Machine Learning and Algorithms for Data Mining
Assessment 1
Computing Communities in Large Networks Using Random Walks

Sebastian Borgeaud — spb61@cam.ac.uk

November 28, 2017

Abstract

afdasflj

1 Main contributions, Key Concepts and Ideas

The paper presents a clustering method for large undirected graphs based on random walks. The paper is based upon the idea that before a random walk converges to the stationary distribution, it should spend more time travelling inside clusters than moving between clusters, for a suitable definition of clusters. In fact, we can use this to create clusters.

1.1 Random walks on graphs

More formally, consider an undirected graph $G = (V, E)$ where V are the vertices and E are the edges of G. Let $n = |V|$ and $m = |E|$. This graph has an **adjacency matrix** A, given by

$$A_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

The paper makes two assumptions about the undirected graphs we work with:

1. Every node in the graph is connected to itself.
2. The graph is connected, that is we can reach any node from any starting node.

We can define a random walk on this graph using the **transition matrix** P, given by:

$$P_{i,j} = \frac{A_{i,j}}{d(i)} \tag{1}$$

where $d(i) = \Sigma_j A_{i,j}$ denotes the **degree** of node i. This means that if at time t the random walker is at node i, it will move with probability $P_{i,j}$ to node j, i.e. move to a neighbour with uniform probability.

1.2 Distance between vertices and distance between clusters

For a random walk of length t, the probability of starting at node i and ending at node j is given by $P^t_{i,j}$. If t is large enough to gather some information about the structure of the graph, but not too large as to make $P^t_{i,j}$ converge to the stationary distribution, $P^t_{i,j}$ can be used to define a notion of distance between two nodes of the graph, as it has the following desirable properties:

1. If two vertices i and j are in the same cluster, then $P^t_{i,j}$ should be high.
2. The probability of $P^t_{i,j}$ is influenced by $d(j)$ as the walker is more likely to go to high degree vertices.
3. A random walk starting from a vertex i or j of the same cluster, should end in vertex k with similar probability. That is for every vertex k, $P^t_{i,k}$ and $P^t_{j,k}$ should be similar.

We can now define the distance between two vertices i and j:

Definition 1. We can now define the distance between two vertices i and j:

$$r_{i,j} = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{i,k} - P^t_{j,k})^2}{d(k)}}$$

Note this is really an euclidian distance:

$$r_{i,j} = \|D^{-\frac{1}{2}} P^t_{i,\bullet} - D^{-\frac{1}{2}} P^t_{j,\bullet}\|$$

where $D_{i,j} = \delta_{i,j} \cdot d(i)$ is the diagonal matrix of the degrees of the vertices in G and $P^t_{i,\bullet}$ is the column vector containing probabilities $(P^t_{i,k})_{1 \leq k \leq n}$. Next, we can generalise this notion of distance to clusters.

Definition 2. Let $C_1, C_2 \subset V$ be two clusters in G. The distance r_{C_1,C_2} is defined to be:

$$r_{C_1,C_2} = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{C_1,k} - P^t_{C_2,k})^2}{d(k)}}$$

where the probability $P^t_{C,k}$ to go from cluster C to vertex k in t steps is defined to be

$$P^t_{C,k} = \frac{1}{|C|} \sum_{i \in C} P^t_{i,k}$$

Again, this is really an euclidean distance as

$$r_{C_1,C_2} = \|D^{-\frac{1}{2}} P^t_{C_1,\bullet} - D^{-\frac{1}{2}} P^t_{C_2,\bullet}\|$$

1.3 Clustering algorithm

The distance introduced earlier can be used in a hierarchical clustering algorithm. First, we initialise the initial partition of the graph into n clusters, one per vertex: $\mathcal{P}_1 = \{\{v\} \mid v \in V\}$. Then the algorithm repeatedly merges clusters until only a single cluster is left, by repeating at each step k:

1. Choose two closest clusters C_1 and C_2 to merge.
2. Merge C_1 and C_2.
3. Update the distances between C_3 and its adjacent clusters.
4. Create a new partition $\mathcal{P}_{k+1} = (\mathcal{P} \setminus \{C_1, C_2\}) \cup C_3$

1.3.1 Choosing two clusters to merge

In order to reduce the complexity and to guarantee that every cluster is connected, that is any node in a cluster can be reached from every other node in the cluster without leaving the cluster, only adjacent clusters are considered for merging.

The two clusters are then chosen to be those that, when merged, minimise

$$\sigma_k = \frac{1}{n} \sum_{C \in \mathcal{P}_k} \sum_{i \in C} r_{i,C}^2$$

This is usually a NP-hard problem. However, for our definition of $r_{i,C}$, we can compute the variation $\Delta\sigma(C_1, C_2)$ that would be induced by merging C_1 with C_2, where

$$\Delta\sigma(C_1, C_2) = \frac{1}{n}\left(\sum_{i \in C_3} r_{i,C_3}^2 - \sum_{i \in C_1} r_{i,C_1}^2 - \sum_{i \in C_2} r_{i,C_2}^2 \right)$$

One can show that $\Delta\sigma(C_1, C_2)$ is related to r_{C_1,C_2} by:

$$\Delta\sigma(C_1, C_2) = \frac{1}{n} \frac{|C_1||C_2|}{|C_1| + |C_2|} r_{C_1,C_2}^2 \tag{2}$$

which allows for a more efficient computation.

1.3.2 Merging the clusters

This step is straightforward as the new cluster C_3 consists of the nodes in C_1 and C_2:

$$C_3 = C_1 \cup C_2$$

The updated probability vector $P_{C_3,\bullet}^t$ is computed using $P_{C_1,\bullet}^t$ and $P_{C_2,\bullet}^t$:

$$P_{C_3,\bullet}^t = \frac{C_1 P_{C_1,\bullet}^t + |C_2| P_{C_2,\bullet}^t}{|C_1| + |C_2|}$$

1.3.3 Updating the distances

Next, we need to compute the updated variation $\Delta\sigma(C_3, C)$ for every other cluster C. There are two cases to consider:

1. If C is adjacent to both C_1 and C_2, then we already have the values of $\Delta\sigma(C_1, C)$ and $\Delta\sigma(C_2, C)$. It is then possible to compute $\Delta\sigma(C_3, C)$ in constant time using:

$$\Delta\sigma(C_3, C) = \frac{(|C_1| + |C|)\Delta\sigma(C_1, C) + (|C_2| + |C|)\Delta\sigma(C_2, C) + |C|\Delta\sigma(C_1, C_2)}{|C_1| + |C_1| + |C|}$$

2. If C is adjacent to only C_1 or C_2, we have to compute $\Delta\sigma(C_3, C)$ using equation 2, that is

$$\Delta\sigma(C_3, C) = \frac{1}{n}\frac{|C_3||C|}{|C_3| + |C|} r^2_{C_3,C}$$

1.4 Evaluating the partitions

The final step of the algorithm is to choose the partition $\mathcal{P}_k$ that best captures the notion of community for our graph. The paper presents two methods for this:

1. Choose the partition $\mathcal{P}$ that maximises the modularity Q, defined as

$$Q(\mathcal{P}) = \sum_{C \in \mathcal{P}} e_C - a_C^2$$

   where e_C is the fraction of edges inside cluster C and a_C is the fraction of edges bound to cluster C.

2. An alternative is to use the partition $\mathcal{P}$ associated with the largest value of the increase ratio

$$\eta_k = \frac{\Delta\sigma_k}{\Delta\sigma_{k-1}}$$

   . This relies on the idea that if we merge two clusters that are very different in step k, then $\Delta\sigma_k$ will be large and thus, if $\Delta\sigma_k$ is large, then the clusters at step $k - 1$ must be relevant.

1.5 Complexity

2 Implementation

I implemented this algorithm in Python, making heavy use of `numpy` to handle the computations over the matrices and vectors.

3 Results & analysis