Comp9418 – Report

Deepansh - Z5199370

Daniel Constantinidis - z5162121

Shashank Reddy Boosi – z5222766

Task-1 D-seperation test

Main challenge for this task was to create a d-seperation algorithm that works on directed graphs to find if two nodes X and Y are d-seperated given Z. This task was implemented in 3 steps..

1) Graph refinement, i.e. removal of leaf nodes and unwanted edges.

2) Making a directed graph into undirected graph.

3) finding the path between X and Y.

The functions created for this purpose are explained below.

Function: d_seperation(Graph, X, Y, Z).

Complexity: O(n^2), n is the number of nodes.

- The function takes in 4 values:
  - Graph
  - Start Node – X (A list of nodes)
  - End Node – Y (A list of nodes)
  - Observed Node – Z (A list of nodes)
- The function takes use of a recursive helper function(refine graph) to apply the first step of the algorithm.
- Function refine_graph(G,X,Y,Z,union) returns a concentrated graph without leaf nodes and the edges outgoing from node Z deleted. Union is basically X U Y U Z.
- This concentrated graph is a directed graph, Hence a new undirected graph is created from the graph returned by refine_graph().
- Find_connection(dict,start_node,end_node) recursively finds all the paths from start_node to end_node. It returns a list of all the paths.
- If there is a path present, X and Y are not d-seperated(False). If find_connection() returns an empty list then X and Y are d-seperated(True). Returns Boolean values

Function: refine_graph(Graph, X, Y, Z)

- Recursively finds empty list values of keys in the dictionary to remove and also removes the key from other lists, Hence deleting the existence of the node
- Also removes the Outgoing edges from node Z. i.e. second step of the algorithm.
- Return a dictionary

Function: find_connection(Graph,X,Y)

- The algorithm uses an important technique called backtracking: it tries each possibility in turn until it finds a solution. O(m+n)
- Recursively finds all paths from start node to end node in a undirected graph and returns either the path if it is able to find otherwise empty list.

Reference: https://www.python.org/doc/essays/graphs/

Task-2

Since most of this task had already been implemented in the tutorials, there was not much challenge to this task.

The size of the joint distribution with 16 nodes is **2654208.**

We find this by multiplying the size of the outcome space of each variable by each other. For example outcomeSpace = (A: (0,1), B: (0,1,2), C: (0,1,2,3))

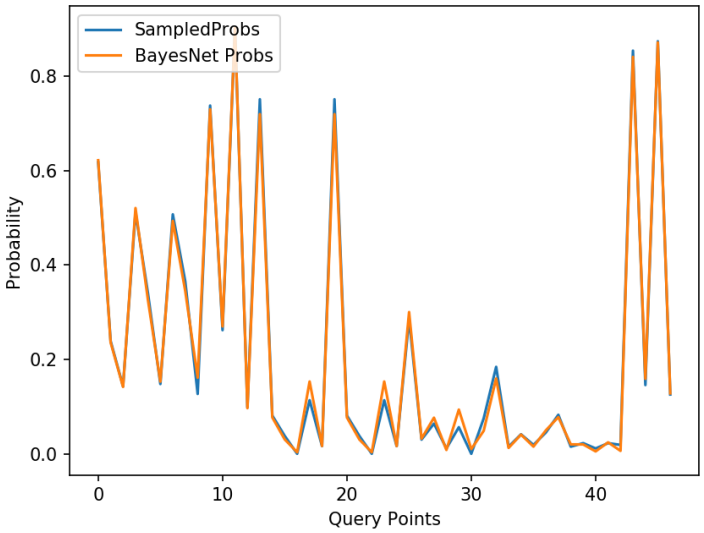
We get 2 * 3 * 4 here, multiplying the size of each variables outcome space.

Task-3

Main **Challenges** for this task included setting up the data structures for getting the sampled probabilities, as well as the Bayes Net queries. It was tricky, since we needed to collect probabilities for each outcome combination and compare each one. This was resolved by using the structure of the conditional probability tables, and accessing each dictionary item as a probability, with the keys being the outcome.

The **time complexity** of the sampling procedure is O(N*M), N being the number of variables, and M being the number of samples generated. We need to access each node once per sample generation, and we are sampling M times from the graph.

The **accuracy** of the sampled estimates are pretty close to that of the Bayesian Network for the given queries. We manually created 10 queries, and compared them at each outcome probability.



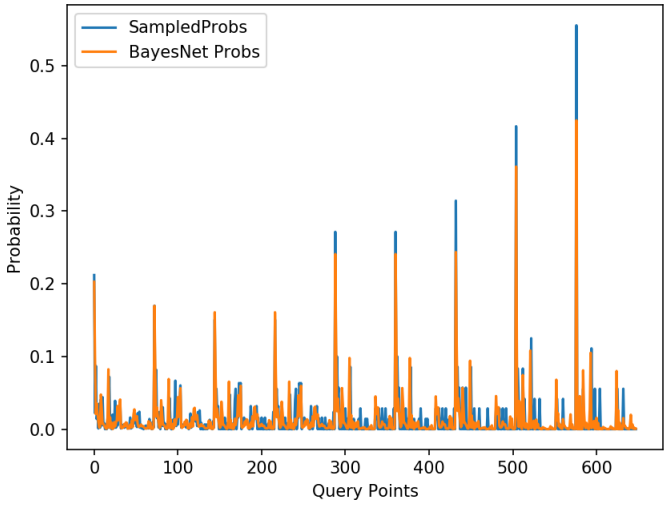
A Root Mean Squared Error metric was used to determine how closely the sampled estimates represent the ground truth probabilities from the Bayes Net.

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

For this experiment the results were;

```
RMSE = 0.02569876727287478
```

We used queries with varying amount of ‘evidence’, in order to get a general estimate of the accuracy of the sampling. As you add more observed variables in the query, the effective sample size reduces. This is because we are discarding samples that disagree with the evidence, and only considering ones where the evidence agrees with the sample. If we add more and more conditions on the variable outcomes, it becomes less likely you will obtain a sample with those observations. Since we are reducing the effective sample size the accuracy of the sampled estimates is reduced, because there are less probability points which we can use in the comparison, resulting in a potential bias from the reduced sample size. This is evident in the following experiment.



`RMSE = 0.023700893359826652`

We created 9 queries, with the same variables, but each with an increasing number of observed variables. The further along we go on the x-axis, the more variables we observe. We can see where each new query begins at each ‘spike’ in the graph. As you increase the evidence, the sampled probabilities disagree more and more with the Bayes Net ground truths.

Task-4

Breast Cancer dataset has been compared between Random Forest which is an ML classifier and the Bayesian Network. For the ML classifier, the features are label encode and normalized and the data is split into 80 and 20 percent respectively and then the random forest with a depth of 5 is applied on the dataset and an accuracy of 89.6 percent and an error of 27.2 percent is achieved.

For Bayesian Network, the dataset is split the same way as for the ML classifier and the labels. The conditional probabilities are extracted for the training data and the classification in done on the feature BC considering that all the other features are observable and an accuracy of 88.8 percent is achieved for Bayesian network. Bayesian network achieves as much as powerful ML algorithms that too with simple computation time.