Project 4 : Anomaly Detection
UnityId : akagrawa
Paper No : 4
Publication : NetSimile : A Scalable Approach to Size-Independent Network Similarity

Introduction: This document contains the description of NetSimile[1] : A Scalable Approach to Size-Independent Network Similarity) for the purpose of anomaly detection. The algorithm is implemented to detect the anomalies in the time sequence in the graph-based dataset. The report describes the nature and type of graph used for this algorithm followed by the algorithm description. Further in this report, the algorithm performance and results over the graph datasets are illustrated. Please refer **netsimile.R** for the algorithm implementation code and **README.md** file for the code description and usage. For the output anomaly results, please refer the folder "**output files"**.
Note: I have kept the top 15 anomalies. Not ignored any anomalies since for the large graphs taken in this case, the number of anomalies are not big enough to be ignored [merely 1-2%]. This assumption is taken based on the observation not due to the computation complexity as it is fairly easy to calculate top X anomalies.

Graph Description: The anomaly detection is performed on the graph data-set of **enron-nonempty[3], as-733[4] and reality_mining_voices[5]** data. These data files contain the vertex-vertex connection separated by the new line which is parsed and loaded into graph-like data structures for algorithm implementation. From the given dataset, we can conclude that the graphs are **simple(no parallel and self-loop), dynamic(across different time slots), undirected, unweighted and unlabeled**. The large graph as-733 is more dense as compare to other graphs.

Algorithm: "NetSimile : A Scalable Approach to Size-Independent Network Similarity"
Netsimile[1] is an approach to find similarity between graphs by computing signature vectors that captures most of the features in the aggregated form. These signature vectors can be further used for any data mining methods like clustering, anomaly detection, pattern finding etc. The below implementation is to detect anomaly in the graphs.

Algorithm is implemented into four stages
Stage 1. **Feature Extraction :** Seven features(degree, clustering coefficient, average number of node's two-hop away neighbors, average clustering coefficient of node's neighbours, number of edges in egonetwork of node, number of outgoing edges from node's egonetwork, number of neighbours of node's egonetwork) which comprises of local network properties of graph is captured in (node **x** feature) matrix.

Stage 2. Feature Aggregation : Aggregating the above features vectors in 5 different measures(median, mean, standard deviation, skewness, and kurtosis) and forming a single signature vector for a graph.

Stage 3 : Similarity Score : Computing similarity between signature vectors as distance measure using **canberra distance** and captured as scores.

Stage 4: Anomaly Detection : These scores are then used to detect anomalies by computing moving range average. Points above a threshold are considered as anomalous points in time sequence. Threshold is computed using median + 3 * moving_range_average.

Algorithm Evaluation: The goodness of the algorithm can be further discussed with the following points:

a) **Improvement :** In order to be more robust and possibly identify more anomalies while not producing false positives, multiple signature vectors for a graph can be computed. Instead of just taking one aggregated signature vectors, multiple aggregated signature vectors for each type of aggregation (median, mean, standard deviation, skewness etc.) can be taken for computing multiple scores. In this way, we can take union of all the anomalies in order to identify more anomalies and also giving more weights to common anomalies will reduce the false positive cases.

b) **Time Complexity**: The algorithm is gives linear time complexity[1] i.e. it scales with the number of nodes in the graph. It takes O(nlogn + n) time complexity, where nlogn is the complexity to compute median of the features which requires sorting operation. Rest all the computations are linear time computation i.e. one time excess of graph nodes. The algorithm is already linear, hence computationally it is very fast but it can be further reduced using parallel processing techniques. Netsimile has a great scope for parallel processing since all the signature vectors are calculated for each graph which can be computed in disjoint parallel threads. Also for the features required are local network properties, hence these computation can also be parallelized for very large graphs. Map-reduce programs can make this computation extremely fast for large scale graphs.

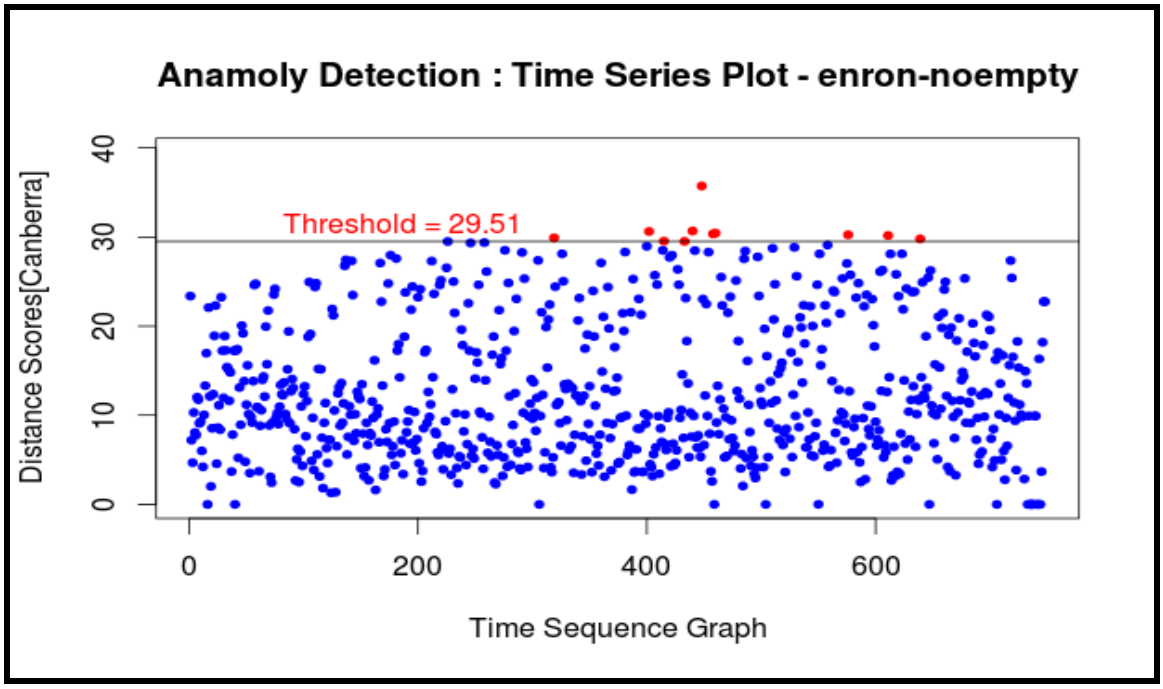
c) **Space Complexity:** O(Tn) , where T is the number of features and n is the number of vertices in the graph. T in this case is constant which is small as compare to n. T consists of space to store 7 features, signature vectors and similarity scores. Also with the linear space complexity it is highly optimized.

d) **Failure cases in anomaly detection:** This algorithm checks for point anomalies, which is computing similarity scores between two graphs and checking against threshold score . There is no concept of sliding window defined for a time evolving graphs in this algorithm for which a window score can be calculated. Here the window size can be parameterized based on the graph context and thus contextual anomalies can also be be detected, which is not considered in this algorithm.

e) **Parameter free and Deterministic :** Netsimile is parameter free. It computes the signature vectors of the graphs without any external parameters. For the anomaly detection part also, the threshold is computed by its moving range average of the scores which is also computed internally as similarity measures. Netsimile is deterministic for the computation of signature vectors for which all the computation are performed in deterministic order and no stochastic assumptions or randomized selection of nodes are taken in account. If random nodes are selected for the implementation, then there is very good chance of neglecting those nodes whose local network properties are anomalous.

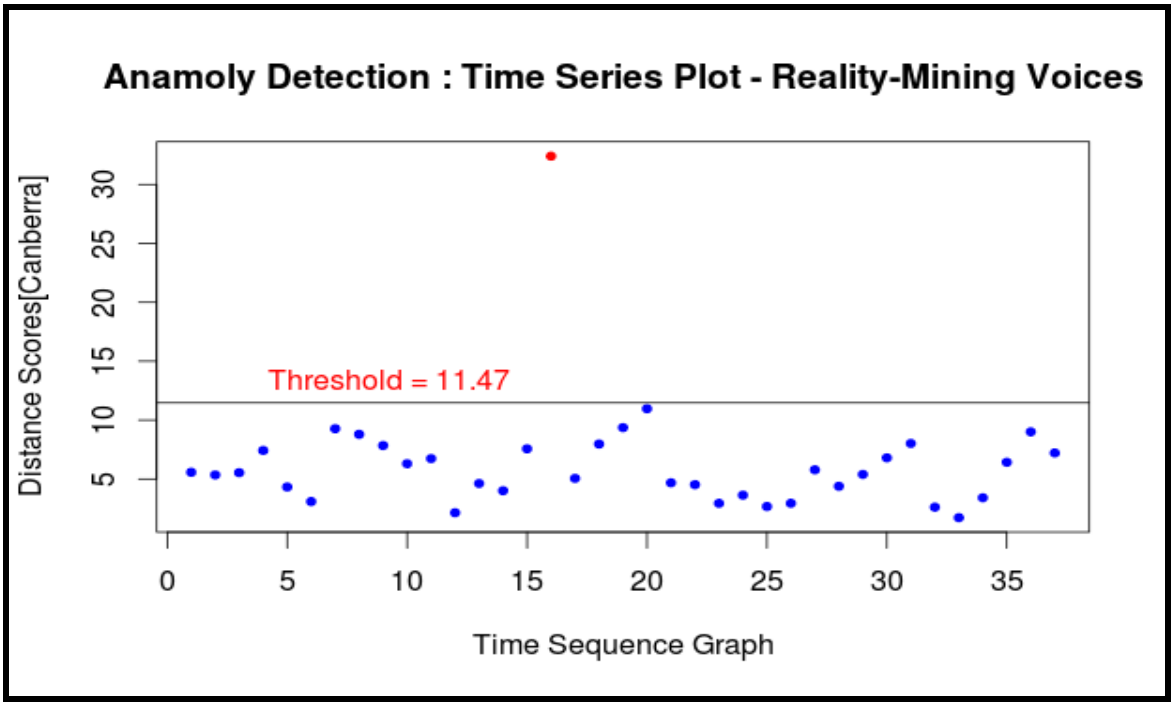
f) **Algorithm Performance on datasets:** Below are the plots which shows the anomalous time points for the different graphs.

1. **Graph : Enron-nonempty**



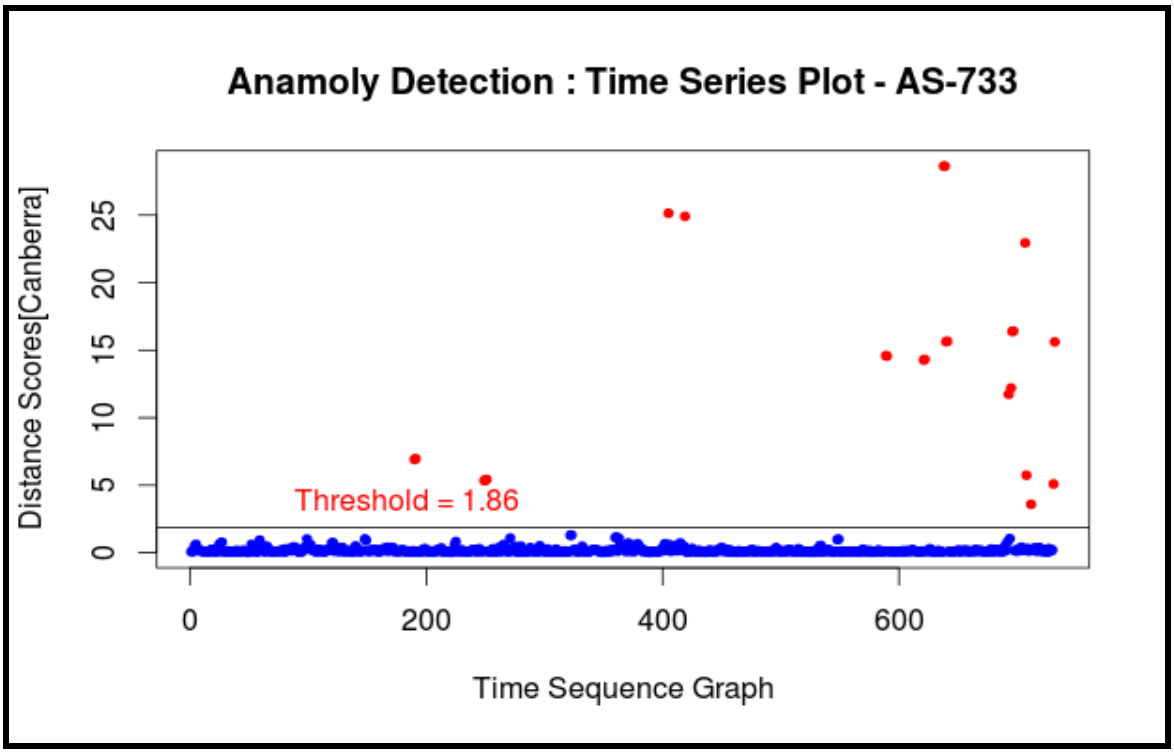
Percent anomalous : 11 out of 749 [1.46%]

2. **Graph : Reality Mining Voice**



Percent anomalous : 1 out of 38 [2.63%]

3. **Graph : As-733**



After pruning the adjacent anomalies : 15 out of 733 [2.04%]
Note: The graph above shows anomalies without pruning.

g) **Sensitivity:** Without the ground truth evaluation, the sensitivity in terms of true positive cannot be determined. Whereas if the output sensitive nature of algorithm is concerned its running time is not dependent on the size of output since it is merely 1-2% of the entire data.

h) **Anomalous points:** Anomalous points from the above observation are near to each others forming small clusters, but these clusters spread only across certain range across the spread. Comparing the signature vectors of anomalous point Vs normal point, it is observed that there exist comparatively large gap between the features scores(clustering coefficient, egoNetwork properties) of the anomalous time point adjacent graphs as compare to a normal time points adjacent graphs. These observations reveal that the local network properties of anomalous point graphs has larger variation than that of normal point graphs.

References :

[1] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "NetSimile: A Scalable Approach to Size-Independent Network Similarity", ;presented at CoRR, 2012.
[2] Kolaczyk, Eric D., and Gábor Csárdi. Statistical Analysis of Network Data with R. NewYork, NY: Springer New York, 2014. Web.
[3] "Enron Email Network." SNAP: Network Datasets:. http://snap.stanford.edu/data/email-Enron.html
[4] ”Autonomous Systems AS-733." SNAP:Network Datasets: Autonomous Systems.
[5] “Reality Commons." : http://realitycommons.media.mit.edu/realitymining.html
[6] http://cran.r-project.org/web/packages/sna/sna.pdf