Likelihood of a tree GM

Question 2.3.10

The algorithm outlined hereby estimates the likelihood, of a specific beta sequence given tree topology and conditional probability distributions (CPD) of the tree node values. Beta is a list of assignments to leaf nodes in a probabilistic binary tree structured graphical model. Therefore, probability $p(\beta|\boldsymbol{T}, \boldsymbol{\Theta})$ is a probability of observations below root node u conditioned by the assignments of the root node.

$$p(\beta|\boldsymbol{T}, \boldsymbol{\Theta}) = p(X_{o\cap u\downarrow}|X_u = i) \tag{1}$$

For simplicity, we use the following notation

$$p(X_{o\cap u\downarrow}|X_u = i) = s(u, i) \tag{2}$$

The graphical model's structure is binary. Thus, we can imply that inner node u has two children v and w. Observations below v and observations below w are independent. We can break down the equation into

$$p(X_{o\cap u\downarrow}|X_u = i) = p(X_{o\cap v\downarrow}|X_u = i) \times p(X_{o\cap w\downarrow}|X_u = i) \tag{3}$$

Elaborate the expression for one of the child nodes

$$\begin{aligned} p(X_{o\cap v\downarrow}|X_u = i) &= \Sigma_j p(X_{o\cap v\downarrow}, X_v = j|X_u = i) && (4)\\ &= \Sigma_j p(X_{o\cap v\downarrow}|X_v = j) p(X_v = j|X_u = i) && (5)\\ &= \Sigma_j s(v, j) p(X_v = j|X_u = i) && (6) \end{aligned}$$

where s is an analogous sub-problem concerned with vertices that are induced with smaller-rooted subtrees. The resulting expression is recursive and converges to leaf nodes where

$$s(V_{leaf}, i) = \begin{cases} 1, & \text{if } V_{leaf} = i \\ 0, & \text{if otherwise} \end{cases}$$

The conditional probabilities in (7) between parent-child node assignments are retrieved from the vector of CPD. Similarly we evaluate the probability for a second child w and combine the results into the product in (3)

$$p(X_{o\cap w\downarrow}|X_u = i) = \Sigma_j s(w, j) p(X_w = j|X_u = i) \tag{7}$$

The Python implementation of the DGM tree marginalization is shown in Appendix (2.3)

Question 2.3.11

If applied to a small-sized tree graphical model, the outputs for likelihood of samples have order of magnitude from e-02 to e-03.

```
  Sample: 0
  Likelihood: 0.008753221441670067
  Sample: 1
  Likelihood: 0.0383969250979291
  Sample: 2
  Likelihood: 0.009129106859990061
  Sample: 3
  Likelihood: 0.0214406975419561
  Sample: 4
  Likelihood: 0.011945567814215127
```

For a medium size tree the order of magnitude for likelihoods dramatically reduces and now is in the range from e-17 to e-19 meaning the probability of a single certain assignment of nodes is very low.

```
Calculating the likelihood...
Sample: 0
  Likelihood: 8.66416414170832e-17
  Sample: 1
  Likelihood: 5.394284454090606e-18
  Sample: 2
  Likelihood: 8.892415333536359e-18
  Sample: 3
  Likelihood: 1.1222302136292958e-18
  Sample: 4
  Likelihood: 7.58934157249135e-19
```

Large-sized tree samples of leaf nodes output very low likelihood value with the order of magnitude in range e-63 to e-69. We can observe that the likelihood decreases with respect to number of nodes in exponential fashion.

```
  Sample: 0
  Likelihood: 1.2296785012112113e-65
  Sample: 1
  Likelihood: 1.4347770777980813e-63
  Sample: 2
  Likelihood: 3.095491016149858e-66
  Sample: 3
  Likelihood: 3.4231977224272667e-69
  Sample: 4
  Likelihood: 4.822393947666207e-67
```

Appendix

Question 2.3 Implementation

```
import numpy as np
from Tree import Tree
from Tree import Node
from collections import defaultdict

# if the parent value of a node checked in topology equals given
   node, then the checked node is a child of given node
def find_children(p, topology):
   children = []
   for index, parent in enumerate(topology):
      if parent == p:
         children.append(index)
   return children

def calculate_likelihood(tree_topology, theta, beta):
   # number of categories of assignments
   cat = len(theta[0])
   # calculate s for the node
   def calculate_s(node):
      # nodes have 2 children: left and right
      children = find_children(node, tree_topology)
      # identify leaves and reveal assignment likelihood (observed
         => 1)
      if(len(children)<1):
         likelihood=np.zeros(cat)
         likelihood[int(beta[node])]=1
         return likelihood
      s_left=calculate_s(children[0])
      s_right=calculate_s(children[1])
      left_likelihood=np.zeros(cat)
      for i in range(cat):
         left_likelihood[i]=np.dot(theta[children[0]][i],s_left)
      right_likelihood=np.zeros(cat)
      for i in range(cat):
         right_likelihood[i]=np.dot(theta[children[1]][i],s_right)
      return left_likelihood*right_likelihood
   likelihood=np.dot(calculate_s(0), theta[0])
   return likelihood


def main():
```

```
    print("Hello World!")
    print("This file is the solution template for question 2.3.")

    print("\n1. Load tree data from file and print it\n")
    filename = "data/q2_3_medium_tree.pkl" #
       "data/q2_3_medium_tree.pkl", "data/q2_3_large_tree.pkl"
    t = Tree()
    t.load_tree(filename)
    t.print()

    print("\n2. Calculate likelihood of each FILTERED sample\n")
    # These filtered samples already available in the tree object.
    # Alternatively, if you want, you can load them from
       corresponding .txt or .npy files

    for sample_idx in range(t.num_samples):
        beta = t.filtered_samples[sample_idx]
        #print("\n\tSample: ", sample_idx, "\tBeta: ", beta)
        print("\tSample: ", sample_idx)
        sample_likelihood =
           calculate_likelihood(t.get_topology_array(),
           t.get_theta_array(), beta)
        print("\tLikelihood: ", sample_likelihood)


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