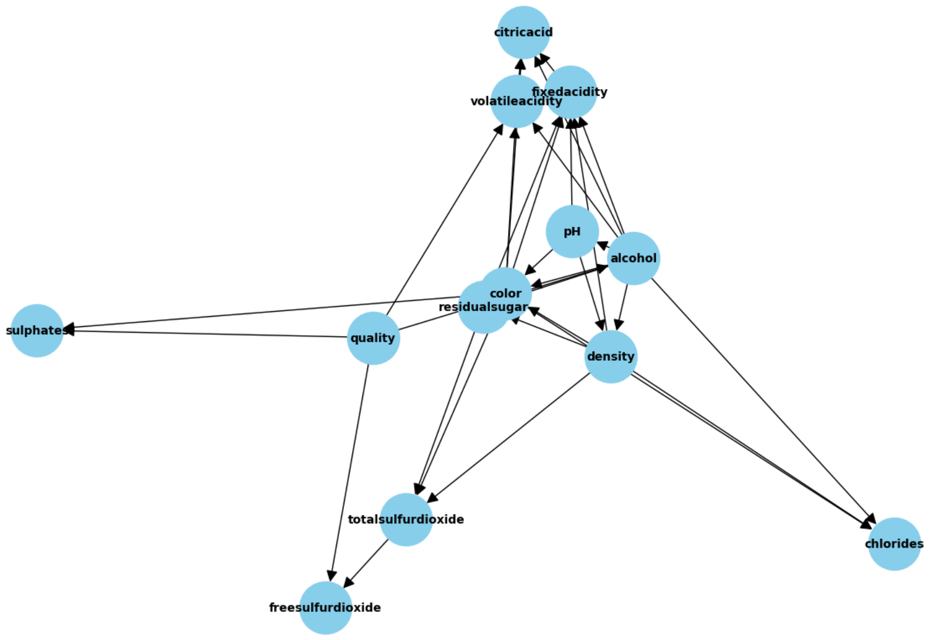

Figure 2: Structure of medium.csv (Compute time: 121s)

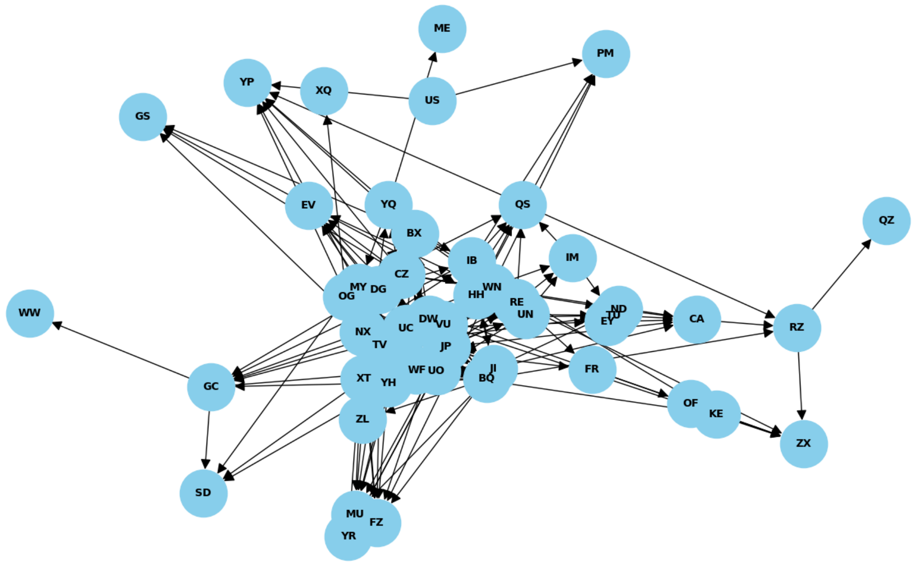

Figure 3: Structure of large.csv (Compute time: 4836s)

3. Code

```
import logging
import random
import sys
import itertools
import tempfile
import graphviz
import networkx as nx
import os
import pandas as pd
import time
from networkx.drawing.nx_pydot import write_dot
from scipy.special import gammaln
import matplotlib.pyplot as plt

def save_graph_to_file(graph, filename):
    """Write the graph's edges to a file in 'parent, child' format."""
    edges = ['{},{}\n'.format(source, target) for source, target_list in nx.
    to_dict_of_lists(graph).items() for target in target_list]
    with open(filename, 'w') as file:
        file.writelines(edges)

def create_graph_from_data(dataframe):
    """Initialize a directed graph from the columns of a dataframe."""
    graph = nx.DiGraph()
    graph.add_nodes_from(list(dataframe.columns))
    return graph

def render_graph(graph, output_filename):
    """Render and save the graph as an image."""
    with tempfile.NamedTemporaryFile() as temp_file:
        write_dot(graph, temp_file.name)
        rendered_image = graphviz.render('dot', 'png', temp_file.name)
        os.rename(rendered_image, output_filename)

def bayesian_score_for_node(graph, node, data):
    """Compute the local Bayesian score for a given node."""
    score = 0
    max_value = max(data[node])
    parents = sorted(graph.predecessors(node))
    parent_instantiations = itertools.product(*[data[parent].unique() for
    parent in parents])

    for inst in parent_instantiations:
        temp_data = data.copy()
        for i, parent in enumerate(parents):
            temp_data = temp_data[temp_data[parent] == inst[i]]

        score += gammaln(max_value) - gammaln(max_value + len(temp_data))
        for value in range(max_value):
            count = len(temp_data[temp_data[node] == value + 1])
            score += gammaln(1 + count)

    return score

def total_bayesian_score(graph, data):
    """Compute the total Bayesian score for the graph."""
    return sum(bayesian_score_for_node(graph, node, data) for node in graph)

def optimize_k2(graph, data, max_parents):
    """Optimize the graph using the K2 algorithm."""
    logging.info('Running K2 algorithm...')
    scores = {node: bayesian_score_for_node(graph, node, data) for node in
    graph}

    for node in graph.nodes:
        logging.info('Optimizing parent set for node: {}'.format(node))
        parents = []

        while len(parents) < max_parents:
            current_score = sum(scores.values())
            best_parent = None
            best_score = scores[node]

            for potential_parent in graph.nodes:
                if potential_parent == node or potential_parent in graph.
    predecessors(node):
                    continue

                temp_graph = graph.copy()
                temp_graph.add_edge(potential_parent, node)

                if nx.is_directed_acyclic_graph(temp_graph):
                    new_score = bayesian_score_for_node(temp_graph, node,
    data)
                    if new_score > best_score:
                        best_score = new_score
                        best_parent = potential_parent

            if best_parent:
                graph.add_edge(best_parent, node)
                parents.append(best_parent)
                scores[node] = best_score
                logging.info('Updated score: {}'.format(sum(scores.values()))
    )
            else:
                break

def optimize_edge_directions(graph, data):
    """Optimize edge directions in the graph."""
    scores = {node: bayesian_score_for_node(graph, node, data) for node in
    graph}

    for edge in list(graph.edges):
        parent, child = edge
        current_score = sum(scores.values())

        temp_graph = graph.copy()
        temp_graph.remove_edge(parent, child)
        temp_graph.add_edge(child, parent)

        if nx.is_directed_acyclic_graph(temp_graph):
            new_parent_score = bayesian_score_for_node(temp_graph, parent,
    data)
            new_child_score = bayesian_score_for_node(temp_graph, child, data
    )

            if new_parent_score + new_child_score > scores[parent] + scores[
    child]:
                graph.remove_edge(parent, child)
                graph.add_edge(child, parent)
                scores[parent] = new_parent_score
                scores[child] = new_child_score
                logging.info('Swapped edge {} -> {}: new score {}'.format(
    parent, child, sum(scores.values())))

def compute_graph(input_file, output_file):
    """Main function to compute the graph structure."""
    data = pd.read_csv(input_file)
    graph = create_graph_from_data(data)

    start_time = time.time()

    while True:
        initial_score = total_bayesian_score(graph, data)
        optimize_k2(graph, data, max_parents=1)
        optimize_edge_directions(graph, data)
        if initial_score == total_bayesian_score(graph, data):
            break

    save_graph_to_file(graph, output_file)
    render_graph(graph, input_file.replace('csv', 'png'))

    end_time = time.time()
    logging.info('Computation completed in {} seconds.'.format(end_time -
    start_time))

def setup_logging():
    """Configure logging settings."""
    logging.basicConfig(level=logging.INFO, format='%(asctime)s [%(levelname)
    s] %(message)s', datefmt='%Y-%m-%d %H:%M:%S')

def main():
    if len(sys.argv) != 3:
        raise Exception("Usage: python script.py <inputfile>.csv <outputfile
    >.gph")

    input_file = sys.argv[1]
    output_file = sys.argv[2]
    compute_graph(input_file, output_file)

if __name__ == '__main__':
    setup_logging()
    main()
```