Project 1: Bayesian Structure Learning

Paul-Emile Giacomelli pegiaco@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

My strategy was the following:

1. Code the Bayesian score correctly. Test it with the example.csv and example.gph files.
2. As a first approach, try tho find the graph with the best bayesian score with the K2 algorithm for the three datasets.
3. As the K2 algorithm was not fast enough on the large dataset, I used a local search as this approach would find a graph much faster.

I tested my Bayesian score function on the example.csv and example.gph files. I found a Bayesian score of -132.57689402451837, which was the score I should have found. The Bayesian score computation I implemented is therefore correct. This allows to search for graphs with good Bayesian scores for the three datasets.

Using the K2 algorithm with 100 iterations, I found a graph for the small and medium datasets. Plots of the graphs can be found in section 2.

- Small dataset:
  - Bayesian score: -3802.8982356708884
  - Computation time (K2, 100 iterations): 44.908034801483154 seconds
- Medium dataset:
  - Bayesian score: -41996.388111958455
  - Computation time (K2, 100 iterations): 222.12284564971925 seconds

As the K2 algorithm was too slow to run on the large dataset, I used a local search approach. This method may find a graph that is less optimal than the K2 algorithm, but the graph will still have a Bayesian sacore that is sufficiently good, and the computation time is greatly reduced. The plot of the graph for the large dataset can be found in section 2. I kept the number of iterations to 100.

- Large dataset:
  - Bayesian score: -479270.0097704328
  - Computation time (K2, 100 iterations): 51.53124856948853 seconds

2. Graphs

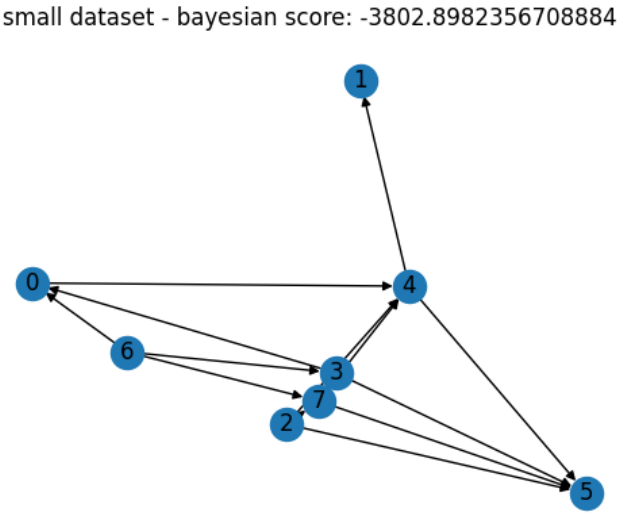

Figure 1: Small dataset - Bayesian score: -3802.8982356708884

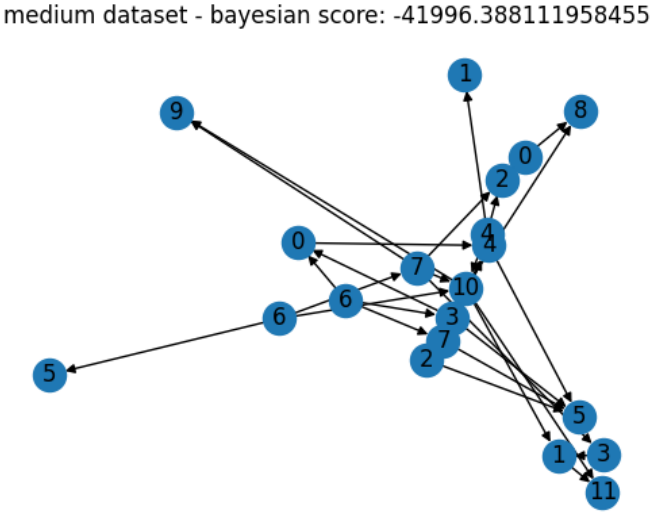

Figure 2: Medium dataset - Bayesian score: -41996.388111958455

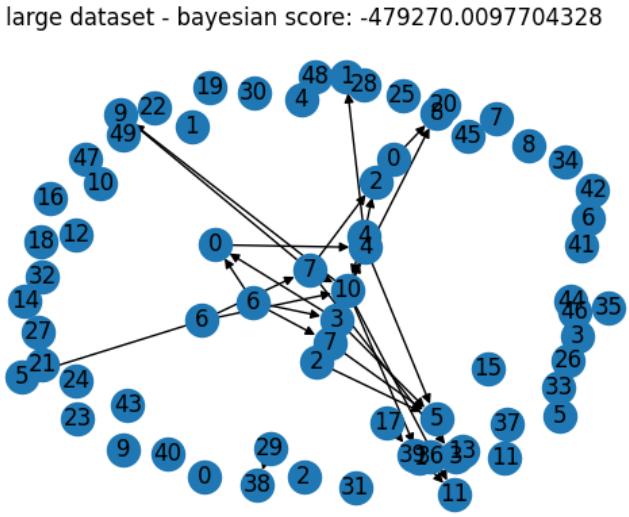

Figure 3: Large dataset - Bayesian score: -479270.0097704328

3. Code

```
import pandas as pd
import networkx as nx
import sys
import time
import numpy as np
from scipy.special import gammaln
import random
import matplotlib.pyplot as plt


###### Utils

class Variable:
    def __init__(self, name, r):
        self.name = name
        self.r = r


def load_data(filename):
    df = pd.read_csv(f"./data/{filename}.csv", delimiter=',')
    df_max = df.max()
    var_names = list(df.columns)
    df = df.groupby(var_names).size().reset_index(name='count')
```

```
    vars = [Variable(var_names[i], df_max.iloc[i]) for i in range(len(
    var_names))]
    return df, vars


def load_graph(filename, vars):
    G = nx.DiGraph()
    G.add_nodes_from(list(range(len(vars))))
    names2idx = {vars[i].name: i for i in range(len(vars))}
    with open(f"./graphs/{filename}.gph", 'r') as f:
        for line in f:
            edge = line.replace('\n', '').replace(' ', '').split(',')
            G.add_edge(names2idx[edge[0]], names2idx[edge[1]])
    return G

def write_graph(dag, score, idx2names, filename):
    plt.title(f"{filename} dataset - bayesian score: {score}")
    nx.draw(dag, with_labels = True)
    plt.savefig(f"./output/{filename}.png", format="PNG")
    with open(f"./output/{filename}.gph", 'w') as f:
        for edge in dag.edges():
            f.write(f"{idx2names[edge[0]]}, {idx2names[edge[1]]}\n")


###### Bayesian score

def sub2ind(siz, x):
    k = np.concatenate(([1], np.cumprod(siz[:-1])))
    return int(np.dot(k, np.array(x) - 1)) + 1


def statistics(vars, G, D):
    n = len(vars)
    r = [var.r for var in vars]

    # Determine number of possible parent configurations for each variable
    q = [np.prod([r[j] for j in G.predecessors(i)]) for i in range(n)]

    # Create empty matrices
    M = [np.zeros((int(q[i]), int(r[i]))) for i in range(n)]

    for var_index in range(n):
        parents = list(G.predecessors(var_index))
        columns_of_interest = [vars[i].name for i in [var_index] + parents]

        grouped_data = D.groupby(columns_of_interest)["count"].sum().
    reset_index()

        for _, row in grouped_data.iterrows():
            k = int(row[vars[var_index].name]) - 1
```

```
            j = 0
            if parents:
                parent_values = [int(row[vars[p].name]) for p in parents]
                j = sub2ind([r[p] for p in parents], parent_values) - 1
            M[var_index][j, k] += row['count']

    return M


def prior(vars, G):
    n = len(vars)
    r = [vars[i].r for i in range(n)]
    q = [np.prod([r[j] for j in list(G.predecessors(i))]) for i in range(n)]
    return [np.ones((int(q[i]), int(r[i])), dtype=int) for i in range(n)]


def bayesian_score_component(M, alpha):
    p = np.sum(gammaln(alpha + M))
    p -= np.sum(gammaln(alpha))
    p += np.sum(gammaln(np.sum(alpha, axis=1)))
    p -= np.sum(gammaln(np.sum(alpha, axis=1) + np.sum(M, axis=1)))
    return p


def bayesian_score(vars, G, D):
    n = len(vars)
    M = statistics(vars, G, D)
    alpha = prior(vars, G)
    return sum(bayesian_score_component(M[i], alpha[i]) for i in range(n))


###### Test bayesian score

def test_score():
    filename = "example"
    # Load data
    df, vars = load_data(filename)
    G = load_graph(filename, vars)
    # Score graph
    expected_score = -132.57689402451837
    score = bayesian_score(vars, G, df)
    score_accuracy = 100 * (1 - (score - expected_score))
    print("Score accuracy: {:.0f}%".format(score_accuracy))


###### K2 algorithm

def best_BN_k2(vars, D, iterations=100):
    var_list = [i for i in range(len(vars))]
    best_score = float('-inf')
```

```
    for k in range(iterations):
        perm = random.sample(var_list, len(var_list))
        G, score = k2_algorithm(vars, D, perm)
        if score > best_score:
            best_score = score
            G_final = G
        print(f"Iteration {k}/{iterations} - score: {score} - best_score: {
    best_score}")

    return G_final, best_score



def k2_algorithm(vars, D, ordering):
    G = nx.DiGraph()
    G.add_nodes_from(range(len(vars)))
    for k, i in enumerate(ordering[1:],1):
        y = bayesian_score(vars, G, D)
        while True:
            y_best, j_best = float('-inf'), 0
            for j in ordering[:k]:
                if not G.has_edge(j, i):
                    G.add_edge(j, i)
                    y_prime = bayesian_score(vars, G, D)
                    if y_prime > y_best:
                        y_best, j_best = y_prime, j
                    G.remove_edge(j, i)
            if y_best > y:
                y = y_best
                G.add_edge(j_best, i)
            else:
                break
    return G, y_best


###### Local search

def best_BN_local_search(vars, D, k_max, iterations):
    n = len(vars)
    best_score = -float('inf')

    for i in range(iterations):
        initial_graph = random_graph(n)
        G, score = local_search_algorithm(vars, D, initial_graph, k_max)
        if score > best_score:
            best_score = score
            G_final = G
        print(f"Iteration {i}/{iterations} - score: {score} - best_score: {
    best_score}")
```

```
    return G_final, best_score


def local_search_algorithm(vars, D, G, k_max):
    y = bayesian_score(vars, G, D)
    for k in range(1, k_max + 1):
        G_prime = random_network_neighbour(G)
        if not nx.is_directed_acyclic_graph(G_prime):
            y_prime = -float('inf')
        else:
            y_prime = bayesian_score(vars, G_prime, D)
        if y_prime > y:
            y, G = y_prime, G_prime
    return G, y


def random_network_neighbour(G):
    n = G.number_of_nodes()
    i = random.randint(0, n-1)
    j = (i + random.randint(2, n - 1)) % n
    G_prime = G.copy()
    if G.has_edge(i, j):
        G_prime.remove_edge(i, j)
    else:
        G_prime.add_edge(i, j)
    return G_prime


def random_graph(n):
    p = 0.10
    G_random = nx.DiGraph()
    for i in range(n):
        G_random.add_node(i)
    for i in range(n):
        for j in range(n):
            if i != j and random.random() < p:
                G_random.add_edge(i, j)
                if nx.is_directed_acyclic_graph(G_random):
                    G_random.remove_edge(i, j)

    return G_random


def main():

    # Test Bayesian score
    test_score()

    small_df, small_vars = load_data("small")
```

```
    medium_df, medium_vars = load_data("medium")
    large_df, large_vars = load_data("large")

    small_vars_names = list(small_df.columns)
    medium_vars_names = list(medium_df.columns)
    large_vars_names = list(large_df.columns)

    iterations = 100

    # Small dataset
    small_idx2names = {i: small_vars_names[i] for i in range(len(
    small_vars_names))}
    start_time = time.time()
    small_best_BN, small_best_score = best_BN_k2(small_vars, small_df,
    iterations)
    end_time = time.time()
    print(f"Time spent per iteration: {(end_time - start_time)/iterations}
    seconds")
    print("Best Bayesian Score:", small_best_score)
    write_graph(small_best_BN, small_best_score, small_idx2names, "small")

    # Medium dataset
    medium_idx2names = {i: medium_vars_names[i] for i in range(len(
    medium_vars_names))}
    start_time = time.time()
    medium_best_BN, medium_best_score = best_BN_k2(medium_vars, medium_df,
    iterations)
    end_time = time.time()
    print(f"Time spent per iteration: {(end_time - start_time)/iterations}
    seconds")
    print("Best Bayesian Score:", medium_best_score)
    write_graph(medium_best_BN, medium_best_score, medium_idx2names, "medium"
    )

    # Large dataset
    large_idx2names = {i: large_vars_names[i] for i in range(len(
    large_vars_names))}
    start_time = time.time()
    large_best_BN, large_best_score = best_BN_local_search(large_vars,
    large_df, 20, iterations)
    end_time = time.time()
    print(f"Time spent per iteration: {(end_time - start_time)/iterations}
    seconds")
    print("Best Bayesian Score:", large_best_score)
    write_graph(large_best_BN, large_best_score, large_idx2names, "large")


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