Bayesian Network for Stem Cell Differentiation
David West, dwest25

For the one-parent node network, the process was relatively simple. I first wrote a function, *count()* that tallies P(N=0) and P(N=1) for a given node and j value (0 or 1 for on or off protein presence). The expected X and j values were taken as parameters along with a list of lines from the provided file. Creating a list of strings allowed more flexibility, but in hindsight, further separating the strings into lists might have been more convenient. The *count()* function looped through the length of the string and incremented accumulator variables *n0j* and *n1j* as necessary. The function returned these tallies as well as the calculated the P(N=0) and P(N=1) values.

The bulk data processing work could be done by the *count()* function. This function, however, was designed to handle only a single X and corresponding j value. *Count()* was separated from the *one_node_network()* function to make the program more modular and thus more flexible (although strategy began to break down later in problem 2). The *one_node_network()* function calls the *count()* function with different parameters to be printed as results. A segmented screenshot of this data are shown below.

```
        X = 1

**** j=0 ****
n0: 211
n1j: 512
P(N=0): 0.292
P(N=1): 0.708

**** j=1 ****
n0: 583
n1j: 230
P(N=0): 0.717
P(N=1): 0.283

log2(P) =  -1373.521677515958
Deviation =  0.002
Percent Deviation =  0.0 %
```

```
        X = 2

**** j=0 ****
n0: 316
n1j: 450
P(N=0): 0.413
P(N=1): 0.587

**** j=1 ****
n0: 478
n1j: 292
P(N=0): 0.621
P(N=1): 0.379

log2(P) =  -1531.099892099619
Deviation =  157.58
Percent Deviation =  11.473 %
```

```
        X = 3

**** j=0 ****
n0: 478
n1j: 314
P(N=0): 0.604
P(N=1): 0.396

**** j=1 ****
n0: 316
n1j: 428
P(N=0): 0.425
P(N=1): 0.575

log2(P) =  -1543.9291565373978
Deviation =  170.409
Percent Deviation =  12.407 %
```

```
        X = 4

**** j=0 ****
n0: 429
n1j: 376
P(N=0): 0.533
P(N=1): 0.467

**** j=1 ****
n0: 365
n1j: 366
P(N=0): 0.499
P(N=1): 0.501

log2(P) =  -1578.2484445170658
Deviation =  204.728
Percent Deviation =  14.905 %
```

```
        X = 5

**** j=0 ****
n0: 403
n1j: 356
P(N=0): 0.531
P(N=1): 0.469

**** j=1 ****
n0: 391
n1j: 386
P(N=0): 0.503
P(N=1): 0.497

log2(P) =  -1578.6424724350709
Deviation =  205.122
Percent Deviation =  14.934 %
```

```
        X = 6

**** j=0 ****
n0: 396
n1j: 346
P(N=0): 0.534
P(N=1): 0.466

**** j=1 ****
n0: 398
n1j: 396
P(N=0): 0.501
P(N=1): 0.499

log2(P) =  -1578.3317632556546
Deviation =  204.812
Percent Deviation =  14.911 %
```

Also in the above results are the $\log_2 P$ values for each single-node network. I wrote a separate function, *compute_log2P(),* that received a list of any number of [n0j, n1j] lists. This is a modified strategy to allow for more general inputs (i.e. a larger number of nodes). The function uses Stirling 's approximation for values of n_{0j} or n_{1j} larger than 50. This function loops through and accumulates these partial sums for the total log2P value that is returned. Included in the printed results from part 1 are deviation from the given value $\log_2 P$=-1373.52. The results show that the highest probability single node is X_1. All other $\log_2 P$ values deviated from this by more than 10%, or nearly -200 in X_4-X_6. Thus, X_1 is 2^{205} times more likely than X_4, the worst one-parent node.

For part 2, I wrote a more intense function that could perform the same tallying tasks as the functions from part 1, but for a multi-node network. In hindsight, the *one_node_network()* combined with *count()* simply performed a special case of *multi_node_network()*. This function counted the instances of a unique node and the n_{0j} and n_{1j} counts for that node. A string of each unique node was stored in a list, and the data for each node type (n_{0j} , n_{1j}, occurences) was stored in a list at a corresponding index. Looking back, alternative methods such as a dictionary might have been more convenient and safer for data. Counting occurrences was not absolutely necessary (and would later be filtered out), but it serves a checkpoint when dealing with the large sets of data in a more code-intensive program.

As mentioned earlier, the *multi_node_network()* function is significantly more flexible than the *one_node_network()*—it takes as a parameter the number of nodes which the network to be tested contains. For the case of problem 2, I took parameter *n* to be 3. The data for this problem (unlike in part 1) was written to a *.dat* file. The data from this file is presented in figure 2. A typed table has been added alongside the raw data for clarity.

```
['1 0 1', 98, 96, 0.505, 0.495]
['0 0 0', 89, 109, 0.449, 0.551]
['1 1 1', 184, 28, 0.868, 0.132]
['1 1 0', 192, 16, 0.923, 0.077]
['0 1 0', 88, 99, 0.471, 0.529]
['0 1 1', 14, 149, 0.086, 0.914]
['1 0 0', 109, 90, 0.548, 0.452]
['0 0 1', 20, 155, 0.114, 0.886]
```

Figure 2: part 2 data

Network	n_{oj}	n_{1j}	P(N=0)	P(N=1)
1 0 1	98	96	0.505	0.495
0 0 0	89	109	0.449	0.551
1 1 1	184	28	0.868	0.132
1 1 0	192	16	0.923	0.077
0 1 0	88	99	0.471	0.529
0 1 1	14	149	0.086	0.914
1 0 0	109	90	0.548	0.452
0 0 1	20	155	0.144	0.886

Table 1: Labeled part 2 data

The *multi_node_network()* function calls the *calculate_log2P()* function sending a list of length 2^n (in this case 8). The value of -1269.16 is ~104 greater than the best value for the one-parent-node network. As stated in the problem description, this would mean that the most likely 3-node network is 2^{104} times greater than the best single-parent-node network.

```
Multi-Node Network
-1269.1600705391697
```

Figure 3: $\log_2 P$ for three parent node network

Highlighted in Table 1 are the two most probable and least probable nodes (green and red respectively). The least probable networks involve the presence of X_1 and X_2, while the two most probable both included X_3. Thus, the rule used to generate the data set can be inferred: low probability in the presence X_1 and X_2 (OTC4 and SOX2), high probability in the presence of X_3 (REX1).

Network	n_{oj}	n_{1j}	P(N=0)	P(N=1)
1 1 1	20	157	0.113	0.887
1 1 0	14	184	0.071	0.929
1 0 1	145	49	0.747	0.253
0 1 0	13	177	0.068	0.932
0 0 0	92	97	0.487	0.513
1 0 0	94	104	0.475	0.525
0 0 1	149	58	0.720	0.280
0 1 1	20	163	0.109	0.891

Table 2: Labeled part 3 data

For part 3, all 4 high-probability networks include X_2 or SOX2. For the two low-probability networks, X_3 is included on both. Of course, X_2 is not present in the low-probability networks, which is to be expected, as 4/8 networks including the protein. X_1, or OTC4, does not seem to have any effect on differentiation. In sum it can be inferred that the presence of SOX2 positively encourages to iPSC differentiation and the presence of REX1 negatively encourages it.

```
['1 1 1', 20, 157, 0.113, 0.887]
['1 1 0', 14, 184, 0.071, 0.929]
['1 0 1', 145, 49, 0.747, 0.253]
['0 1 0', 13, 177, 0.068, 0.932]
['0 0 0', 92, 97, 0.487, 0.513]
['1 0 0', 94, 104, 0.475, 0.525]
['0 0 1', 149, 58, 0.72, 0.28]
['0 1 1', 20, 163, 0.109, 0.891]
```

Figure 5: Part 3 data

If:	P(N=0)	P(N=1)
!SOX4 and REX1	0.9	0.1
SOX4	0.1	0.9
else	0.5	0.5

```
Multi-Node Network
-1180.0968297446984
```

Figure 6: $\log_2 P$ for three-parent node network of file 2