Traditional Machine Learning for Graphs

Yujia Zhu

Tradional ML uses **hand-craft** features and then puts these features into a machine learning interface (e.g. SVM, Random Forest, Neural Network, etc.) to do downstream tasks. For graphs, there mainly exist three kind of tasks:

- Node-level Prediction
- Link-level Prediction
- Graph-level Prediction

For different kinds of tasks, we will design different hand-craft/hand-designed features. We will first introduce node features, then edge/linking features and graph features at last.

Node-level Features

Node-level features mainly include node degree, node centrality, clustering coefficient and graphlets.

Node Degree

For undirected graphs, the node degree k_v of a node v is defined as the number of edges/the number of neighbouring nodes the node v has.

For directed graghs, node degree needs to be divided to *in-degree* and *out-degree*. In-degree means the number of edges coming into a node in a directed graph while out-degree means the number of edges coming out from a node.

Node Centrality

Node Degree treats all the neighbouring nodes without capturing their importance. Node Centrality c_v takes node importance into consideration. The core of defining node centrality is how to model the importance of a node. By different measure ways, we can get different kinds of node centrality.

Eigenvector Centrality

We have a common view that if a node v is surrounded by important neighbouring nodes $u \in N(v)$, then node v is also important. So, one way to measure node importance/centrality is to sum the neighbouring nodes' importance, which can be expressed as:

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$

λ is a positive constant (and you will see it is the eigenvalue of the adjacency matrix later). The equation can be converted to the matrix formula as

$$\lambda c = Ac$$

where $c = (c_1, c_2, ..., c_{|V|})^T$, A is the adjacency matrix of the graph. So, we can easily find that the centrality is the eigenvector. By Perron-Frobenius Theorem, the maximum eigenvalue $\lambda_{\max}$ is unique and positive and the corresponding eigenvector $c_{\max}$ can be chosen to have strict positive components. We use $c_{\max}$ to indicate node centrality and it is called eigenvector centrality.

Bewteenness Centrality

For a node v, if it lies on many shortest paths between other nodes, then v plays in an important role. So, betweenness centrality is defined as

$$c_v = \sum_{s,t:s\neq v\neq t} \frac{|SP(s,t|v)|}{|SP(s,t)|}$$

where $|SP(s,t)|$ means the number of shortest paths between s and t and $|SP(s,t|v)|$ means the number of shortest paths between s and t that contain v.

Closeness Centality

A node v is important if it has small shortest path lengths to all other nodes. Closeness centrality is defined on this criterion. Closeness centrality is defined as

$$c_v = \frac{1}{\sum_{u\neq v} len(SP(u,v))}$$

Clustering Coefficient

Unlike node degree and node centrality which show the characteristic of a given node, clustering coefficient shows the feature of the neighbourhoods of a given node. Clustering coefficient measures the connected extent of a node's neighbouring nodes and is defined as follows:

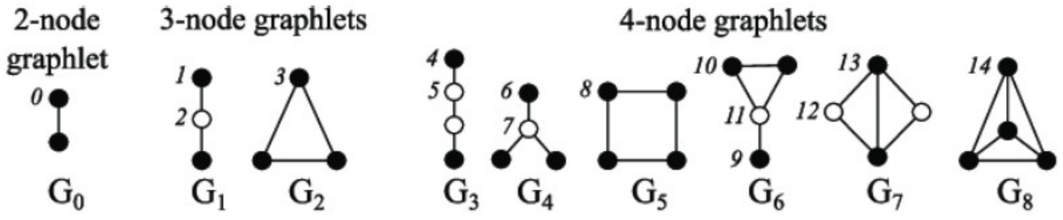
$$e_v = \frac{|E(N(v))|}{C^2_{k_v}}$$

where $|E(N(v))|$ means the number of edges among the neighbouring nodes of the node v ,$C^2_{k_v}$ is the maximum possible number of node pairs among the neighbouring nodes of the node v. We can easily derive that $e_v \in [0,1]$.

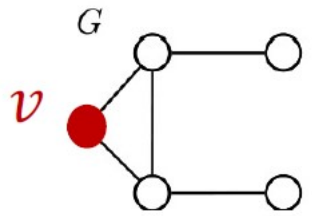
In fact, clustering coefficient counts the number of triangles in the ego-network of the node (the ego-network only takes the node and its neighbouring nodes into consideration).

Graphlets

Just as the name implies, graphlets is a table/list of graph modes. Given the number of the nodes and assumed that the graph is strong-connected, we can get all the possible composition modes. As an example, the following figure shows all the composition modes of graph containing 2/3/4 nodes.



After giving the modes, the topological location of the node in a graphlet is assured. Note that graphlets only measure the local network's topology. We will use an example to show how graphlets map the node to a vector.



We only consider graphlets on 2 to 3 nodes. For the node v in the graph G, node 0 in G_0 emerges for 2 times; node 1 in G_1 energes for 2 times; node 2 in G_1 energes for 0 times; node 3 in G_1 energes for 1 times. So the graphlets vector of v is $(2, 2, 0, 1)^T$.

With graphlets, we can compare the local topological similarity by the vectors. Since we consider a variety of modes, graphlets is a more effective local measurement of a node than node degree and clustering coefficients.

Link-level Features

We usually need to predict new links based on existing links. The key of this prediction problem is to design features for a pair of nodes. We mainly have distance-based feature, local neighbourhood overlap and global neighbourhood overlap.

Distance-based Feature

Distance-based feature is to some extent naive. It directly uses the shortest-path distance between two nodes as the feature.

Local Neighbourhood Overlap

Local neighbourhood overlap captures the number of neighbouring nodes shared between two nodes v_1 and v_2. There are several versions of how to calculate the index.

1. Common Neighbours: Just use the number of overlapping neighbourhoods.

$$I_{v_1,v_2} = |N(v_1) \cap N(v_2)|$$

2. Jaccard's Coefficient: Convert the absolute value to the relative value.

$$I_{v_1,v_2} = \frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$$

3. Adamic-Adar index: Use the node degree of overlapping neighbourhoods in the definition.

$$I_{v_1,v_2} = \sum_{u \in |N(v_1) \cap N(v_2)|} \frac{1}{\log(k_u)}$$

Global Neighbourhood Overlap

Local neighbourhood overlap still has limitations. $|N(v_1) \cap N(v_2)|$ may equal to $\emptyset$ under some circumstances, but the two nodes still have the probability to be linked. Global neighbourhood overlap resolves the problem by considering the entire graph.

Instead of only considering one-step neighbours, we can convert to think of k-step neighbours. Node v and node u are k-step neighbours if there exists a path whose length is k between u and v. We can easily count the number of k-step neighbours by the power of adjacency matrix.

Suppose the adjacency matrix of the graph is $A = [a_{uv}], a_{uv} = 1$ if node u and node v are neighbouring nodes. Let $P_{uv}^{(k)}$ is the number of paths whose length is k between u and v. We can prove that $P^{(k)} = A^k$ (Proof is easy and we omit it).

With the help of $A, A^2, A^3, ..., A^l, ...$,we can compute global neighbourhood overlap. A paticular way is to compute **Katz index**, which is defined as:

$$S_{v_1,v_2} = \sum_{l=1}^{\infty} \beta^l A^l_{v_1,v_2}$$

where β is the discounting factor. So, Katz index is the weighted sum of the number of k-step neighbours, where k ranges from 0 to infinity. By geometric series, Katz index can be computed easily as:

$$S = \sum_{l=1}^{\infty} \beta^l A^l = (I - \beta A)^{-1} - I$$

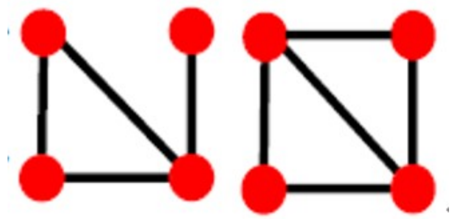
Graph-level Features

Graph-level features is to characterize the structure of an entire graph. We usually use **graph kernels** to map the graph to a vector and compare the simialrity between graphs on the vector level.

Graph kernel is a matrix $K(G, G')$, measuring the similarity between data. There exists a feature representation funtion $\phi(.)$ such that $K(G, G') = \phi(G)^T \phi(G')$. With graph kernels, the feature-extraction of a graph shifts from designing features to designing kernels.

Bag-of-* Kernel

Like bag-of-words in NLP, bag-of-* kernel regards nodes as words. For example, we give two graphs listed following and use different kinds of kernels on the two graphs.

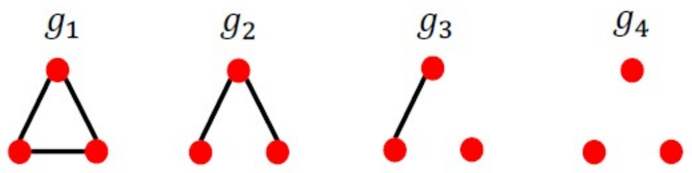


- If we use bag-of-node kernel(map the graph to the number of the nodes), two graphs all have 4 nodes so for each graph $G, \phi(G) = 4$. Two graphs have the same representation vector under this kernel.
- If we use bag-of-degree kernel(map the graph to the node degree for each node), the left graph has one node with degree 1, two nodes with degree 2 and one node with degree 3, while the right graph has two nodes with degree 2, two nodes with degree 3. So the kernel will map the left graph to $(1, 2, 1)^T$ and the right graph to $(0, 2, 2)^T$. Under this kernel, two graphs have differnet representation vector.

Graphlet Kernel

Graphlet kernel is the bag-of-graphlet kernel. The graphlet here in on the graph-level, so it is not the same from node-level graphlets which has been defined before.

The graphlet lisk $G_k = \{g_1, g_2, ..., g_{n_k}\}$ is all the possible graph modes with node number k. For example, G_3 has 4 elements as shown in the following figure. From the figure, we can find that nodes in the graph is not necessarily to be connected.



Then graphlet kernel will map the graph by counting the occurance of the graphlet in the graph $\phi(G) = (|\{g_i \subset G, g_i \in G_1\}|, |\{g_i \subset G, g_i \in G_2\}|, ..., |\{g_i \subset G, g_i \in G_K|\})$. To avoid the size of the graph influencing the kernel outcome,we usually normalize the feature vector after counting, namely $\phi(G) := \frac{\phi(G)}{\sum \phi(G)}$.The kernel is $K(G, G') = \phi'(G)\phi(G)$.

But, counting graphlets is very time-consuming. It is a NP-hard problem under the worst case, so we need to design a more efficient kernel.

Weisfeiler-Lehman Kernel

The idea of Weisfeiler-Lehman kernel (WL kernel) is to use neighborhood structure to iteratively enrich node vocabulary. We will first assign an initial label $c^{(0)}(v)$ to each node v. Then we will iteratively refine node label by the step:

$$c^{(k+1)}(v) = HASH\left(\left[c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right]\right)$$

We will count all the labels emerged in $c^{(0)}(v), c^{(1)}(v), ..., c^{(K)}(v)$ and get the representation vector by the counting outcome. The algorithm may be blurred when you first see it, but it will be clear when you see an example. The example is really long so I won't type the example here. You can check slides 55-59 of Lecture 2 of CS224W.

Summary

In this file, we cover how to design features from the node-level, link-level and graph-level.

1. Node features include: Node degree, centrality, clustering coefficient, graphlets.
2. Link features include: Distance-based feature, local/global neighborhood overlap.
3. Graph features are obtained by graph kernel. Graph kernels include: Bag-of-* kernel, Graphlet kernel, WL kernel.