Brandon Franz

Project 1 README

The K2 search algorithm was implemented to find a graph with the Bayesian score as its metric. The algorithm was chosen because it is greedy and prioritizes higher scores. The ordering of the variables was randomized before starting the graph search. The algorithm was augmented to exclude acyclic graphs and to periodically output the graph for large datasets with long runtimes.

A graph was found with the small dataset in approximately 8 seconds with a score of approximately -3,867. The small graph can be seen in Figure 1. A graph was found with the medium dataset in approximately 321 seconds with a score of approximately -42,810. The medium graph can be seen in Figure 2. A graph was not found with the large dataset. However, after 10 minutes a graph was output with a score of approximately -497,200. The large graph can be seen in Figure 3.

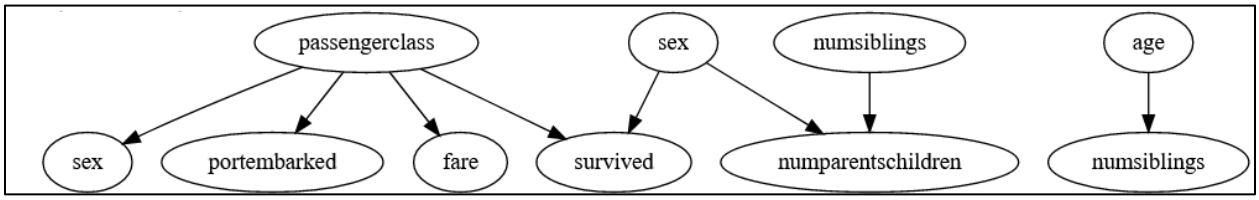

Figure 1: Small Graph

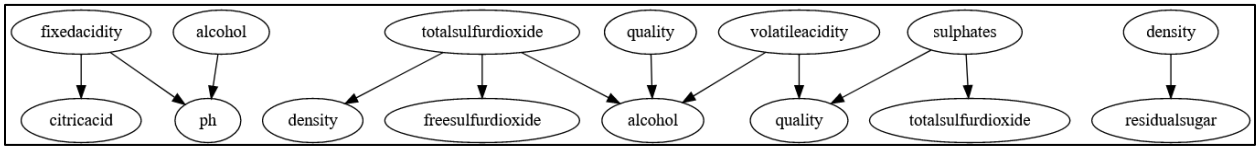

Figure 2: Medium Graph

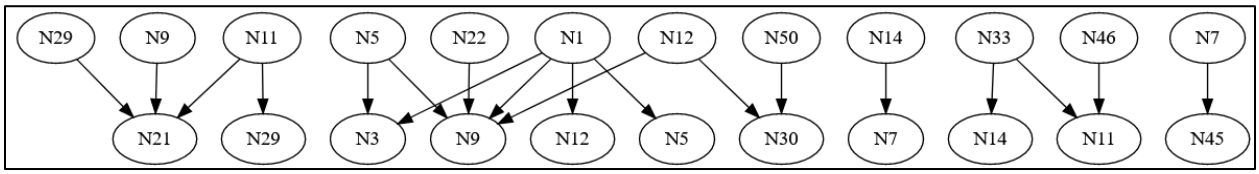

Figure 3: Large Graph

Project 1 Code

```
import numpy as np
from scipy.special import loggamma
import pandas as pd
import os
import itertools
import networkx as nx
import matplotlib.pyplot as plt
import pydot
import time

class BayesianStructureLearning:

    def __init__(self, inputCSV, outputGraph):
        self.inputCSV = os.path.abspath(inputCSV)
        self.outputGraph = outputGraph

    def importCSV(self):
        contents = pd.read_csv(self.inputCSV)
        return contents.columns, contents.to_numpy()

    def buildNoEdgeGraph(self, nodeNames):
        graph = nx.DiGraph()
        graph.add_nodes_from(range(len(nodeNames)))
        return graph

    def buildGraph(self, nodeNames, gph):
        with open(gph, 'r') as f:
            data = f.read()
        data = data.strip().split('\n')
        data = [data[i].strip().split(',') for i in range(len(data))]
        graph = self.buildNoEdgeGraph(nodeNames)
        graph.add_edges_from(data)
        return graph

    def isAcyclic(self, graph):
        return nx.is_directed_acyclic_graph(graph)

    def graphSearch(self, nodeNames, data, method):
        if method == 'K2':
            start = time.time()
            graph = self.buildNoEdgeGraph(nodeNames)
            orderedVariables = list(graph.nodes)
            np.random.shuffle(orderedVariables)
            for (k, i) in list(enumerate(orderedVariables[1:])):
                score = self.scoreGraph(data, graph)
                while True:
                    score_best, j_best = -1000000, 0
                    for j in orderedVariables[0:k]:
                        if not(graph.has_edge(j, i)):
                            graph.add_edge(j, i)
                            score_new = self.scoreGraph(data, graph)
                            if score_new > score_best:
                                score_best, j_best = score_new, j
                            graph.remove_edge(j, i)
                    if score_best > score and self.isAcyclic(graph):
                        score = score_best
                        graph.add_edge(j_best, i)
                        end = time.time()
                        if end-start > 600:
                            self.writeFile_gph(nodeNames, graph)
                            print("Graph Overwritten. Still Solving. Score:")
                            print score
                            start = time.time()
                            nodeMapping = {list(graph.nodes())[i]: nodeNames[i] for i in
range(len(nodeNames))}
                            graph = nx.relabel_nodes(graph, nodeMapping)
                    else:
                        break
            nodeMapping = {list(graph.nodes())[i]: nodeNames[i] for i in range(len(nodeNames))}
```

```
            graph = nx.relabel_nodes(graph, nodeMapping)
            return graph, score

    def indexParentalInstantiation(self, numValues, varParents, parentSample):
        index = 0
        varParentValues = [range(1, numValues[parent]+1) for parent in varParents]
        instants = list(itertools.product(*varParentValues))
        for currentInstant in instants:
            if np.all(parentSample == currentInstant):
                return index
            index+=1

    def graphData(self, data, graph):
        variables = list(graph.nodes())
        parents = [list(graph.predecessors(var)) for var in variables]
        numValues = np.amax(data, axis=0)
        numParentalInstants = np.array([np.prod([numValues[parent] for parent in parents[var]])
for var in variables])
        m = [np.zeros((int(numParentalInstants[var]), int(numValues[var]))) for var in variables]
        for sample in data:
            for var in variables:
                value = sample[var]-1
                instantiation = 0
                if len(parents[var]) != 0:
                    instantiation = self.indexParentalInstantiation(numValues, parents[var],
sample[parents[var]])
                m[var][instantiation, value] += 1

        return m

    def scoreGraph(self, data, graph):
        m = self.graphData(data, graph)
        variables = list(graph.nodes())
        numParentalInstants = [len(m[i][:,0]) for i in range(len(variables))]
        score = 0
        alpha = [np.ones_like(m[var]) for var in variables]
        for var in variables:
            for instant in range(numParentalInstants[var]):
                p = np.sum(loggamma(alpha[var][instant,:]+m[var][instant,:]))
                p -= np.sum(loggamma(alpha[var][instant,:]))
                p+= np.sum(loggamma(np.sum(alpha[var][instant,:])))
                p-= np.sum(loggamma(np.sum(alpha[var][instant,:]) + np.sum(m[var][instant,:])))
                score += p

        return score

    def writeFile_gph(self, nodeNames, graph):
        with open(self.outputGraph, 'w') as f:
            for edge in graph.edges():
                f.write("{}, {}\n".format(edge[0], edge[1]))
        f.close()
        print("Write Graph Complete\n")

    def exportGraph(self, graph, path):
        nx.nx_pydot.write_dot(graph, path)

    def solve(self):
        while True:
            variableNames, data = self.importCSV()
            graph, score = self.graphSearch(variableNames, data, 'K2')
            self.writeFile_gph(variableNames, graph)
            print("Solve Complete with Score:")
            print score
            break

    def solve_timed(self):
        start = time.time()
        self.solve()
        end = time.time()
        print("\nRuntime (s):")
        print end-start
```