Chapter 12 - Outlier Detection in Graphs and Networks

1 Introduction

The dataset may consist of many small graphs over a common set of nodes (in this case we are looking for outlier graphcs) or it could be one large graph (in this case we are looking for outlier nodes and edges). We also may have graphs that change over time.

2 Outlier Detection in Many Small Graphs

We treat each graph is a single point and use multidimensional outlier detection. To represent a graph as a point, we can mine frequent subgraphs and represent a graph as a bag of frequent subgraphs. You can also use k nearest neighbors with a graph distance metric (e.g. graph edit distance, largest commons subgraph, largest matching node set). There are also graph kernels (e.g. random walk kernel, shortest path kernel), so you can make a kernel (similarity) matrix and use a one-class SVM, soft kernel PCA, and/or the Mahalanobis method.

3 Outlier Detection in a Single Large Graph

Let our undirected graph be $G = (N, A)$ with n nodes and m edges.

To find node outliers, we need some features. We consider a node i's 1-neighborhood and compute the number of neighbors (n_i), number of edges between all neighbors (e_i), sum of weights to neighbors (w_i), and the principal eigenvalue of the weighted subgraph in the 1-neighborhood of i (λ_i). We can then plot n_i vs. e_i, w_i vs. e_i, and λ_i vs. w_i and look for deviations from the trend (which will be a power law distribution). Or, more generally, we can consider the $1, 2, ..., k$-neighborhood and compute features for each, create a point, and use our regular multidimensional outlier detection algorithms. You can also use domain specific features (e.g. number of messages sent between node and its k-neighborhood). If you want to use the Mahalanobis method, you need to make a similarity matrix (use SimRank, PageRank difference, Jaccard coefficient, or Katz measure as your similarity metric).

To find linkage (edge) outliers, matrix factorization works well. The edges are represented with adjacency matrix A. We then minimize $||A - UV^T||^2$ subject to $U, V \geq 0$. Then $R = A - UV^T$ gives us the outlier scores for each edge, and you can use extreme value analysis here. The entries of U and V can provide some interpretability as well. When doing the optimization, you should use regularization term $\alpha(||U||^2 + ||V||^2)$. For large networks, you need to use special techniques to make the optimization tractable. You can compute a score for each node i by averaging row i and column i of R (ignore the diagonal entries though). Notice that this technique can be extended to directed and weighted graphs. If you set the diagonal entries of A to the node's degree, you get an eigendecomposition where $U = V = P\sqrt{\Lambda}$ and $P\sqrt{\Lambda}$ is an embedding of each node (so you can use Mahalanobis method). And of course, if you don't want to use the adjacency matrix, you can use a kernel method instead.

We can use clustering methods for linkage outliers as well. We cluster nodes and edges connecting clusters are therefore outliers. To avoid the instability of random initialization, we use randomized ensembles for clustering. To do this, we randomly sample (you can be clever about the sampling strategy) edges from

the adjacency matrix and find connected components. Given clusters $\mathcal{C} = C_1, ..., C_k$, we want to find $p_{ij}(\mathcal{C})$, which is the probablity that a randomly chosen edge connects C_i and C_j (this is easily computed from the data). We also define $I(i, \mathcal{C})$ to be the cluster index (1 to k) for node i. The likelihood fit of an edge is therefore $p_{I(i,\mathcal{C}),I(j,\mathcal{C})}$. We should do this for each clustering in the ensemble and take a median, we call this $\mathcal{MF}(i, j, \mathcal{C}_1, ..., \mathcal{C}_r)$. If we have multiple graphs, we can define this metric for an entire graph $\mathcal{GF}(G, \mathcal{C}_1, ..., \mathcal{C}_r) = (\prod_{(i,j)\in G} \mathcal{MF}(i, j, \mathcal{C}_1, ..., \mathcal{C}_r))^{1/|G|}$.

To find subgraph outliers, we can use minimum description length (MDL). Basically, we look for a common substructure and compress it into a single node. The description length is $F1(S, G) = DL(G|S) + DL(S)$, which leads to high outlier scores for individual nodes, so we fix this with $F2(S, G) = Size(S) + Instances(S, G)$, where $Size(S)$ is is the number of nodes in the subgraph and $Instances(S, G)$ is the number of instances of S in G. Another option is the SUBDUE method.

4 Node Content in Outlier Analysis

Sometimes, nodes have content (e.g. words) inside them that we want to leverage. Imagine we have adjacency matrix A and content matrix C (one row per example, one column per word, and value is frequency). We can then use matrix factorization and minimize $||A - UV^T||^2 + \beta||C - UW^T||^2 + \alpha(||U||^2 + ||V||^2 + ||W||^2)$ subject to $U, V, W \geq 0$ using gradient descent. Then you can take $A - UV^T + C - UW^T$ to get outlier scores. Heterogeneous Markov Random Fields and Tie Strength are also useful here.

5 Change-Based Outliers in Temporal Graphs

This is also called evolutionary network analysis. We have node anomalies (e.g. the edges around a node are doing strange things), linkage anomalies (e.g. strange edges), community evolution (e.g. unsual changes in community structure), distance evolution (e.g. distance between nodes changes a lot), and latent embedding changes (i.e. if you embed the graph in a smooth space, are there changes there?).

We can create an adjacency matrix, $A(i)$, around a node i. If we keep track of how the eigenvectors of $M(i) = A(i)^T A(i)$ change, then we can detect anomalies in the amount of activity (magnitude of eigenvectors changes) and linkage patterns (angles of eigenvectors change).

We can also use linkage anomaly detection functions here. We just need to update clusters as we get new clusters. To do this, we need reservior sampling with a monotonic set function (i.e. monotonically non-decreasing set function has if $S_2 \subseteq S_1$ then $f(S_1) \geq f(S_2)$). Some options are the number of connected components in S or the number of nodes in the largest connected component of S. Making some insights from this, we can use a special hash function to determine which edges to drop and add as we process a stream.

The ENetClus (extends NetClus) lets us do soft clustering on data streams and detects changes in those clusters.

We can also maintain a differential graph, which measures how much activity happens on an edge over time.

GraphScope is a good technique for handling bipartite graphs.

We may also want to look at how shortest paths between nodes changes. This is expensive, so there are randomized heuristic techniques for approximating it. See the book for links to these papers.

We can also do matrix factorization $A_t \approx U_t V_t^T$ by minimizing $\sum_{t=1}^{T} ||A_t - U_t V_t^T||^2 + \alpha \sum_{t=1}^{T-1} ||U_t - U_{t-1}|| + \beta \sum_{t=1}^{T-1} ||V_t - V_{t+1}||^2$ subject to nonnegativity constraints. There a variety of ways to extend this method with new regularization functions, orthogonality constraints, etc. This method works for many kinds of graphs.

6 Conclusions and Summary

There are various ways to define outliers in graphs (e.g. nodes, edges, subgraphs) and we get even more options when the graph evolves over time. We also have techniques for the situation when the nodes have content.