Centrality Measures

Vincenzo Bonifaci

April 5, 2016

1 Centrality measure: the degree

A crude measure of the importance of a node in a network is its degree. For example, the number of friends in a social network. For a directed network, we have a choice between the in-degree or the out-degree of nodes. For example, in a citation network, where the nodes are scientific articles and the links represent citations, it makes sense to consider the in-degree as a measure of importance.

The degree centrality is easy to compute in linear time – if the graph is stored in adjacency list format, just count the number of incident edges to each node with one pass over the graph, increasing the appropriate counter when an edge is encountered.

2 Centrality measure: eigenvector centrality

A limitation of the degree measure is that it gives the same weight to all the neighbors of a node when computing its importance. However, it may make more sense to give a larger weight to nodes that are themselves important. In a social network, for example, one node may be important because it has social ties with few but important nodes (instead of just participating in many ties).

We come to the notion of eigenvector centrality. The idea is that we would like to associate a score x_i with every node $i \in V$, and in such a way that the score of a node is proportional to the combined score of its neighbors. Thus, we would like to find a vector $x \in \mathbb{R}^n$ such that

$$x_i = \alpha \sum_{j \in V \,:\, (i,j) \in E} x_j = \alpha \sum_{j \in V} A_{ji} x_j,$$

or in vector notation, $x = \alpha A^\top x$, where α is some constant of proportionality. For an undirected graph, the adjacency matrix is symmetric, so $A = A^\top$ and the problem is equivalent to solving

$$Ax = \alpha^{-1} x. \tag{1}$$

Now, there is always a special vector v_1 such that $Av_1 = \kappa_1 v_1$, where κ_1 is the largest eigenvalue of the matrix A. This vector is what we call the *eigenvector centrality* vector; $(v_1)_i$ measures the centrality score of node i. Note that the eigenvector centrality vector solves equation (1) when $\alpha = \kappa_1^{-1}$.

How do we compute vector v_1? We start with some appropriate initial vector $x(0) \in \mathbb{R}^n$, and repeatedly perform the update

$$x(t) = Ax(t-1),$$

until the direction of the vector x stabilizes (here A is the adjacency matrix of the graph). This is called the *power method.*

The power method can be used to compute v_1 starting from any $x(0)$, as we now show. Notice that we don't care for the actual scaling of v_1 (indeed, if $Av_1 = \kappa_1 v_1$, then $Acv_1 = \kappa_1 cv_1$ for any constant c). What matters is the relative magnitude of the different entries of v_1. If we want, we can normalize the vector at each step; for example, we could always normalize it so that $\sum_i |x_i(t)|^2 = n$ for all t. In the analysis below we do not normalize the vectors.

Since the eigenvectors of A form a basis of $\mathbb{R}^n$, we can always write $x(0) = \sum_i c_i v_i$ where the v_i are the n normalized eigenvectors of A, and c_i are some appropriate constants. Then,

$$x(t) = A^t x(0) = \sum_i c_i \kappa_i^t v_i = \kappa_1^t \sum_i c_i \left(\frac{\kappa_i}{\kappa_1}\right)^t v_i. \tag{2}$$

In particular, if $|\kappa_i|/\kappa_1 < 1$, then $(\kappa_i/\kappa_1)^t \to 0$ as $t \to \infty$. So, if we can prove that $|\kappa_i| < \kappa_1$ for all $i \neq 1$, we will have shown that $x(t)$ will tend to $\kappa_1^t c_1 v_1$ (plus a vanishing error term), that is, the power method will work correctly, because the vector $x(t)$ will slowly become parallel to v_1.

One useful property of the power method is that every iteration can be implemented efficiently if the graph is sparse: $O(m+n)$ operations are sufficient to multiply A with any vector (why?).

However, we still have to clarify:

1. Convergence: Why $|\kappa_i| < \kappa_1$ for all $i \neq 1$;
2. Initialization: How to initialize $x(0)$ appropriately;
3. Running time: How large a t do we need to consider.

2.1 Convergence of the Power Method

The theorems that we state below justify the correctness of the power method. They are a special case of the so-called *Perron-Frobenius* theorem about nonnegative matrices.

Theorem 2.1. *Let A be a real nonnegative matrix, with eigenvalues $\kappa_1 \geq \kappa_2 \geq \ldots \geq \kappa_n$. Then the eigenvalue with largest absolute magnitude is κ_1. In other words, $|\kappa_i| \leq \kappa_1$ for all i.*

Proof. We prove it for the case when A is symmetric. Let $\mu \in \mathbb{R}$, $w \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ be an eigenpair of A, that is, $Aw = \mu w$. Using $w^\top w > 0$,

$$|\mu|\, w^\top w = \left|\mu w^\top w\right| = \left|w^\top A w\right| \leq \sum_{i,j} |A_{ij} w_i w_j| = \sum_{i,j} A_{ij}\, |w_i|\, |w_j| = x^\top A x$$

where x has components $|w_i|$. We have used the triangle inequality: $|a+b| \le |a| + |b|$. Rewriting,

$$|\mu| \le \frac{x^\top Ax}{w^\top w} = \frac{x^\top Ax}{x^\top x}.$$

Let $x = \sum_i c_i v_i$, where v_i are the normalized eigenvectors of A. Then

$$\frac{x^\top Ax}{x^\top x} = \frac{\sum_j c_j v_j^\top A \sum_i c_i v_i}{\sum_j c_j v_j^\top \sum_i c_i v_i} = \frac{\sum_j c_j v_j^\top \sum_i c_i \kappa_i v_i}{\sum_j c_j v_j^\top \sum_i c_i v_i} = \frac{\sum_i c_i^2 \kappa_i}{\sum_i c_i^2} \le \frac{\sum_i c_i^2 \kappa_1}{\sum_i c_i^2} = \kappa_1.$$

(Notice, incidentally, that equality is possible only when $c_1 = 1$ and $c_i = 0$ for $i \neq 1$). Therefore, $|\mu| \le (x^\top Ax)/(x^\top x) \le \kappa_1$. (This also shows that $\kappa_1 \ge 0$.) □

However, we wanted to prove $|\mu| < \kappa_1$ but we only proved $|\mu| \le \kappa_1$. The reader may wonder if it can happen that $\kappa_n = -\kappa_1$. Indeed it is possible to show that, if A is the adjacency matrix of a connected graph, this happens only when the graph is bipartite. Intuitively, the power method is not guaranteed to work in that case, because the vector $x(t)$ will "flip-flop".

2.2 How to choose $x(0)$

From (2) one can see that the power method will not work if $x(0)$ is such that $c_1 = 0$, that is, if $x(0)$ is orthogonal to v_1. To avoid this, we select $x(0)$ so that all its components are positive. Then, using the following theorem, $v_1^\top \cdot x(0) > 0$ and so $x(0)$ will not be orthogonal to v_1.

Theorem 2.2. *Let A be a real nonnegative matrix. The components of the leading eigenvector v (the eigenvector associated to κ_1) are all nonnegative.*

Proof. Again, we only prove it for the case when A is symmetric. Let v be any eigenvector corresponding to κ_1. Observe that $\kappa_1 \ge 0$, since $\kappa_1 + \ldots + \kappa_n = \mathrm{tr}(A) \ge 0$ because A is nonnegative. We have

$$\kappa_1 v^\top v = \left|\kappa_1 v^\top v\right| = \left|v^\top Av\right| \le \sum_{i,j} |A_{ij} v_i v_j| = \sum_{i,j} A_{ij} |v_i| |v_j| = x^\top Ax$$

where x has components $|v_i|$. Rewriting,

$$\kappa_1 \le \frac{x^\top Ax}{v^\top v} = \frac{x^\top Ax}{x^\top x}$$

and moreover, the last fraction is at most κ_1, by what we have shown in the proof of Theorem 2.1. Therefore, we must have equalities everywhere and x must be an eigenvector of κ_1, too, with nonnegative components.

Can there be any other eigenvector (different from x) with eigenvalue κ_1? It turns out that this is impossible in a connected and non-bipartite graph, but we omit the proof. In summary, x ($= v$) is the only eigenvector with eigenvalue κ_1, and it has all components nonnegative. □

2.3 Number of iterations required

We have already argued that every iteration of the power method can be implemented in time $O(m+n)$, which, assuming the network is connected, simplifies to $O(m)$. But the crucial question is, how many iterations are necessary if we want to achieve a small error, say less than some ϵ?

If v_1 is the centrality eigenvector, then let us measure our error by the formula

$$\sqrt{\left\| \frac{x(t)}{c_1 \kappa_1^t} - v_1 \right\|^2}.$$

After t iterations, we have

$$x(t) = \kappa_1^t \sum_{i=1}^{n} c_i \left(\frac{\kappa_i}{\kappa_1} \right)^t v_i,$$

or

$$\frac{x(t)}{c_1 \kappa_1^t} = v_1 + \frac{c_2}{c_1} \left(\frac{\kappa_2}{\kappa_1} \right)^t v_2 + \dots$$

so if we ignore the lower order terms, which vanish faster, the error at time t is approximately

$$\frac{c_2}{c_1} \left(\frac{\kappa_2}{\kappa_1} \right)^t.$$

In order to have the error at most ϵ we therefore need

$$t \geq \frac{\log(1/\epsilon) + \log(c_1/c_2)}{\log(\kappa_1/\kappa_2)}.$$

In practice, what is often done is to iterate the method until the variation between $x(t+1)/\left\|x(t+1)\right\|$ and $x(t)/\left\|x(t)\right\|$ is small enough.

2.4 Computing the second largest eigenvalue and eigenvector

Sometimes we do not want to compute the largest eigenvalue and associated eigenvector, but we want instead the second largest eigenvalue and eigenvector.

Consider for example the Laplacian matrix L, with eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$. We have mentioned that λ_2 is a good measure of the connectivity of the graph. Now, λ_2 is the second smallest eigenvalue, not the second largest, however notice that if v_i is the eigenvector related to λ_i, then

$$(\lambda_n I - L) v_i = (\lambda_n - \lambda_i) v_i,$$

so v_i is also an eigenvector of the matrix $\lambda_n I - L$ with eigenvalue $\lambda_n - \lambda_i$. The matrix $\lambda_n I - L$ has the same eigenvalues as L but in reverse order. So to compute the second smallest eigenpair of L we can compute the largest eigenvalue of L (that is, λ_n) and then use it to compute the second largest eigenpair of $\lambda_n I - L$.

So, how can we compute the second largest eigenpair of a matrix A? We have seen before that we know how to compute the largest eigenpair (λ_1, v_1). Now instead of starting from an arbitrary vector x, we make sure that we start from a vector y that has no component in the direction of v_1. One way is to define

$$y = x - (v_1^\top x)v_1.$$

In fact we can check that $v_1^\top y = 0$ and $v_i^\top y = v_i^\top x$ for all $i \neq 1$. So

$$y = \sum_{i=2}^{n} c_i v_i$$

and if we apply the power method we obtain

$$y(t) = A^t y(0) = \kappa_2^t \sum_{i=2}^{n} c_i \left(\frac{\kappa_2}{\kappa_i}\right)^t v_i.$$

Assuming only a single eigenvalue of value κ_2, the ratio $\frac{\kappa_2}{\kappa_i}$ is less than 1 so the method converges.

In practice, numerical errors may give some noise that accumulates in the v_1 component, so it is necessary to periodically remove from $y(t)$ any component in the direction of v_1 (say, once every K iterations for some value K).

3 Katz centrality

A shortcoming of the eigenvector centrality is that, for a directed network, only nodes in a strongly connected component of two or more vertices can have a positive centrality value. In particular, for networks with few or no cycles, such as citation networks, the eigenvector centrality becomes useless.

One way around this problem is to give every node some value of centrality "for free". We use the revised formula

$$x_i = \alpha \sum_j A_{ji} x_j + \beta,$$

where α and β are some positive parameters. In matrix terms, we get the linear system

$$x = \alpha A^\top x + \beta \mathbf{1},$$

which we can solve for x to get $x = \beta(I - \alpha A^\top)^{-1} \cdot \mathbf{1}$. In fact since only the relative order of the nodes is important, the parameter β is not important and we can set $\beta = 1$ to get

$$x = (I - \alpha A^\top)^{-1} \cdot \mathbf{1}.$$

The value of α, however, is relevant. If α is too small, then the measure is not very interesting because as $\alpha \to 0$, all nodes get the same weight (1). Larger values of α will differentiate more between the nodes. However we have to make sure that $\det(I - \alpha A^\top) \neq 0$, otherwise the linear system has no solution. The condition $\det(I - \alpha A^\top) = 0$ is equivalent to

$$\det(A^\top - \alpha^{-1} I) = 0,$$

so we see that the smallest value of α^{-1} for which the determinant is zero is exactly the largest eigenvalue κ_1 of $A^\top$. In practice, α is often set relatively close to this value.

How to compute the Katz centrality vector? One way of course is to solve the linear system given by its definition, but this can be expensive. Instead, similar to the power method, we can iterate the update

$$x(t+1) = \alpha A^\top x(t) + \mathbf{1}$$

until the direction of x does not change significantly.

4 Pagerank

In some situations, the Katz centrality is still not appropriate because the centrality value is spread from a node to its successors independently of how many they are. Instead, sometimes it may be more natural to subdivide the centrality value among all successors. This may also give more resistance against "spam". Mathematically:

$$x_i = \alpha \sum_j A_{ji} \frac{x_j}{\delta_j^+} + \beta,$$

where δ_j^+ is the out-degree of node j.

However, some nodes have $\delta_j^+ = 0$ (the sinks). What to do? In this case we arbitrarily consider as if $\delta_j^+ = 1$. Note that such nodes still do not contribute any value to the centrality of other nodes.

So define the matrix D to be the diagonal matrix with values $D_{ii} = \max(\delta_i^+, 1)$. Then x is the solution to

$$x = \alpha A^\top D^{-1} x + \beta \mathbf{1}.$$

This centrality measure is called the PageRank since it is a central part of Google's web search technology.

The actual value of the scalar β does not matter since it is simply a scaling factor. If we set $\beta = 1$, the explicit solution can be written as

$$x = (I - \alpha A^\top D^{-1})^{-1} \mathbf{1}.$$

To see more clearly what is happening, take $\beta = (1-\alpha)/n$; then the original equation becomes the following

$$x = \alpha W x + (1-\alpha) \frac{\mathbf{1}}{n},$$

where $W = A^\top D^{-1}$ is the *walk matrix*[1]. The equation above is nothing but *the stationary distribution equation for a random walk.* In fact, Pagerank has the following alternative interpretation, called the *random surfer model*: from a given node j, with probability α pick randomly an outgoing edge and

[1]In the random walks literature, the walk matrix is more commonly defined as $W = D^{-1}A$, so that the sum of the entries in each row is 1, and the stationary distribution is given by a row vector. However, we follow the linear algebra convention, where we seek for column vectors (instead of row vectors).

move to each adjacent node with probability $1/\delta_j^+$; and with probability $(1-\alpha)$, go to a *random webpage*, i.e., any of the n webpages, with identical probability $1/n$.

By analogy with Katz's centrality, the value of α should be less than the inverse of the largest eigenvalue of $A^\top D^{-1}$. This eigenvalue has value 1, so we need $\alpha < 1$. In practice, web search publications often mention that the value $\alpha \simeq 0.85$ seems to work well in practice.

By using non-uniform values β_i instead of the same value β for all nodes, we can give more or less importance to certain "seed" pages. We then obtain the so-called "personalized PageRank" score, defined as the solution of

$$x_i = \alpha \sum_j A_{ji} \frac{x_j}{\delta_j^+} + \beta_i,$$

which is, in matrix notation, the same as

$$x = \alpha A^\top D^{-1} x + \beta,$$

and thus has solution

$$x = (I - \alpha A^\top D^{-1})^{-1} \beta.$$

Take $\beta = (1-\alpha)v$ (again, the scaling factor in front of v does not really matter). The random surfer interpretation is now that, with probability α, we move to an adjacent node as before, and with probability $(1-\alpha)$, we follow the random distribution given by the vector v (in the uniform case, v was just $\frac{1}{n} \cdot \mathbf{1}$):

$$x = \alpha A^\top D^{-1} x + (1-\alpha) v.$$

As for the Katz centrality, the Pagerank vector is computed in practice not by inverting a matrix (which costs $O(n^3)$ and so would be extremely slow for a large network), but by interpreting the definition as an update rule and iterating it until convergence.

A better way to approximate the Pagerank vector is to use the power method. Indeed, the following identity is true for any square matrix X as long as $I - X$ is invertible:

$$(I - X)^{-1} = I + X + X^2 + X^3 + \dots,$$

therefore, taking $X = \alpha W = \alpha A^\top D^{-1}$, the Pagerank vector can also be written as

$$\begin{aligned} x &= (1-\alpha)(I + \alpha W + \alpha^2 W^2 + \alpha^3 W^3 + \dots) v \\ &= (1-\alpha) \sum_{t \geq 0} \alpha^t W^t v. \end{aligned}$$

Each term of the sum can be computed from the previous term by one multiplication by the (sparse) matrix αW, so we can use the equivalent update rule

$$x(t+1) = \alpha W x(t) + x(t),$$

$$x(0) = (1-\alpha) v.$$

In a sparse graph, every such iteration requires $O(m)$ time and typically a small number of iterations is sufficient for the centrality vector to stabilize (clearly, it also depends on how close α is to 1).

5 Closeness centrality

Let d_{ij} denote the distance, in a network, between nodes i and j, measured as the minimum number of hops needed to move from i to j. The mean distance from i to any other node of the network is given by

$$\ell_i = \frac{1}{n}\sum_j d_{ij}.$$

If a node has *small* ℓ_i, then roughly speaking it is close to many nodes of the network. So taking the reciprocal gives a centrality measure, called the *closeness centrality* of i:

$$C_i = 1/\ell_i = \frac{n}{\sum_j d_{ij}}.$$

Note, we could also use the definition $C_i = \frac{n-1}{\sum_{j\neq i} d_{ij}}$, but the relative ordering of the nodes would be exactly the same.

The closeness centrality vector can be computed by computing ℓ_i for each node i. Each ℓ_i can be computed with a single BFS visit of the graph (how?). Therefore the complete centrality vector can be computed in time $O((m+n)n)$. This is still quite slow for large networks, so faster approximation algorithms have been proposed. See the paper by Eppstein and Wang in the project proposals – you could implement this algorithm as your project.

The closeness centrality suffers from a couple of drawbacks:

- The numerical range of the centrality values is often small. There is no easy fix for this. For example, in a measure on the largest component of the actor collaboration network (using IMDB data), the node with best closeness is Christopher Lee (who played Saruman in *Lord of the Rings*), with a closeness of 1/2.4138, while the node with worst closeness is Leia Zanganeh, an Iranian actress with closeness of 1/8.6681.

- If the network is disconnected, all nodes have centrality zero (why?). An alternative is to use the harmonic mean of the distances from i to other nodes:

$$C_i' = \frac{1}{n-1}\sum_{j\neq i}\frac{1}{d_{ij}}.$$

The average closeness of a node gives one possible global measure of how closely a network is connected. We define it as:

$$\ell = \frac{1}{n}\sum_i \ell_i.$$

If the network is not connected however, $\ell = \infty$ so in that case it may be preferable to use the inverse mean of the harmonic centralities:

$$\ell' = \left(\frac{1}{n}\sum_i C_i'\right)^{-1}.$$

6 Betweenness centrality

The betweenness centrality of a node i captures, roughly speaking, how often node i is found on a shortest path between two random nodes of the network. To define it, let

- g_{st} be the number of shortest paths between s and t
- n^i_{st} be the number of shortest paths between s and t that pass through node i.

The *betweenness centrality* of i is the value

$$x_i = \sum_{s,t \in V} \frac{n^i_{st}}{g_{st}},$$

with the convention that $n^i_{st}/g_{st} = 0$ if both n^i_{st} and g_{st} are 0.

Note, The above value is not normalized between 0 and 1. If normalized values are required, the value can be divided by n^2 to ensure normalization.

Betweenness can also be defined for links, not just nodes. If we define n^{ij}_{st} as the number of shortest paths between s and t that use edge (i, j), then the *edge betweenness* of (i, j) can be defined as

$$x_{ij} = \sum_{s,t \in V} \frac{n^{ij}_{st}}{g_{st}}.$$

The notion of edge betwenness is used by certain clustering methods that we will study later on.

An alternative interpretation of edge betweenness is in terms of flows: for every pair s, t, let node s split a unit flow along all possible paths to another node t. The combined flow on edge (i, j) is exactly the edge betweenness of (i, j). Also, note that the betweenness of i is simply the sum of the betweenness of (i, j) across all j adjacent to i.

How can we compute betweenness value? We use the alternative characterization of betweenness. We first compute the flow sent from a given node a towards all other nodes of the network. We repeat this from all starting nodes. The combination of these flows will give us the betweenness for all nodes of the network.

The algorithm proceeds in three main steps:

1. Perform a BFS visit of the network starting from a.
2. Determine the number of shortest paths from a to each other node.
3. Determine the amount of flow from a to all other nodes that use each edge.

For the details, see the algorithm description in the book by Easley and Kleinberg, Section 3.B. Overall, the algorithm runs in time $O(n(m + n))$. Again, this is too large for very large networks, so you are invited to implement fast algorithms to approximate the betweenness in your project, such as the algorithm by Riondato and Kornaropoulos (see the references in the projects proposals on the website).