Clustering and community detection

Social and Technological Networks

Rik Sarkar

University of Edinburgh, 2018.
• Project list is out.
  – On email and piazza

• No Class on Tuesday 16\textsuperscript{th} October.

• Notes 4 on growth and expansion are out
Community detection

• Given a network
• What are the “communities”
  – Closely connected groups of nodes
  – Relatively few edges to outside the community

• Similar to clustering in data sets
  – Group together points that are more close or similar to each other than other points
Community detection by clustering

• First, define a metric between nodes
  – Either compute intrinsic metrics like all pairs shortest paths [Floyd-Warshall algorithm $O(n^3)$]
  – Or embed the nodes in a Euclidean space, and use the metric there
    • We will later study embedding methods

• Apply a clustering algorithm with the metric
Clustering

• A core problem of machine learning:
  – Which items are in the same group?
• Identifies items that are similar relative to rest of data
• Simplifies information by grouping similar items
  – Helps in all types of other problems
Clustering

• Outline approach:
  • Given a set of items
    – Define a distance between them
      • E.g. Euclidean distance between points in a plane; Euclidean distance between other attributes; non-euclidean distances; path lengths in a network; tie strengths in a network...
    – Determine a grouping (partitioning) that optimises some function (prefers ‘close’ items in same group).

• Reference for clustering:
  – Charu Aggarwal: The Data Mining Textbook, Springer
    • Free on Springer site (from university network)
  – Blum et al. Foundations of Data Science (free online)
K-means clustering

- Find k-clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$
  - With centers $c_1, \ldots, c_k$,
  - That minimize the sum of squared distances of nodes to their clusters (called the k-means cost)

$$\Phi_{kmeans}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{a_i \in C_j} d^2(a_i, c_j)$$
K-means clustering: Lloyd’s algorithm

• There are n items
• Select k ‘centers’
  – May be random k locations in space
  – May be location of k of the items selected randomly
  – May be chosen according to some method
• Iterate till convergence:
  – Assign each item to the cluster for its closest center
  – Recompute location of center as the mean location of all elements in the cluster
  – Repeat
• Warning: Lloyd’s algorithm is a Heuristic. Does not guarantee that the k-means cost is minimised
K-means

• Visualisations

• [http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html](http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html)

• [http://shabal.in/visualls/kmeans/1.html](http://shabal.in/visualls/kmeans/1.html)
K-means

• Ward’s algorithm (also Heuristic)
  – Start with each node as its own cluster
  – At each round, find two clusters such that merging them will reduce the k-means cost the most
  – Merge these two clusters
  – Repeat until there are k-clusters
K means: discussion

• Tries to minimise squared sum of distances of items to cluster centers
  – NP-hard. Computationally intractable
  – Algorithm gives local optimum

• Depends on initialisation (starting set of centers)
  – Can give poor results
  – Submodular optimisation can help

• The right ‘k’ may be unknown
  – Possible strategy: try different possibilities and take the best

• Can be improved by heuristics like choosing centers carefully
  – E.g. choosing centers to be as far apart as possible: choose one, choose point farthest to it, choose point farthest to both (maximise min distance to existing set etc)...
  – Try multiple times and take best result..
K-medoids

• Similar, but now each center must be one of the given items
  – In each cluster, find the item that is the best ‘center’ and repeat

• Useful when there is no ambient space (extrinsic metric)
  – E.g. A distance between items can be computed between nodes, but they are not in any particular Euclidean space, so the ‘center’ is not a meaningful point
Other center based methods

• K-center: Minimise maximum distance to center:

\[ \Phi_{k_{\text{center}}}(C) = \max_{j=1}^{k} \max_{a_i \in C_j} d(a_i, c_j) \]

• K-median: Minimise sum of distances:

\[ \Phi_{k_{\text{median}}}(C) = \sum_{j=1}^{k} \sum_{a_i \in C_j} d(a_i, c_j) \]
Hierarchical clustering

• Hierarchically group items
Hierarchical clustering

• Top down (divisive):
  – Start with everything in 1 cluster
  – Make the best division, and repeat in each subcluster

• Bottom up (agglomerative):
  – Start with n different clusters
  – Merge two at a time by finding pairs that give the best improvement
Hierarchical clustering

- Gives many options for a flat clustering
- Problem: what is a good ‘cut’ of the dendogram?
Density based clustering

• Group dense regions together
• Better at non-linear separations
• Works with unknown number of clusters
**DBSCAN**

- **Density at a data point:**
  - Number of data points within radius $Eps$
- **A core point:**
  - Point with density at least $\tau$
- **Border point**
  - Density less than $\tau$, but at least one core point within radius $Eps$
- **Noise point**
  - Neither core nor border. Far from dense regions

**Algorithm**

```
Algorithm DBSCAN(Data: $D$, Radius: $Eps$, Density: $\tau$ )
begin
  Determine core, border and noise points of $D$ at level ($Eps, \tau$);
  Create graph in which core points are connected
    if they are within $Eps$ of one another;
  Determine connected components in graph;
  Assign each border point to connected component
    with which it is best connected;
  return points in each connected component as a cluster;
end
```

- **Construct UDG of core points**
- **Connected components of the graph give the clusters**
- **Assign border points to suitable clusters** (E.g. to the cluster to which it has most edges)
DBSCAN: Discussions

• Requires knowledge of suitable radius and density parameters (Eps and τ)
• Does not allow for possibility that different clusters may have different densities
DBSCAN

• Useful in cases where it is clear which objects can be considered similar but number of clusters is not known

• Known to perform very well in real problems

• Worst case complexity: $O(n^2)$

• Current research: Making faster in special cases, approximations, distributed algorithms.
Other density based clustering

• Single linkage (same as Kruskal’s MST algorithm)
  – Start with n clusters
  – Merge two clusters with the shortest bridging link
  – Repeat until k clusters

• Other, more robust methods exist
Communities

• Groups of friends
• Colleagues/collaborators
• Web pages on similar topics
• Biological reaction groups
• Similar customers/users ...
Other applications

• A coarser representation of networks
• One or more meta-node for each community
• Identify bridges/weak-links
• Structural holes
Community detection in networks

• A simple strategy:
  – Choose a suitable distance measure based on available data
    • E.g. Path lengths; distance based on inverse tie strengths; size of largest enclosing group or common attribute; distance in a spectral (eigenvector) embedding; etc..
  – Apply a standard clustering algorithm
Clustering is not always suitable in networks

• Small world networks have small diameter
  – And sometime integer distances
  – A distance based method does not have a lot of option to represent similarities/dissimilarities

• High degree nodes are common
  – Connect different communities
  – Hard to separate communities

• Edge densities vary across the network
  – Same threshold does not work well everywhere
Definitions of communities

• Varies. Depending on application

• General idea: **Dense subgraphs:** More links within community, few links outside

• Some types and considerations:
  – Partitions: Each node in exactly one community
  – Overlapping: Each node can be in multiple communities
Comment: Finding dense subgraphs is hard in general

• Finding largest clique
  – NP-hard
  – Computationally intractable

• Decision version: Does a clique of size k exist?
  – Also NP-complete
  – Computationally intractable
  – Polynomial time (efficient) algorithms unlikely to exist

• We will look for approximations
Dense subgraphs: Few preliminary definitions

- For $S, T$ subgraphs of $V$
- $e(S,T)$: Set of edges from $S$ to $T$
  - $e(S) = e(S,S)$: Edges within $S$
- $d_S(v)$: number of edges from $v$ to $S$
- Edge density of $S$: $|e(S)|/|S|$  
  - Largest for complete graphs or cliques
Dense subgraph Problem

• Find the subgraph with largest edge density
• There also exists a decision version:
  – Is there a subgraph with edge density $> \alpha$
• Can be solved using Max Flow algorithms
  – $O(n^2m)$: inefficient in large datasets
  – Finds the one densest subgraph
• Variant: Find densest $S$ containing given subset $X$
• Other versions: Find subgraphs size $k$ or less
• NP-hard
Efficient approximation for finding dense S containing X

Let $G_n \leftarrow G$.

for $k = n$ downto $|X| + 1$ do
  Let $v \notin X$ be the lowest degree node in $G_k \setminus X$.
  Let $G_{k-1} \leftarrow G_k \setminus \{v\}$.
  Output the densest subgraph among $G_n, \ldots, G_{|X|}$.

• Gives a 1/2 approximation
• Edge density of output S set is at least half of optimal set S*
Betweenness & graph partitioning

• We want to split network into tightly knit groups (communities etc)
• Idea: Identify the “bridges” and remove them
• Bridges are “central” to the network
  – They lie on shortest paths
• Betweenness of edge (e) (or vertex (v)):
  – We send 1 unit of traffic between every pair of nodes in the network
  – measure what fraction passes through e, assuming the flow is split equally among all shortest paths.
Computing betweenness

• Computing all shortest paths separately is inefficient

• A more efficient way:

• From each node:
  – Step 1: Compute BFS tree
  – Step 2: Find number of shortest paths to each node
  – Step 3: Find the flow through each edge
  – See kleinberg-Easley for detailed algorithm
Partitioning (Girvan-newman)

Repeat:
- Find edge e of highest betweenness
- Remove e

- Produces a hierarchic partitioning structure as the graph decomposes into smaller components
- Network version of hierarchic clustering
Modularity

• What is the right “cut” in a hierarchic clustering that represents good communities?
• Clustering a graph
• Problem: What is the right clustering?
• Idea: Maximize a quantity called *modularity*
Modularity of subset $S$

- Given graph $G$
- Consider a random $G'$ graph with same node degrees (remember configuration model)
  - Number of edges in $S$ in $G$: $|e(S)|_G$
  - Expected number of edges in $S$ in $G'$: $E[|e(S)|_{G'}]$
  - Modularity of $S$: $|e(S)|_G - E[|e(S)|_{G'}]$
  - More coherent communities have more edges inside than would be expected in a random graph with same degrees
  - Note: modularity can be negative
Modularity of a clustering

- Take a partition (clustering) of $V$: $\mathcal{P} = \{S_1, \ldots, S_k\}$
- Write $d(S_i)$ for sum of degrees of all nodes in $S_i$
- It can be shown that $E[|e(S)|_{G'}] \sim d(S_i)^2$
- Definition: Sum over the partition:

$$q(\mathcal{P}) = \frac{1}{m} \sum_{i} |e(S_i)|_G - \frac{1}{4m} d(s_i)^2$$
• Can be used as a stopping criterion (or finding right level of partitioning) in other methods
  – Eg. Girvan-newman
Modularity based clustering

• Modularity is meant for use more as a measure of quality, not so much as a clustering method

• Finding clustering with highest modularity is NP-hard

• Heuristic: Louvain method:
  – Place each node in its own community
  – For each community, consider merging with neighbor.
    • Make the greedy choice – make the merge that maximizes modularity
    • Or do not merge if none increases modularity
  – Repeat

• Note: Modularity is a relative measure for comparing community structure.

• Not entirely clear in which cases it may or may not give good results

• A threshold of 0.3 or more is sometimes considered to give good clustering
Karate club hierarchic clustering

• Shape of nodes gives actual split in the club due to internal conflicts
  – Newman 2003
Correlation clustering

- Some edges are known to be similar/friends/trusted marked “+”
- Some edges are known to be dissimilar/enemies/distrusted marked “-”
- Maximize the number of + edges inside clusters and
- Minimize the number of - edges between clusters
Applications

- Community detection based on similar people/users
- Document clustering based on known similarity or dissimilarity between documents
- Use of sentiments and/or other divisive attributes
Features

• Clustering without need to know number of clusters
  – k-means, medians, clusters etc need to know number of clusters or other parameters like threshold
  – Number of clusters depends on network structure
• Actually, does not need any parameter
• NP hard
• Note that graph may be complete or not complete
  – In some applications with unlabeled edges, it may be reasonable to change edges to “+” edges and non-edges to “-” edges
Approximation

• Naive 1/2 approximation:
  – If there are more + edges
    • Put them all in 1 cluster
  – If there are more - edges
    • Put nodes in n different clusters

• (not very useful)!
Better approximations

- 2 ways of looking at it:
  - Maximize agreement or Minimize disagreement
  - Similar idea, but we know different approximation algorithms
- Nikhil Bansal et al. develop PTAS (polynomial time approximation scheme) for maximizing agreement:
  - (1-\(\varepsilon\)) approximation, running time \(O(n^2 e^{O(1/\varepsilon)})\)
- Min-disagree:
  - 4-approximation
Local detection of communities

• Suppose there is a huge graph, like www, or facebook network
• We often want to find the community that contains a particular node or group
  – E.g. to make recommendations: “your friends have watched this movie...”
  – To infer preferences and attributes
• Running a full scale community detection is computationally impractical
• We do not know the number of communities
• A “local method” like DBSCAN can help
Conductance: measure of edges inside community vs outside

- Given subsets $S, T$ in $V$
- $e(S, T)$: set of edges between $S$ and $T$
- Volume of edges: $\text{vol}(S) = \sum_{v \in S} d(v)$

- Conductance of $S$ is defined as:

\[
\Phi(S) := \frac{e(S, \overline{S})}{\min(\text{vol}(S), \text{vol}(\overline{S}))}
\]

- Communities are likely to have low conductance
Personalised pagerank

• Given a seed set X
• Find the community S that contains X
• Pagerank style: Use random walks
• Algorithm
  – Set a limit k to number of steps in random walks
  – Repeat:
    • Select at random a start point from X
    • Take k random steps in the graph
  – Count how frequently each node occurs – pagerank
  – Nodes in the community have high pagerank
Personalised pagerank

• Alternative Algorithm
  – Set a probability to reset random walk
  – Repeat:
    • Select at random a start point from X
    • With probability $1 - \varepsilon$ move to a random neighbor
    • With probability $\varepsilon$ move to a random node in X
    • Count how frequently each node occurs – pagerank
  – Nodes in the community have high pagerank
Personalised pagerank

- Communities have low conductance
- Therefore, short random walks will leave the community only rarely
- Therefore, nodes in the community of X will have high pagerank compared to those outside
- It can be proved that nodes outside a low conductance community will occur infrequently.
  - We will omit this proof