Similarity Metrics in Networks

Mentee: Samuel Hsu, Mentor: Vydhourie Thiyageswaran

2023-12-15

Introduction

This will be a brief overview of the basics of graph theory, resistor network analysis, and random walks. The ultimate goal is to find similarity metrics that can be used to compute similarities between different nodes in a graph and group them together.

Some quick notes: This writeup was adapted from a presentation I gave on this topic, so the wording and presentation of topics will be very similar. Also, most of this writeup comes from my reading of *Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation* by Fouss et al. and the textbook *Spectral and Algebraic Graph Theory* by Daniel Spielman.

Graph Theory

This first section will be devoted to untangling the idea of a graph.

The Encyclopedia Britannica defines a graph as "a network of points connected by lines", while Wikipedia defines them as "mathematical structures used to model pairwise relations between objects". In other words, the purpose of this network of points is to "model connections between things", according to Spielman. For example, imagine a social media app. You can think of a social media site as a graph: all of the different people on that social media site are represented by different points, and all of the different "friendships" on that site are represented by the lines connecting those points. (A friendship graph is one of Spielman's examples.)

Note that the points in a graph are called *vertices* or *nodes*, while the lines in a graph are called *edges*. Edges can have different weights and directions, in some scenarios. A weighted graph is one where all edges have some weight associated with them, and a directed graph is one where all edges have a directiion.

The degree of a node is the number of edges connected to it in an unweighted graph. In a weighted graph, it's the sum of all of the weights of the edges connected to the node.

Here's a picture of a graph:

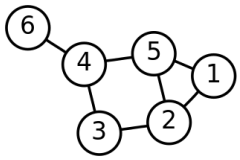

Figure 1: User:AzaToth, Public domain, via Wikimedia Commons, https://commons.wikimedia.org/wiki/File:6n-graf.svg

An important aspect of graphs is finding ways to compute how similar two nodes are (in order to group different nodes together). Much of this write-up will be focused on building up the knowledge needed to compute similarity metrics for nodes on a graph.

Matrices Associated with Graphs

One way we can analyze graphs is by associating each graph with some matrices. We first need to number our vertices. The adjacency matrix $\mathbf{A}$ tells you how all of the vertices are connected. If each row and column corresponds to a vertex, than any individual entry in that matrix corresponds to some pair of vertices. An entry takes the value 1 if there is an edge connecting those vertices, and a 0 if there isn't.

The degree matrix $\mathbf{D}$ tells you the degree of every vertex. The only nonzero entries occur when the row and the column are the same number; in other words, the only nonzero entries don't look at pairs of nodes, but only one node. The entry associated with a node is the nodes's degree.

The Laplacian matrix ($\mathbf{L} = \mathbf{D} - \mathbf{A}$) is computed by finding the difference between the degree and adjacency matrices. It measures the *smoothness of a graph function*, which is a function that maps each vertex to a number. A graph function is *smooth* if the function doesn't jump too dramatically between connected vertices.

Below is a picture of a graph and an associated graph function:

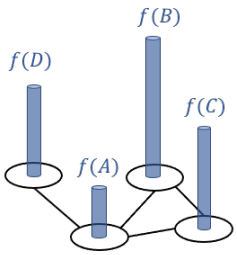

Figure 2: From Matthew N. Bernstein at https://mbernste.github.io/posts/laplacian_matrix/

The *smoothness* of a function is given by $\mathbf{f}^T\mathbf{L}\mathbf{f}$ where $\mathbf{f}$ is a column vector representing the value of our graph function at every vertex. Another way of writing this is $\sum_{u\sim v} w_{uv}(f(u) - f(v))^2$: the sum of all of the squared differences between the graph function for every pair of neighboring nodes. Smooth functions should minimize this expression.

(For those of you familiar with multivariable calculus, the graph Laplacian is the discrete version of the Laplacian operator on a function: the divergence of the gradient of a multivariable function. In a sense, this graph Laplacian acts as a sort of "second derivative" for the graph function).

Keep this idea of the graph Laplacian in your mind; it'll be important for computing various similiarity metrics.

Resistor Networks

Graphs can be used to analyze electrical circuits. In particular, we can analyze resistor networks, which are simply a collection of electrical resistors. The idea of the effective resistance between two points will be important for computing similarity metrics later.

Let's first start by defining some terms related to electrical circuits: - Current (I): the rate charge flows at, measured in amperes or amps - Voltage (V): how much some charge has to be "pushed" to get through some element of a circuit, measured in volts - Resistance (R): a property of an element of a circuit that *resists* the flow of current, measured in ohms

These three quantities are connected via Ohm's Law, which says that the voltage across a resistor is equal to the product of its resistance and the current flowing through it, or $V = IR$.

To understand this, consider a battery powering a lightbulb. If the lightbulb resists the flow of current a lot, the battery has to push harder to get more current across the lightbulb.

Below is a fun little cartoon illustrating the idea:

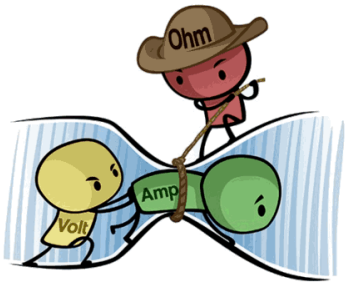

Figure 3: From https://makeabilitylab.github.io/physcomp/electronics/electricity-basics.html

A network of resistors can be analyzed using graphs and the tools of graph theory. We can imagine the points where different resistors "meet up" to be nodes, while each resistor is an edge. Since current flows differently through different resistors (and along different edges), we can make this graph a weighted graph. An edge or a resistor with high weight should have a lot of current flowing through it, and will thus have a lower resistance. An edge or a resistor with low weight should have less current flowing through it, and will thus have a higher resistance.

Random Walks

One more concept we have to discuss before exploring similarity metrics is the idea of a random walk on a graph.

Imagine you are "standing" at a vertex of a graph. The next moment, you decide to randomly walk to another neighboring vertex, and you repeat this process of randomly walking to other vertices a few times. The path you take is called a random walk.

If you're walking along an unweighted graph, when you stand at a node, you have an equal probability of walking to each one of your current node's neighbors. In a weighted graph, you have a higher probability of walking along higher-weighted edges. In general, more similar nodes will have more edges connecting them that are higher in weight, while less similar nodes will have fewer edges connecting them that are lower in weight.

The random variable $s(t)$ contains the current location of the walker, and the equation $s(t) = i$ means that a walker is at position i at time t. The probability that the walker visits a neighboring node j at time $t + 1$ given that they were just at node i at time t is $P(s(t + 1) = j|s(t) = i)$.

To more thoroughly understand the idea of the random walk, we need to develop a tool known as a Markov chain. $\mathbf{p}$ is a vector of probabilities that tells you how likely a random walker is to be at any node. As random walkers take steps to nearby nodes, the probability of being at a specific node changes over time, and so the vector $\mathbf{p}$ changes over time. As the vector $\mathbf{p}$ changes, the chain of $\mathbf{p}$ vectors over time is called a Markov chain. To see how the $\mathbf{p}$ vector changes over time, multiply it by the transition or random-walk matrix $\mathbf{W} = \mathbf{A}\mathbf{D}^{-1}$.

I'm barely skimming the surface with this description, but we finally have the tools we need to tackle our goal of computing similarity metrics in graphs.

Similarity Metrics

Here are some similarity metrics for vertices on a graph, as listed by Fouss et al.:

- the average first passage time $m(k|i)$
- the average first passage cost $o(k|i)$
- the pseudoinverse of the graph Laplacian $\mathbf{L}^+$
- the average commute time $n(i,j)$
- the Euclidean Commute Time Distance $[n(i,j)]^{\frac{1}{2}}$

The ultimate goal of these similarity metrics is to find ways to group similar vertices together. This is important for recommendation algorithms. As an example, imagine some database of people and some movies they've watched recently. (This example comes from the paper by Fouss et al.):

- "Computing similarities between people allows us to cluster them into groups with similar interest about watched movies."
- "Computing similarities between people and movies allows us to suggest movies to watch or not to watch."
- "Computing similarities between people and movie categories allows us to attach a most relevant category to each person."

Let's now go through and look at the similarity metrics:

Pseudoinverse of the Laplacian

One metric to consider is the entries in the pseudoinverse of the graph Laplacian matrix.

Not all matrices are invertible (including the Graph Laplacian), so the idea of a pseudoinverse extends the idea of an inverse to matrices that otherwise wouldn't be invertible.

The Moore-Penrose pseudoinverse of the Laplacian is calculated by this formula: $\mathbf{L}^+ = \left(\mathbf{L} - \frac{\mathbf{e}\mathbf{e}^T}{n}\right)^{-1} + \frac{\mathbf{e}\mathbf{e}^T}{n}$, where $\mathbf{e}$ is the all-ones vector and n is the number of nodes.

The pseudoinverse of the Laplacian is a similarity matrix (the similarity of two vertices i and j can be found by looking at the ith row and jth column of L^+, or vice versa, since this is a symmetric matrix).

This matrix's entries are used to calculate many of the following quantities.

Average First-Passage Time

The average first passage time, $m(k|i)$, is defined as the average number of steps that a random walker at node i takes to visit node k. It can be thought of as the *expected value* of the minimum time of hitting state k if you start at state i, represented in this formula: $m(k|i) = E[T_{ik}|s(0) = i]$

Here are some formulas describing how to calculate it: - $\begin{cases} m(k|k) = 0 \\ m(k|i) = 1 + \sum_{j=1}^{n} p_{ij} m(k|j) \end{cases}$ - $m(k|i) = \sum_{j=1}^{n} \left(l_{ij}^{+} - l_{ik}^{+} - l_{kj}^{+} + l_{kk}^{+}\right) d_{jj}$ - Computed from the pseudoinverse of the Laplacian

Average First-Passage Cost

The average first passage cost is $o(k|i)$, and it's a closely related quantity to the average first-passage time. The only difference is that a random walker incurs a cost $c(j|i)$ if they walk from i to some neighboring vertex j. So, the average first-passage cost is the average cost a random walker incurs if they want to visit any node k from node i.

Here are some formulas describing how to calculate it: - $\begin{cases} o(k|k) = 0 \\ o(k|i) = \sum_{j=1}^{n} p_{ij} c(j|i) + \sum_{j=1}^{n} p_{ij} o(k|j) \end{cases}$ - $o(k|i) = \sum_{j=1}^{n} \left(l_{ij}^{+} - l_{ik}^{+} - l_{kj}^{+} + l_{kk}^{+}\right) b_j$ - Computed from the pseudoinverse of the Laplacian, again

Average Commute Time

The average commute time is a "symmetric" version of the average first-passage time. When computing the average-first passage time from node i to j, it matters whether you're going from i to j or from j to i. So, the average commute time is a sum of the Sum of the average-first passage times *in both directions* between i and j, and it's notated as such: $n(i, j) = m(i|j) + m(j|i)$

Here are some formulas describing how to calculate it: - $n(i, j) = m(i|j) + m(j|i)$ - $n(i, j) = V_G \left(l_{ii}^{+} + l_{jj}^{+} - 2l_{ij}^{+}\right)$ - l_{ab}^{+} is an element of the matrix $\mathbf{L}^{+}$ - V_G, the volume of the graph, is the sum of all of the degrees - $n(i, j) = V_G \left(\mathbf{e}_i - \mathbf{e}_j\right)^T \mathbf{L}^{+} \left(\mathbf{e}_i - \mathbf{e}_j\right)$ - $\mathbf{e}_i$ is a standard basis vector (like $< 1, 0, \ldots, 0 >$ or $< 0, 1, \ldots, 0 >$)

The average commute time is proportional to the *effective* resistance between two nodes in the corresponding resistor network, which is why it's also called the "resistance distance" (along with the "commute-time distance").

Euclidean Commute Time Distance and Principal Component Analysis

The Euclidean Commute Time Distance is defined as $[n(i,j)]^{\frac{1}{2}}$. It's the square root of the average commute-time distance.

Here's what's interesting: You can define vectors that correspond to each node called *transformed node vectors* using the spectral decomposition of the pseudoinverse of the Laplacian matrix. Here are the formulas describing how to obtain them:

$\mathbf{x}_i' = \ ^{\frac{1}{2}}\mathbf{U}\mathbf{e}_i$, where $\mathbf{U}$ contains the eigenvectors of $\mathbf{L}^+$, while is a diagonal matrix with the eigenvalues.

The distance between the transformed node vectors is exactly the Euclidean Commute Time Distance, and taking inner products of the node vectors gets you the entries of the pseudoinverse of the Laplacian matrix. According to Fouss et al., this justifies the use of the Laplacian pseudoinverse as a similarity matrix.

Principal Component Analysis gives you lower-dimensional transformed node vectors that are still roughly separated by the Euclidean Commute Time Distance.

Conclusion

To review, we started by looking at graphs and their associated matrices. We then quickly examined resistor networks and how they could be understood using graphs. A discussion of random walks on graphs followed. Finally, all of these topics were used to construct various similarity metrics for nodes on a graph. Interestingly, the pseudoinverse of the Laplacian matrix can be used to calculate a lot of these similarity metrics. Its entries can be calculated from taking the inner products of the transformed node vectors, and its entries serve as similarity metrics themselves.

Sources:

- Spectral and Algebraic Graph Theory by Daniel Spielman
- Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation by Fouss et al.
- Quora post on the Graph Laplacian by Muni Sreenivas Pydi
- Cross Validated post on Principal Component Analysis by amoeba
- Post on the Graph Laplacian by Matthew Bernstein
- Linear Algebra with Applications by Jeffrey Holt
- Basics of Applied Stochastic Processes by Richard Serfozo
- OpenStax University Physics, Volume 2
- Optimization Models by Laurent El Ghaoui

As noted earlier, most of this writeup comes from my reading of *Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation* by Fouss et al. and the textbook *Spectral and Algebraic Graph Theory* by Daniel Spielman.

In addition, the portion on the Graph Laplacian matrix was informed by a well-written Quora post by Muni Sreenivas Pydi and a page on Matthew Bernstein's website; these two sources helped provide me with a good intuition for what the Laplacian actually does. I also consulted OpenStax Physics for the section on resistor networks, along with Linear Algebra with Applications and Basics of Applied Stochastic Processes for two perspectives on Markov chains. Finally, my understanding on Principal Component Analysis was improved by a well-written post on Stack Exchange by the user "amoeba", and my undewrstanding of spectral theory was furthered by Laurent El Ghaoui's online textbook on optimization models.