Mathematical Networks

Curtis Hu

July 2023

1 Shannon Entropy

$$H = -\sum p_i log(p_i)$$

Some conditions:

- H must span for an entire sample space. So $\sum p_i = 1$.
- H is continuous
- As events become more equally likely $\rightarrow$ H becomes more maximized

H then measures a quantity of "information" in an intuitive sense. In layman's terms, as the likelihood of events become more and more equal, we require more "information" to convey information. For example, if 2 events are equally likely to happen, we require 1 bit to convey what actually happened. If only one bit can happen in a sample space, then we require 0 bits.

More more formally, Shannon Entropy is the amount of information needed to express a symbol of information based on the given sample space.

Intuitively, for a coin toss, you'd need 1 bit or 2 states to competeley convey what happened.

$$H_2 = -\sum p_i log_2(p_i)$$

$$1 = -\frac{1}{2} log_2(1/2) - \frac{1}{2} log_2(1/2)$$

For a dice, you'd need 3 bits.

$$-log_2(1/6) = 2.58 \approx 3$$

The maximal amount of entropy is then:$H_{max} = log_2(1/n)$ because when all events are equally likely, there is most uncertainty.

2 Graph Theory

2.1 Storing Graphs:

- Adjacency Matrix (Nodes v. Nodes)
- Distance Matrix (Nodes v Nodes)
- Connectedness Matrix (Nodes v. Nodes)
- Incidence Matrix (Nodes v. Edges)
- Storing list of Nodes, list of Edges separately

Adjacency matrix is more commonly found in textbooks. Distance and connectedness matrix are variations of this. Each column and row are

Storing lists of Nodes and lists of Edges.

2.2 Minimum Spanning Trees:

- Kruskal's Algorithm (Start with PQ)
- Prim's Algorithm (Start at random node)

2.2.1 Kruskal's Algorithm

In layman's terms, you put all the edges and their weights in a priority queue, where the smallest-weighted edges is first. Pop off the smallest and connect the two nodes. Keep popping off the smallest-weighted edge such that it does not create a loop. When all the elements are connected, you've found a minimum spanning tree.

2.2.2 Prim's Algorithm

In layman's terms, you start at a random node. Add all the possible edges. Choose the smallest edge and connect with it. Now update all possible edges again. Repeat, without creating cycles, until you have a spanning tree.

2.3 Connectedness:

Visually finding connectedness is easy, but using an adjacency matrix as its size grows becomes hard.

2.3.1 One Computationally Hard Method:

Notice the pattern. A shows connectedness with nodes one arc away. A^2 shows connectedness with nodes two arcs away. A^3 shows connectedness with nodes three arcs away. There are a maximum of $n-1$ arcs between two nodes. Hence, $Y = A + A^2 + A^3 + ... + A^{n-1}$ will show connectedness for the graph. Zero indicates no connection.

2.3.2 Connectedness Algorithm:

- Initialize X as the adjacency matrix (non-directed graph). Set $i = 1$ and $c = 0$
- Find $X_{ij} \neq 0 (j > i)$. Logically add row j to row i, column j to column i. Delete row j and column j from X. If no j exists, go to step 3. Otherwise, repeat step 1.
- Set $c = c + 1$. Find $k > i$ such that row k has not been deleted. If no such k exists, then stop. The graph has c components. Otherwise, set $i = k$ and go to step 2.

In layman's terms (try to get the general motion with you hands):

- Index the first layer for each nonzero. Logically AND outwards the rows and columns that nonzero has indicated. Delete.
- Increase the counter for number of components. Go to next non-deleted row in the sub matrix unless you can't

2.4 Optimal Paths:

2.4.1 Dijksta's Algorithm

Layman's Method. Visual.

- Visit node.
- Add neighbor nodes that aren't in visited nodes list into priority queue with total weight from the start.
- Choose top from priority queue and add to visited nodes list. Visit this node.
- Stop when you visit the target node.

Distance Matrix Method

The result will be a spanning tree so you can keep track of the tree in an array *parents*.

- Assign a temporary label $l(i) = \infty$, except set $l(s) = 0$. Denote $l(s)$ as permanent label (meaning officially the shortest path to this node). Set $r = s$.
- For each node i with a temporary label, redefine $l(i) = \mathbf{min}(l(p), l(p) + d(p, i))$ (where p is the node of where it came from.) Assign its parent as p if $l(i) > l(p) + d(p, i)$. Then find the node i with the smallest temporary label, make the label permanent and set $r = i$. (This is officially the shortest path to this node since taking the other nodes will yield a greater weight)
- If node t has a temporary label, repeat step 2. Otherwise the value of t's permanent label is equal to the shortest path from s to t.

2.4.2 Further Explorations

- Ford's Algorithm - compute shortest paths with negative weights
- Floyd's Algorithm - compute all shortest paths at once

2.4.3 Pollack's Algorithm

Determining the second shortest path. Really simple. Determine the shortest path via Dijkstra's. Eliminate one of the arcs and redetermine the shortest path via Dijkstra's. Repeat for each arc. Take note of the shortest path.

- Find the shortest path through the network. Number the arcs in the path 1 to m. Set $k = 1, q = \infty$
- Set the length of arc $k = \infty$ temporarily. Find the shortest path through the modified network. I fthe length is less than q, set q equal to the length of the path.
- If $k = m$, stop. Otherwise, set $k = k + 1$ and repeat step 2.