Final Project: Coconut Oil Price Movements

Bookmark this page

Final Project: Coconut Oil Price Movements

100/100 points (graded)

You really want coconut oil, but you don't know which of your four nearby markets to buy the coconut oil at. You decide to do a detailed study on how these four markets relate, beginning with how coconut oil prices change across these markets. Download `finalproj.zip`. After unzipping, you should see the following:

Code:

`final_proj.py` — your code goes here

Data:

`coconut.csv` — coconut oil dataset

The data provided consists of 2.8 years of daily coconut oil price movement data quantized to +1 (price went up in the past day), 0 (price stayed exactly the same in the past day), and -1 (price went down in the past day). Each line of `coconut.csv` looks something like the following:

```
10/9/2012,-1,1,0,1
```

Note that there are five items: the date (month/day/year) followed by price movements for markets 0, 1, 2, and 3.

Let's suppose for the purposes of this project that the price movements on different days are reasonably modeled to be i.i.d.

- **(a)** Code the Chow-Liu algorithm for learning a tree for these four markets. (Your code should of course work if there are instead some arbitrary number of markets.) To do so, first fill in the functions `compute_empirical_distribution` and `compute_empirical_mutual_info_nats`, and then proceed to fill in `chow_liu`. (The expected inputs and outputs are provided in comments for the functions.)

  To help you out with the function `chow_liu`, we have provided code for what's called a Union-Find data structure (in the Python class `UnionFind` – do not modify this class!), and we have initialized an instance of the class for you as well within `chow_liu` (if you look inside this function, you'll see that there is a variable `union_find` that is initialized for you).

  To check whether two nodes `i` and `j` are already connected, note that the test `union_find.find(i) == union_find.find(j)` returns `True` if the two nodes are already connected and `False` otherwise. If you want to add the edge for nodes `i` and `j`, then be sure to tell the Union-Find data structure that you're adding this edge, for which you do `union_find.union(i, j)`. Those are the only two ways in which you will interact with the variable `union_find`.

- **(b)** Once you learn the Chow-Liu tree, of course the next thing is learning parameters for this specific tree. Fill in the functions `compute_empirical_conditional_distribution` and `learn_tree_parameters`. (The expected inputs and outputs are provided in comments for the functions.)

  Important: Please read the documentation for the function `convert_tree_as_set_to_adjacencies`; how this function is used is already provided for you in `learn_tree_parameters` but it'll be helpful for you to know how to work with edges defined as a dictionary (namely `edges[i]` is a list that says which nodes are neighbors of node `i`).

  Hint: There are different ways you could traverse the tree in code. To help you out, we provide a snippet of code for one way to traverse the tree:

```
fringe = [root_node]  # this is a list of nodes queued up to be visited next
visited = {node: False for node in nodes}  # track which nodes are visited
while len(fringe) > 0:
    node = fringe.pop(0)  # removes the 0th element of `fringe` and returns it
    visited[node] = True  # mark `node` as visited
    for neighbor in edges[node]:
        if not visited[neighbor]:
            # do some processing that involves the edge `(neighbor, node)` here

            # finally after you do your processing, add `neighbor` to `fringe`
            fringe.append(neighbor)
```

- **(c)** After learning both the tree and the parameters, of course this means that we now have a graphical model, for which we can solve specific inference tasks. First, let's not worry about incorporating observations yet. Fill in the function `sum_product` for running the general Sum-Product algorithm for a given tree-structured graphical model. (As before, see the comment at the beginning of the function for what the inputs and output for the function should be.)

  Some important notes: Please use the node potentials' dictionaries' keys to figure out what the alphabets are for each random variable. For the purposes of this final project, it's fine to *not* do the optimization we discussed for speeding up Sum-Product.

- **(d)** We want to be able to answer questions like “what if we see in the last day that the price moved up for markets 1 and 2, but we didn't get to observe what happened with markets 0 and 3"? What can we say about the posterior marginals for markets 0 and 3? This amounts to incorporating the observations that markets 1 and 2 each have observed values of +1. Fill in the general function `compute_marginals_given_observations` that let's us incorporate any subset of observations for the four markets and that also produces posterior marginals. (As before, see the comment at the beginning of the function for what the inputs and output for the function should be.)

THIS IS WHERE YOU PASTE YOUR CODE

```
import copy
import numpy as np
from collections import Counter


# DO NOT MODIFY THIS FUNCTION
def convert_tree_as_set_to_adjacencies(tree):
    """
    This snippet of code converts between two representations we use for
    edges (namely, with Chow-Liu it suffices to just store edges as a set of
    pairs (i, j) with i < j), whereas when we deal with learning tree
    parameters and code Sum-Product it will be convenient to have an
    "adjacency list" representation, where we can query and find out the list
    of neighbors for any node. We store this "adjacency list" as a Python
    dictionary.
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

Hide output

CORRECT

```
Test: compute_empirical_distribution(array([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, -1, 0, 0, 0, 1, -1, 0, -1, 0, 0, 0, -1, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 1,
0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 1, -1, 1, -1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 1,
1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, -1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 1, 0, 1, 1, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, -1,
1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, -1, 0, -1, 0, 0, 0, -1, 0, 0, 1, 1,
0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, -1, 0,
0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 1, 1, 1, 0, 0, 1,
0, 0, 0, 0, 0, -1, 0, 1, -1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, -1, 0, 0, 0, 0,
0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, -1, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, -1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, -1, 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0, 0, 0, 1, 0, -1,
0, 0, 0, 0, 0, 0]))
```

Output:

```
[["-1", 0.07625], ["0", 0.8175], ["1", 0.10625]]
```

Test: compute_empirical_mutual_info_nats(array([

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, -1, 0, 0, 0, -1, -1, -1, 0, 0, 0, 0, -1, -1, 1, 0, 0, -1, 0, 0,
-1, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, -1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, -1, 1, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1,
1, -1, 0, 0, 1, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, -1, -1, -1, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
0, 1, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, -1, 1, 0, 0, -1, 0, 0, 1, 1,
0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -1, 0, -1, -1, -1, -1, 0, 0, 0, 0, 1, 0,
-1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, -1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0, 1, 1, 1, 0, 1, 0, -1, 0, -1, 0, 0, 0, 1, 0, -1, 1, 0, 0, 0, 0, -1, -1, -1, 0, 0,
-1, 0, 1, 0, 0, 1, -1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, -1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
1, 0, 1, 1, 0, 1, 1, 0, 0, -1, -1, 0, -1, 0, -1, 0, 1, 0, 0, -1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
```

```
1, 1, 1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, -1, 0, -1, 0, 0, -1, -1, -1, -1, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 1, 1, 1, -1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, -1, 0, -1, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, -1, 0, -1, 0, 0, 0, -1, 0, -1, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, -1, -1, -1, -1, -1, 0, 0, 1, 1, 0, 0, 0, 1, 0, -1, -1, 1, 0, -1, 1, 0, 1, 0,
1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, -1, -1, 0, 0, 0, -1, -1, -1, -1, -1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 1, 1, 1, 0, -1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, 0, -1, -1, 0, 0, 0, 0, 0, 0,
-1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, -1, -1, 0, 0, 1, 0, 1, 1, 1, 1]), array([ 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, -1, 0,
-1, 0, -1, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 1, 0, 0, -1, -1,
0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, -1, 1, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, -1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, -1, 1,
0, 0, 0, 0, 0, -1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, -1, 0, -1, -1, -1,
-1, 0, -1, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, -1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
-1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, -1, 0, 0, -1, -1, 0, -1, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, -1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, 0, 0, -1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, -1, -1, -1, -1, -1, 0, 0, -1, -1, -1, -1, -1, -1,
0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, -1, -1, -1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]))
```

Output:

```
0.0528594532718
```

```
Test: chow_liu(array([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], ..., [1, 0, 0, 1], [1, 0, 1,
0], [1, 0, 0, 1]]))
```

Output:

```
[[0, 1], [0, 2], [0, 3]]
```

```
Test: compute_empirical_conditional_distribution(array([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 1, -1, 0, -1, 0, 0, 0, -1, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0,
0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, -1, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 1, -1, 1, -1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0,
1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, -1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
1, 1, 0, -1, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, -1, 0, -1, 0, 0, 0, -1,
0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 1,
1, 1, 0, 0, 1, 0, 0, 0, 0, 0, -1, 0, 1, -1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, -1, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0,
0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, -1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0,
0, 0, 1, 0, -1, 0, 0, 0, 0, 0, 0]), array([ 0, 0, 1, 0, 0, 0, 0, -1, -1, 0, 0, 0, 0, 1, 0,
-1, -1, 1, -1, 0, -1, 0, 0, -1, 0, 1, 0, -1, -1, 0, -1, 1, -1, 0, -1, -1, 0, 0, 0, 0, 0, -1,
0, -1, -1, 0, 1, 0, 1, 1, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, -1, -1, 0, -1, -1, 1, 0, 1,
-1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, -1, -1, 0, 0, 1, 0, 0, -1, -1, 0, 1, 0, 0, -1,
```

```
-1, 0, 0, 1, -1, 0, -1, 0, 0, -1, -1, -1, 0, -1, 1, 1, 0, -1, 0, 0, -1, -1, -1, 0, -1, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 1, 0, -1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, -1, -1,
-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, -1, 0, 0, -1, -1, 0, 0, 1, -1, 0, -1, -1, -1, 1, 1, -1, 0, 0,
1, -1, 1, 1, 1, 1, -1, 0, 1, 1, -1, -1, 0, 0, -1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 1, 0, -1, 1, 0, 1, 1, 0, 0, 0, -1, 0, 1, -1, 0, 0, 0, 0, 1, -1, -1, -1, 0, 0, 0, 1,
0, 0, -1, 0, 0, 1, 1, 0, 1, 1, 0, -1, -1, 0, 1, 1, 1, 0, 1, -1, 0, 1, 0, -1, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, -1, -1, 0, 0, 0, 1, -1, -1, 0, 1, 0, 0, 0, -1, -1, 1, 0, 0, 0, 1, -1, 0,
1, 1, 0, 1, -1, 0, 1, 0, 1, 0, 1, 0, -1, 0, 1, 0, 0, -1, -1, 0, -1, 0, 1, 0, 1, 1, 0, -1,
-1, -1, 0, -1, -1, -1, 0, 1, 0, 0, 1, 1, 0, -1, -1, 1, 0, 0, 0, 0, 1, 1, 1, 0, -1, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, -1, -1, 1, 1,
0, -1, -1, -1, 0, 1, 0, 0, 0, -1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, -1, -1, 0, -1, 0, 0,
0, 1, 1, 1, 1, 0, 1, 0, -1, -1, 0, 1, -1, -1, 0, -1, 1, 1, -1, 1, 1, 0, 1, 1, -1, 1, -1, 1,
0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, -1, -1, 0, 0, 1, 1, 1, 0, 0, 0, 1, -1, 1, 0, -1, -1, 0,
-1, -1, -1, 0, 0, -1, 0, 0, 1, -1, 1, 1, -1, 0, -1, -1, 0, 1, 1, 0, 0, -1, 1, 1, 0, -1, -1,
0, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, 0, 1, 1, 1, 1, -1, 0, 1, -1, 1, 1, -1, 0, 1, 1, 0,
1, 1, -1, -1, -1, 0, 1, -1, 1, -1, 0, 0, 0, -1, -1, 0, -1, -1, 1, 1, 0, -1, 0, 0, 1, 1, 1,
-1, 1, 0, 1, 1, -1, 0, 1, 0, 0, -1, 0, 0, 1, 1, 1, 0, 1, 1, 1, -1, 0, 1, 0, 1, 1, 0, 0, -1,
1, -1, 1, 1, -1, -1, -1, -1, 0, 0, 0, -1, -1, 0, 1, 0, 1, -1, -1, -1, -1, 1, 0, 1, 1, -1, 1,
0, 0, 0, 0, 0, -1, -1, 1, 1, 0, -1, 0, -1, 0, 1, -1, 0, -1, -1, -1, 0, 0, 1, 0, 0, 1, 1, 1,
-1, 0, 0, -1, -1, -1, -1, 0, 1, 0, 1, 0, 0, 0, 1, -1, 0, -1, -1, -1, 1, 1, 0, 0, 0, 0, 1, 0,
-1, -1, 0, -1, 1, 0, 0, 0, -1, 0, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, 1, 1, 0, 1, 0, 0,
-1, -1, 1, 0, 1, 1, 1, -1, -1, 1, 0, 1, 1, 1, -1, 0, 0, 0, 1, 0, 1, 0, 1, -1, 0, -1, -1, 0,
0, 1, 1, 0, 1, 1, 0, -1, -1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, -1, -1, 0, 0, 0, 0, 1, 0,
-1, 0, 0, 1, -1, 0, -1, 0, -1, 0, -1, -1, -1, 1, 1, 0, 0, -1, -1, -1, -1, -1, -1, 0, 0, 0,
-1, 1, 1, 1, 0, 0, -1, 0, 0, 1, 1, 0, 1, -1, 1, 1, 1, 0, 0, 0, -1, -1, -1, -1, 1, 0, 1, 0,
1]))
```

Output:

```
[["-1", [["-1", 0.1031390134529148], ["0", 0.7937219730941704], ["1",
0.1031390134529148]]], ["0", [["-1", 0.05830903790087463], ["0",
0.8658892128279884], ["1", 0.07580174927113703]]], ["1", [["-1",
0.07692307692307693], ["0", 0.7692307692307693], ["1",
0.15384615384615385]]]]
```

Test: learn_tree_parameters(array([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], ..., [1, 0, 0, 1], [1, 0, 1, 0], [1, 0, 0, 1]]), {(0, 1), (0, 3), (0, 2)}, 0)

Output:


```
[[["0", [["-1", 0.14], ["0", 0.69], ["1", 0.17]]], ["1", [["-1", 1], ["0",
1], ["1", 1]]], ["2", [["-1", 1], ["0", 1], ["1", 1]]], ["3", [["-1", 1],
["0", 1], ["1", 1]]]], [["(0, 1)", [["-1", [["-1", 0.16071428571428573],
["0", 0.7946428571428571], ["1", 0.044642857142857144]]], ["0", [["-1",
0.06521739130434782], ["0", 0.8641304347826086], ["1",
0.07065217391304347]]], ["1", [["-1", 0.05147058823529415], ["0",
0.6470588235294118], ["1", 0.3014705882352941]]]]], ["(0, 2)", [["-1",
[["-1", 0.2767857142857143], ["0", 0.6785714285714286], ["1",
0.044642857142857144]]], ["0", [["-1", 0.06521739130434782], ["0",
0.8369565217391305], ["1", 0.09782608695652174]]], ["1", [["-1",
0.04411764705882353], ["0", 0.6323529411764706], ["1",
0.3235294117647059]]]]], ["(0, 3)", [["-1", [["-1", 0.6517857142857143],
["0", 0.26785714285714285], ["1", 0.08035714285714286]]], ["0", [["-1",
0.2318840579710145], ["0", 0.4963768115942029], ["1", 0.2717391304347826]]],
["1", [["-1", 0.16176470588235295], ["0", 0.2867647058823529], ["1",
0.5514705882352942]]]]], [["(1, 0)", [["-1", [["-1", 0.16071428571428573],
["0", 0.06521739130434782], ["1", 0.05147058823529415]]], ["0", [["-1",
0.7946428571428571], ["0", 0.8641304347826086], ["1", 0.6470588235294118]]],
["1", [["-1", 0.044642857142857144], ["0", 0.07065217391304347], ["1",
0.3014705882352941]]]]], ["(2, 0)", [["-1", [["-1", 0.2767857142857143],
["0", 0.06521739130434782], ["1", 0.04411764705882353]]], ["0", [["-1",
0.6785714285714286], ["0", 0.8369565217391305], ["1", 0.6323529411764706]]],
["1", [["-1", 0.044642857142857144], ["0", 0.09782608695652174], ["1",
0.3235294117647059]]]]], ["(3, 0)", [["-1", [["-1", 0.6517857142857143],
["0", 0.2318840579710145], ["1", 0.16176470588235295]]], ["0", [["-1",
0.26785714285714285], ["0", 0.4963768115942029], ["1",
0.2867647058823529]]], ["1", [["-1", 0.08035714285714286], ["0",
0.2717391304347826], ["1", 0.5514705882352942]]]]]]
```


Test: sum_product({1, 2, 3, 4, 5}, {1: [2, 3], 2: [1, 4, 5], 3: [1], 4: [2], 5: [2]}, {1: {'blue': 0.3, 'green': 0.9}, 2: {'blue': 0.1, 'green': 0.2}, 3: {'blue': 0.4, 'green': 0.6}, 4: {'blue': 0.1, 'green': 0.5}, 5: {'blue': 0.5, 'green': 0.3}}, {(1, 2): {'blue': {'blue':

```
0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}, (1, 3): {'blue': {'blue': 0, 'green':
10}, 'green': {'blue': 10, 'green': 0}}, (4, 2): {'blue': {'blue': 0.1, 'green': 1},
'green': {'blue': 1, 'green': 0.1}}, (2, 5): {'blue': {'blue': 1, 'green': 3}, 'green':
{'blue': 3, 'green': 1}}, (3, 1): {'blue': {'blue': 0, 'green': 10}, 'green': {'blue': 10,
'green': 0}}, (5, 2): {'blue': {'blue': 1, 'green': 3}, 'green': {'blue': 3, 'green': 1}},
(2, 4): {'blue': {'blue': 0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}, (2, 1):
{'blue': {'blue': 0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}})
```

Output:

```
[["1", [["blue", 0.2847164012293937], ["green", 0.7152835987706063]]], ["2",
[["blue", 0.6982397317686504], ["green", 0.30176026823134955]]], ["3",
[["blue", 0.7152835987706063], ["green", 0.28471640122939373]]], ["4",
[["blue", 0.2148644872869516], ["green", 0.7851355127130484]]], ["5",
[["blue", 0.5008382229673093], ["green", 0.4991617770326907]]]]
```

Test: compute_marginals_given_observations({1, 2, 3, 4, 5}, {1: [2, 3], 2: [1, 4, 5], 3: [1], 4: [2], 5: [2]}, {1: {'blue': 0.3, 'green': 0.9}, 2: {'blue': 0.1, 'green': 0.2}, 3: {'blue': 0.4, 'green': 0.6}, 4: {'blue': 0.1, 'green': 0.5}, 5: {'blue': 0.5, 'green': 0.3}}, {(1, 2): {'blue': {'blue': 0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}, (1, 3): {'blue': {'blue': 0, 'green': 10}, 'green': {'blue': 10, 'green': 0}}, (4, 2): {'blue': {'blue': 0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}, (2, 5): {'blue': {'blue': 1, 'green': 3}, 'green': {'blue': 3, 'green': 1}}, (3, 1): {'blue': {'blue': 0, 'green': 10}, 'green': {'blue': 10, 'green': 0}}, (5, 2): {'blue': {'blue': 1, 'green': 3}, 'green': {'blue': 3, 'green': 1}}, (2, 4): {'blue': {'blue': 0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}, (2, 1): {'blue': {'blue': 0.1, 'green': 1}, 'green': {'blue': 1, 'green': 0.1}}}, {1: 'green', 4: 'blue'})

Output:

```
[["1", [["green", 1.0]]], ["2", [["blue", 0.27999999999999997], ["green",
0.7200000000000001]]], ["3", [["blue", 1.0]]], ["4", [["blue", 1.0]]], ["5",
[["blue", 0.7000000000000001], ["green", 0.3]]]]
```

Hide output

Submit You have used 13 of 1000 attempts

✔ Correct (100/100 points)

Discussion

Topic: Final Project / Final Project: Coconut Oil Price Movements

Show Discussion


Download on the
App Store
ANDROID APP ON
Google play