Spatio-Temporal Anamoly Detection with Unsupervised Learning using Network Struture

Juhi Amitkumar Naik juhinaik@stanford.edu
Stanford University

Kratarth Goel kratarth@stanford.edu
Stanford University

Abstract

With the growth of these social and information networks, there is a rising need for detection of both spatial and temporal anomalies in the network components. Numerous applications in dynamic social networks, ranging from telecommunications to financial transactions, create evolving datasets. Detecting outliers in such dynamic networks is inherently challenging, because the arbitrary linkage structure with massive information is changing over time. We use unsupervised learning methods to be able to extract rich hierarchical features that help us improve performance on these tasks. The focus is on special types of anomalies: global, neighbor-based, and community-based in both the spatial as well as temporal domain. We propose various feature extraction methods that are fit for spatial anomaly detection, and then extend them to temporal features. These features are then subjected to either algorithms fit for spatial anomaly detection or specialized algorithms that detect temporal anomalies and ensure that this can be highly efficient in large and gradually evolving networks. We use the Intel Sensor Dataset as the real world network on which we analyze all of our approached. Along with this, synthetically generated anomalous graphs are also analyzed.

.

1. Introduction

Outlier detection is a task to uncover and report observations which appear to be inconsistent with the remainder of that set of data. Since outliers are usually represented truly unexpected knowledge with underlying value, research has been widely studied in this area, often applicable to static traditional strings or attribute-value datasets. Little work, however, has focused on outlier detection in dynamic graph-based data. With the unprecedented development of social networks, various kinds of records like credit, personnel, financial, medical, etc. all exist in a graph form, where vertices represent objects, edges represent relationships among objects and edge weights represent link strength. Graph-based outlier detection problem is specially challenging for three major reasons as follows:

Dynamic changes: Vertices, the relationships among them as well as the weight of the relationships are all continuously evolving. For example, users join friendship networks (e.g. Facebook), friendships are established, and communication becomes increasingly frequent. To capture outliers in evolving networks, detecting approaches should obtain temporal information from a collection of snapshots instead of a particular instant. For example, snapshots of the Facebook graph should be taken periodically, forming a sequence of snapshot graphs.

Massive information: Compared with average data sets, social networks are significantly larger in size. The volume is even larger when the network is dynamic, massive information involved in a series of snapshots with millions of nodes and billions of edges. In this case, it is difficult for algorithms to obtain full knowledge of the entire networks. **Deeply hidden outliers**: Recent studies suggest that social networks usually exhibit hierarchical organization, in which ver-

tices are divided into groups that can be further subdivided into groups of groups, and so forth over multiple scales. Therefore, outliers are more difficult to distinguish from normal ones if they are hidden deeply among their neighbored but not globally.

However, outlier detection in social networks has not yet received as much attention as some other topics, e.g. community discovery. Only a few studies have been conducted on graph-based outlier detection. While a more detailed discussion on these approaches will be provided in section 2, it suffices to point out here that most of these approaches identify outliers in un-weighted graphs from a more global perspective. For example, community based algorithms identify objects whose evolving trends are different with that of entire community. All such global outlier detection algorithms require the entire structure of the graph be fully known, which is impractical when dealing with large evolving networks. Furthermore, the local abnormality may be highly covered by global evolution trend. Thus, existing global methods fail to identify the objects with abnormal evolutionary behavior only relative to their local neighborhood.

2. Related Work

Anomaly detection in weighted graphs: OddBall algorithm (Akoglu et al., 2010) is a popular algorithm for anomaly detection in graphs and network structures. The main algorithm in OddBall is to construct an induced sub-graph or 'ego-net' around each node in the graph, and then try to compute sets of features that are most characteristic for a majority of the egonets. The authors of this paper have identified various forms of local anomalies that can be found in weighted, static networks, i.e. networks with weighted edges that represent a single snapshot in time.

Local and global anomaly detection in static, un-weighted graphs: In (Vengertsev & Thakkar, 2005), the authors have tried to implement various feature extractors and algorithms to detect three different types of anomalies - local, community and global. They have used processes similar to clustering to find the nearest 1-hop neighbors as well as detect communities within graphs.

Local Evolutionary Outlier Detection: Both the above papers have proposed solutions to detect anomalies in static networks or single snapshots of networks in time. However, social networks as well as other types of networks like web graphs or sensors are often evolutionary and time-varying. Finding anomalies in such networks is often much more complicated. (Ji et al., 2013) have defined a new concept of CoreNets for each node that builds on the definition of EgoNets and SuperEgoNets. These are then used to compute certain features that can enable the detection of local and Deeply Hidden Outliers in both spatial and temporal dimensions.

Most networks today, especially social networks are weighted, dense graphs with evolutionary edges. For example, on Facebook, friends are constantly added or removed. On a sensor network, whether one sensor sends a signal to another at each time step is probabilistic. The signals themselves also have values. When one node malfunctions, either because a person on Facebook is spamming people or if a sensor is giving erroneous readings, they may not register as a global outlier, but it may still be an outlier within its neighborhood/community. On the other hand, it may not be detected as an anomaly at this snapshot, and we would need to consider its behavior in previous (and potentially later) time steps to see if it's behavior is an outlier.

Most of the work done yet focuses on one type of network instead of taking all these issues into consideration. In (Akoglu et al., 2010), the algorithm mainly focuses on weighted graphs. Also, the neighborhood used for anomaly detection is very limited, based on egonets (direct neighbors) of the nodes. The edge weights are only used in the outlier score calculation and not while computing the neighborhood of the node itself, as done in (Ji et al., 2013).

Another major problem with both the above algorithms is that they only consider spatial and not temporal network properties. For example, if a sensor readings are changing smoothly over time because of external conditions, even though that of the neighbors aren't, the above algorithms would classify it as anomalous. On the other hand, if a bunch of sensors in the neighborhood suddenly fail, a snapshot wouldn't be able to identify this without looking at the readings at the previous time step.

(Vengertsev & Thakkar, 2005) has also experimented with auto-encoders to detect anomalies based on reconstruction loss. The energy (or loss) of auto-encoders is a smooth curve and thus, finding the threshold for outliers requires some in-depth cross-validation, which may be the results of the poor performance they achieved despite being a superior model. (Ji et al., 2013) tries to solve a lot of these problems. They consider evolutionary networks to compute neighborhoods of the nodes and then calculate the scores based on both temporal and spatial variations. However, the closeness score used to calcu-

late the outlier is simplistic, based on just the sum of weights of the edges.

2.1. Our Contribution

Considering the various limitations of the works discussed above, we can try to combine them and extend them using various new feature extraction techniques and outlier scoring techniques. Firstly, we need a technique that works on evolutionary, time-varying, weighted multi-graphs with any kind of anomaly, either local or global. To do this, we need to augment the work of (Vengertsev & Thakkar, 2005) and (Akoglu et al., 2010) with time-varying features and scoring methods. We can consider the 'context' of each snapshot of the graph (previous time steps) to see if there are temporal spikes leading to anomalies. In cases where the analysis need not be performed in real-time, and we have data from the next few time steps, we can use these too as context.

We can also potentially improve the work in (Vengertsev & Thakkar, 2005) for better results by using Denoising Auto-Encoders to mitigate the problem caused by noise as discussed in the previous section. The features learned by the encoder can themselves be used in other scoring techniques to detect outliers instead of using the reconstruction loss. We can also try SVMs and Elliptic Envelopes instead of Isolation Forests as they are very different in terms of performance on different types of datasets.

If we consider (Ji et al., 2013) CoreNets are a good way of computing closeness of neighbors. Incremental Closeness can be used to see how the closeness between nodes has changed over time. This is particularly useful in sensor networks where signals are highly time-varying. However, the outlier scoring using these CoreNets can be improved by using better techniques like the scoring system in (Akoglu et al., 2010). Alternatively, the features from CoreNet can be fed into a Denoising AutoEncoder to learn the typical encoding and decoding of the nets. The encoded features can either directly be used in scoring or can be decoded to get the reconstruction loss.

3. Dataset

3.1. Intel Lab Real-world Dataset

As a benchmark dataset for testing and method comparison we have selected Intel Lab data set [13]. This data set is represented as time-varying weighted multi-attributed graph that corresponds to sensors work for period from February 28th and April 5th, 2004 with properties shown in below table Table 1:

Property	Value
Number of nodes |V|	54
Max. Number of Edges |E|	2916
Feature Vector D(t)	d=4
Total readings	2.3 million

Table 1. Properties of Intel lab Dataset.

The raw sensor data is given in 3 files:

1. Node Information: The node ids and x and y coordinates of each sensor is stored in a file. There are a total of 54 sensors and thus, 54 nodes in the graph. While the primary transmission and reception information is stored in the edges, the geographical location can be useful to detect/avoid certain anomalies.

2. Edge Information: The graph given is a fully connected graph with directed nodes from each node to every other node. However, the probabilities of a sensor transmitting to another sensor, and thus, the weight of the edges varies. Using this, we created different graphs, with edges between nodes conditioned on the probability being above a certain threshold. Graphs were thus created with edges formed between nodes with probability threshold 0.75, 0.6 and 0.5.

3. Sensor Readings: Sensor readings for humidity, temperature, voltage and light are provided for 65535 different time instances. These thus represent about 65,535 epochs * 54 nodes * 4 features = 14,155,560 total readings.

Data Modifications:
We created smaller datasets for testing the effectiveness of each algorithm by varying number of edges by varying the probability threshold as described above. The number of edges increase from 25 to 187 as we decrease probability threshold from 0.75 to 0.5
Another way of creating smaller datasets was to decrease the number of epochs. Thus, tests can be performed for all algorithms on graphs for sensor readings for the first 1000 and first 10000 epochs alongwith the full data with 65535 epochs.
While the given dataset has a lot of anomalies based on features of the node, there are no edge creation or deletion anomalies. However, the IcLEOD algorithm is well-known for finding anomalies when there are new edges added/removed and when the edge weights change. DNODA and CNA alongwith Isolation Forests can also detect these types of anomalies. To test these, we added a time-function, δ_t, to the

weight of an edge (src, dst) given by:

$$\delta_t(src, dst) = 1 + \sum_{f \in features} |src_{f,t} - dst_{f,t}|$$

The new weight of the edge, w' is given by

$$w'_t(src, dst) = prob(src, dst) * \delta_t(src, dst)$$

3.2. Synthetic Dataset

Although the Real World dataset acquired from Intel Lab is structured and ideal for anomaly detection, it does not have labels attached to the anomalies. Thus, barring manually labelling the anomalies, it is impossible to judge the raw performance of our algorithms on this dataset. Thus, we generated a synthetic dataset similar to the Intel Lab (IL) set but with anomalies inserted of our own.

The edge probability information was kept the same as the IL set. To generate synthetic sensor readings, we copy the readings for the first epoch from the IL set. For every subsequent epoch, the reading is changed by a small δ chosen uniformly at random from a small range ($< 0.5\%$ of the maximum range of that value). With probability p, an anomaly was inserted in a particular sensor reading of an epoch. This was done by picking any value uniformly at random from the entire maximum range for that feature.

By picking $p = 0.001$, there were 14042 anomalies generated in about 65535 readings each for 54 nodes. This labelled dataset was used to evaluate the performance of our algorithms based on accuracy, precision and recall. The thresholds for each algorithm were determined by using the ROC curves for them on the synthetic dataset. These thresholds were then used to identify anomalies in the IL dataset.

4. Problem Formulation

Given a graph $G = (V(t), E(t), w, D(t))$, where $V(t)$ is a set of n nodes at time t, $E(t) \subset V \times V$ is a set of m undirected edges, and $w : V \times V \in [0, 1]$ is a weight function such as $w(u, v) = 0$ if and only if $(u, v) \notin E$, $D(t) \in R^d$ is a row vector of time-dependent node attributes, d is number of attributes. With subscript $D_v(t)$ we denote vertex v which has attribute vector $D(t)$ and with superscript $D^i(t)$ we denote i-th component of the node attribute vector $D(t), i \in \{1, d\}$.

Definition (Type 1 Anomaly): For graph G a node v is called type 1 anomaly, if it has attributes $D_v(t)$ that are rare and differ from the majority of other node's attributes $D_u(t)$ for $u \in V \backslash v, t \in [t_0, t_0 + \delta)$, for small $\delta > 0$. This anomaly is referred to as global anomaly, Fig.1.a. This anomaly type does not take into account any network structure or temporal structure in the data. But as we mentioned earlier network data incorporates interdependence, therefore we study three additional pattern-based definitions of anomalies.

Definition (Type 2 Anomaly): For graph G a node v is called type 2 anomaly, if it has attributes $D_v(t)$ that are rare and differ from the majority of neighboring node's attributes $D_u(t)$, where $u \in N(v)$, where $N(v)$ are neighboring nodes of node v, $t \in [t_0, t_0 + \delta)$, for small $\delta > 0$. This anomaly is referred to as neighbor anomaly.

Definition (Type 3 Anomaly): For graph G a node v is called type 3 anomaly, if it has attributes $D_v(t)$ that are rare and differ from the majority of the same community node's attributes $D_u(t)$, $u \in C(v)$, where $C(v)$ are nodes from the same community node v, $t \in [t_0, t_0 + \delta)$, for small $\delta > 0$. By community, we mean a densely connected groups of "close" nodes in the graph. This anomaly is referred to as community anomaly.

Definition (Type 4 Anomaly): For graph G a node v is called type 4 anomaly, if it has attributes $D_v(t)$ that are rare and differ from the majority of the same nodes attributes at different time steps in the same time series, $D_v(t')$, $t \in [t_0,' - \delta] \cup [t' + \delta, T]$, for small $\delta > 0$. This anomaly is called temporal anomaly.

5. Theory

5.0.1. Important Graph Properties

Following are some important graph properties that we will be using to extract features for anomaly detection.

Definition 1 (Egonet): Given a node $v \in V(t)$, the egonet of v is defined as $egonet(v) = \{v\} \cup \{u | u \in V(t), e_{vu} \in E(t)\}$. Where e_{vu} is the edge between v and u.

Definition 2 (Super-egonet): Given a node $v \in V$, the superegonet of v is defined as $super - egonet(v) = \{ego(v)\} \cup \{ego(u) | u \in V(t), e_{vu} \in E(t)\}$.

Obviously, these two concepts are very simple in obtaining the local substructure: they just regard 1-hop neighbors(egonet) or neighbors within 2-hop(superegonet) as the ego's closest neighbors. However, they will encounter problems when dealing with weighted graphs. As in the case of a friendship network with edge-weights representing interactions between friends, one is likely to be closer to his intimate friend's intimate friend instead of his nodding acquaintances. The concept of egonet focuses only on structural connection but ignores the power of closeness

transmission. Therefore, it requires a forceful measurement considering both connectivity and closeness. First, we propose the following two notions to assess the closeness between ego and its neighbors. We call the node of interest core to differentiate it from egonet.

Definition 3 (Closeness related to the core): Let node v_0 be core, $v_0 \in V(t)$. For $v_l \in V$, we assume that there are d paths connecting v_0 and v_l . The jth path (l in length) passes through nodes $\{v_0, v_1, v_2, \dots, v_l\}$ in sequence, where $1 \leq j \leq d$. Then the closeness between v_0 and v_l is defined as:

$$Closeness(v_0, v_l) = \max_{1 \leq j \leq d} \Pi_{i=0}^{l-1} \frac{w_{v_i v_{i-1}}}{w_{v_i}}$$

Where $w_{v_i v_{i+1}}$ is the weight of the edge between v_i and v_{i+1}, and w_{v_i} is the sum of the weights of the edges connected to node v_i . Obviously, $\forall v_l \in V(t)$, $Closeness(v_0, v_l) \in [0, 1]$. The higher the value, the more intimate the relation is. It is possible that a node directly connected with the core owns a smaller closeness. In the case that two (or more) identical values of closeness are obtained from two (or more) different paths, to avoid closeness drift, we prefer the path that includes the edge directly connecting the core with maximum weight.

Definition 4 (k-closeness of the core): Let node v_0 be core, $v_0 \in V(t)$. For $\forall k > 0$, the k-closeness of the core, denoted as k-closeness(v_0), is defined as :
(i) For at least k nodes $v_p \in \{V(t) \backslash v_0\}$, it holds that $Closeness(v_0, v_p) \geq k - closeness(v_0)$,and
(ii) For at most k-1 nodes $v_p \in \{V(t) \backslash v_0\}$, it holds that $Closeness(v_0, v_p) > k - closeness(v_0)$. Different with the concepts of Egonet and Super-egonet, this definition considers the top-k "closest" neighbors of the core only based on closeness transmission, instead of linking relationships. In this definition, the "closest" neighbors are those nodes with larger value of closeness, rather than directly connecting with the core.

Definition 5 (k-closeness neighborhood of the core): Given the k-closeness of core v_0, the k-closeness neighborhood of v0 contains every node whose closeness related to v_0 is not smaller than the $k - closeness(v_0)$. Formally, $N_k(v_0) = \{v_p \in V(t) \backslash v_0 | Closeness(v_0, v_p) \geq k - closeness(v_0)\}$. As mentioned above, egonet concerns only the nodes directly connected with the node of interest, while the closeness measurement (Def. 3-5) mainly consider closeness transmission. The former completely ignores the edge-weight information, similarly, the latter ignores the risk that the reliability may reduce after successive transmissions. Thus, for the purpose of discovering the local context for the core, we propose a notion named Corenet that balances the topology structure and the closeness transmission.

5.0.2. Feature Extraction

We study four different kind of features and compare them for the task of anomaly detection

1. **Direct Neighbour Outlier Detection Algorithm (DNODA)**: DNODA is an algorithm that considers use of the direct neighbours $u \in N_k(v)$ of a given node $v \in V(t)$. A DNODA outlier feature is calculated as below. Intuitively the feature is directly proportional to the distance of v from its direct neighbours using the feature vector $D_v(t)$, refer to score as $DNODA(v_0) \in R^d$. Hence an aberrant variation of any node would indicate an anomaly. Formally we can calculate this using the following equation

$$DNODA(v_0) = D_{v_0}(t) - \frac{\sum_{t \in N_k(v_0)} D_u(t)}{k}$$

   To account for the temporal dimension in our problem, we also consider the readings of the same node at time-steps $t-1$ and $t+1$ as neighbors. Thus, the formula now changes to

$$DNODA(v_0) = D_{v_0}(t) - \frac{D_{v_0}(t-1) + D_{v_0}(t+1)}{k+2} - \frac{\sum_{t \in N_k(v_0)} D_u(t)}{k}$$

2. **Community Neighbor Algorithm (CNA)**: Communities were detected in the graph using Spectral Clustering on the network. Spectral clustering (Buitinck et al., 2013) allows a precomputed distance matrix to be used for the clustering function. Thus, the matrix for the sensors consisting of the probabilities of the corresponding edges was used as an affinity metric. When the communities are identified we calculate the score within community as

$$CNA(v_0) = D_{v_0}(t) - \frac{\sum_{t \in C(v_0)} D_u(t)}{k+2} - \frac{D_{v_0}(t-1) + D_{v_0}(t+1)}{k+2}$$

   where $C(v_0)$ is the community as defined by the MCL algorithm.
   Again, we add the features for the previous and following time step to the ones in the community to detect outliers.

3. **Denoising Autoencoders**: An autoencoder (Vincent et al., 2008) takes an input $\mathbf{x} \in [0,1]^d$ and first maps it (with an encoder) to a hidden representation $\mathbf{y} \in [0,1]^{d'}$ through a deterministic mapping, eg:-

$$\mathbf{y} = s(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1}$$

Where s is a non-linearity such as the sigmoid. The latent representation $\mathbf{y}$, or code is then mapped back (with a decoder) into a reconstruction $\mathbf{z}$ of the same shape as $\mathbf{x}$. The mapping happens through a similar transformation, e.g.:

$$\mathbf{z} = s(\mathbf{W}'\mathbf{y} + \mathbf{b}') \tag{2}$$

The reconstruction error can be measured in many ways, depending on the appropriate distributional assumptions on the input given the code. The traditional squared error $L(\mathbf{x}\mathbf{z}) = ||\mathbf{x} - \mathbf{z}||^2$, can be used. If the input is interpreted as either bit vectors or vectors of bit probabilities, cross-entropy of the reconstruction. If the input is interpreted as either bit vectors or vectors of bit probabilities, cross-entropy of the reconstruction can be used:

$$L_H(\mathbf{x}, \mathbf{z}) = -\sum_{k=1}^{d} [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)] \tag{3}$$

If there is no constraint besides minimizing the reconstruction error, one might expect an autoencoder with n inputs and an encoding of dimension n (or greater) to learn the identity function, merely mapping an input to its copy. Such an autoencoder would not differentiate test examples (from the training distribution) from other input configurations.

The idea behind denoising autoencoders is simple. In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, we train the autoencoder to reconstruct the input from a corrupted version of it. The denoising auto-encoder is a stochastic version of the auto-encoder. Intuitively, a denoising auto-encoder does two things: try to encode the input (preserve the information about the input), and try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder. The latter can only be done by capturing the statistical dependencies between the inputs.

4. **Corenet**: Given the k-closeness of core, $k - closeness(v_0)$, the Corenet of v_0 contains nodes that satisfy the conditions: (i) the closeness related to v_0 is not smaller than the $k - closeness(v_0)$, and (ii) they are in the $super - egonet$ of v_0. Formally, $v_p \in super - egonet(v_0) \backslash v_0$, $Corenet(v_0)$ is defined as:

$$Corenet(v_0) = \begin{cases} super - egonet(v_0), \\ \text{if } \min_{v_p} Closeness(v_0, v_p) \geq \\ \qquad k - closeness(v_0) \\ \\ N_k(v_0), \qquad \text{others} \end{cases}$$

So far, we have defined corenet as the local context of the core, which fully takes closeness transmission into account and avoids meaningless excessive transmissions by imposing a structural restriction. It is obvious that only the nodes in $super - egonet(v_0)$ need to be calculated closeness related to the core and the maximum size of corenet is the number of the core's neighbors within 2-hop.

5.0.3. Spatio-Temporal Anamoly Detection

Once the features have been extracted, we use each of them and pass them through the following three algorithms.

1. **Incremental Local Evolutional Outlier Detection (IcLEOD)**:- Before we present the particular measuring function, we first analyze the signs that a node is evolving abnormally. Consider we have two snapshots G_{t-1} and G_t , and the node of interest is v, there are two major signs to show that v is likely to be a outlier:
(1) The members of $Corenet(v)$ in G_{t-1} no longer belong to $Corenet(v)$ or their closeness related to v is getting weaker from G_{t-1} to G_t;
(2) The new members added to $Corenet_t(v)$ have clear distinction with the former members, moreover, their closeness related to v can be unexpected high. These two anomalous indication can be measured by Score 1 and Score 2 respectively, and the outlying score is the sum.

Let, $Corenet_{t-1}(v)$ and $Corenet_t(v)$ represent the Corenets of node v in G_{t-1} and G_t respectively. We denote the intersection of $Corenet_{t-1}(v)$ and $Corenet_t(v)$ except v as C_{old}, which is the set of old neighbors of node v. The elements of $Corenet_{t-1}(v) \backslash C_{old}$ are the neighbors removed from $Corenet(v)$ at time t, denoted as $C_{removed}$.

The elements of $Corenet_t(v) \backslash C_{old}$ are new neighbors of v, denoted as C_{new}. The outlying score of node v is defined as:

$$OutlyingScore(v) = \sum_{v_r \in C_{removed}} closeness_{t-1}(v_r, v) + \sum_{v_i \in C_{old}} [closeness_{t-1}(v_i, v) - closeness_t(v_i, v)] + \sum_{v_j \in C_{new}, v_i \in c_{old}} [1 - \frac{w_{v_i v_j}}{w_{v_j}} \times closeness_t(v_j, v)]$$

Where $w_{v_i v_j}$ is the weight of edge between v_i and v_j , w_{v_j} is the sum of the weights of the edges connected to v_j . The sum of former summation terms is Score 1, which measures outlying degree caused by situation (1). Similarly, the third summation term represents Score 2, which measures outlying degree caused by new neighbors in situation (2).

2. **Measuring Reconstruction error** $L_H(x, z)$: Using autoencoders, anomalies can be detected since they will be different from learned "normal" patterns, and therefore would have a higher reconstruction error.

$$L_H(\mathbf{x}, \mathbf{z}) = -\sum_{k=1}^{d} [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)]$$

3. **Isolation Forest**: One efficient way of performing outlier detection in high-dimensional datasets is to use random forests (Pedregosa et al., 2011). The algorithm in Isolation Forest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a node is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is a measure of abnormality and our decision function. Thus, random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular nodes, they are highly likely to be anomalies.

6. Experiments & Results

We conducted various experiments for different datasets, that qualitatively give us proof that the implementation of our algorithms gives the correct result.

	Type 1	Type 2	Type 3	Type 4
DNODA	✗	✓	✗	✓
DNODA + Isolation Forest	✗	✓	✗	✓
CNA	✗	✗	✓	✓
CNA + Isolation Forest	✗	✗	✓	✓
lcLEOD	✓	✓	✓	✓
Denoising Auto-Encoders	✓	✓	✓	✓

Figure 1. A table describing what algorithms are applicable for the detection of what kind of anamolies.

In all, we conducted 12 separate experiments to discover all the types of anamolies i.e. from Type 1 to Type 4. Along the way we also compare different algorithms for anamoly detection and different feature extraction techniques. There are 6 techniques in total and 2 different datasets (the intel data set and the synthetic dataset described above). Here we present a brief outline of all the 6 experiments that were conducted. All the details about the feature extractors and the anomaly detection algorithms can be found in previous sections of this report.

1. DNODA feature extractor with DNODA score as the anomaly detection measure: This is useful for detection of Type 2. Our definition of DNODA features also takes into account the features of the neighbors at time T-1 and T+1. Thus this enables the detection of Type 4 anomalies.

2. CNA feature extractor with CNA score as anomaly detection measure: This is useful for detecting Type 3 anomalies, i.e. anomalies within a community. Our definition of CNA features is slightly modified to include the community features at time instances T-1 and T+1, thus this also enabled this model to detect Type 4 anomalies.

3. DNODA feature extractor with isolation forest as anomaly detection algorithm: Same as experiment (1). This is conducted to compare the performance of DNODA score with the isolation forest.

4. CNA feature extractor with isolation forest as the the anomaly detection algorithm: Same as experiment (2). We can compare the performance of

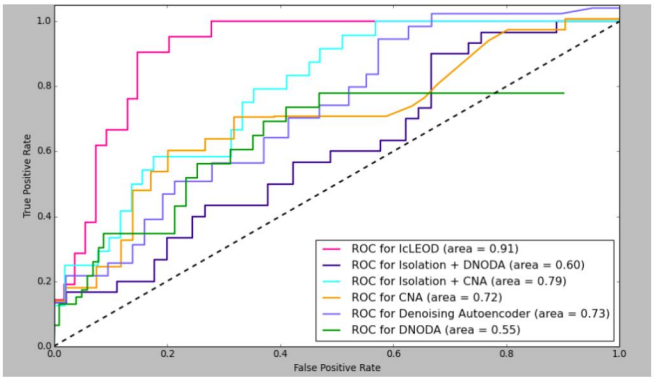

Figure 2. A graph comparing the quantitative performance of various algorithms on the synthetic dataset.

CNA score and the isolation forest algorithm.

5. Corenet feature extractor with IcLEOD algorithm for anomaly detection: This algorithm is designed to predict Type 4 anomalies i.e. the temporal anomalies accurately.

6. Denoising Autoencoders as the feature extractors with reconstruction loss as the anomaly detection measure: The denoising autoencoder can learn the probability distribution of any kind data that is presented to it. In our experiments we make the autoencoder learn a probability distribution of sensor reading as compared to their surrounding and also node features at time instances T-1 and T+1. This is done to have a fair comparison between the other algorithms and the denoising autoencoder. In its current state this experiment will also detect Type 2 and Type 4 anomalies.

6.1. Synthetic Dataset

True Positive Rates (TPR), False Positive Rates (FPR), Receiver operating characteristic (ROC) curves and Area under ROC curves(AUC) scores are used to measure the performance of the algorithms on the labelled data. Table 2 outlines the results obtained for all 6 experiments. Figure 2 shows the ROC's of each of the algorithm. As we can see, IcLEOD, being an algorithm specifically tailored for Temporal Anomaly detection, performs the best. Isolation Forests in combination with DNODA and CNA perform better than their vanilla counterparts. CNA versions perform better than the DNODA versions since the graph is highly clustered and CNA considers more 'neighbours' than DNODA does. Auto-encoders does decently well.

Algorithm	ROC AUC
IcLEOD	0.91
Isolation Forest with CNA features	0.79
MSE for denoising Autoencoder	0.73
CNA scores	0.72
Isolation Forest with DNODA features	0.60
DNODA scores	0.55

Table 2. Quantitative results for various algorithms on the synthetic dataset

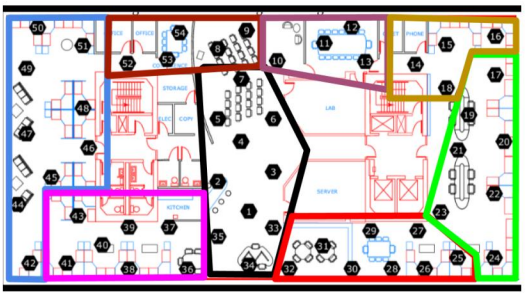

Figure 3. Sensor Boundaries identified by the Spectral Clustering on edges of Intel Lab dataset. Different colored boxes represent the different clusters. Boundaries closely represent the geographical position of sensor nodes

6.2. Real World Dataset

Although it is not possible to quantitatively evaluate the results for the real-world dataset, we can visualize the performance of various algorithms and analyze the performance qualitatively.

As we can see from Figures 4 to 7, according to the readings of various sensors displayed, mote ids $\{12, 31, 50\}$ seem to be outliers. Out of these, the mote ids 12 and 31 classify as a Type 2 , 3 anomaly and mote id 50 as a Type 4 anomaly. This is what is predicted by the various algorithms which can be seen in the figures 8 to 10.

We can clearly see that DNODA and CNA are good at discovering Type 2 and Type 3 anomalies but cannot detect the Type 4 anomalies (for mote 50) that are easily discovered by IcLEOD (due to the spike for the reading of the light sensors).

7. Conclusion

1. Normal spatial anomaly detection algorithms with few modifications can detect temporal anomalies in evolutionary networks.

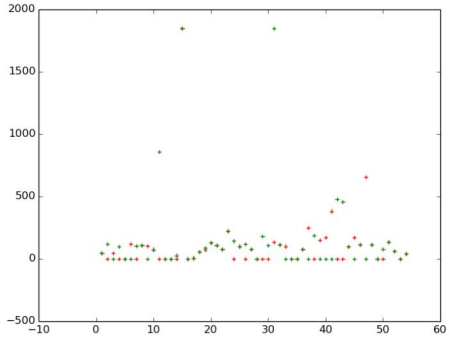

Figure 4. The light reading for all the 54 sensors at two time steps (red and green). As we can see here that there is a huge spike between the two different time steps for mote 50. Classifying this as a Type 4 Anomaly. Also, we can see the spikes for the green scatter plot for node 12 and 31, classifying it as a Type 2 and Type 3 anamoly.

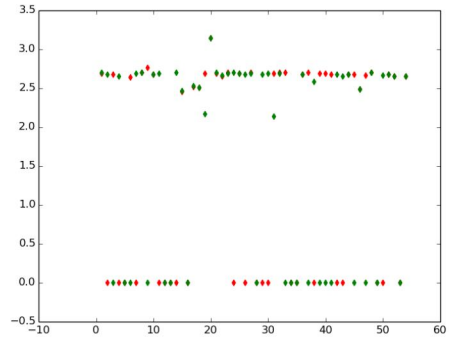

Figure 5. The voltage reading for all the 54 sensors at two time steps (red and green). No clear spikes appear in these readings.

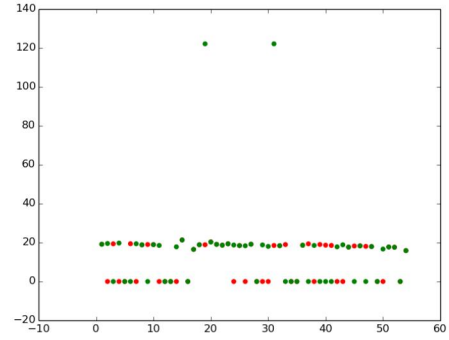

Figure 6. The temperature reading for all the 54 sensors at two time steps (red and green). Here the spikes occur within the same time step for nodes as compared to their neighbors, thus this may result in Type 2 and Type 3 anamolies.

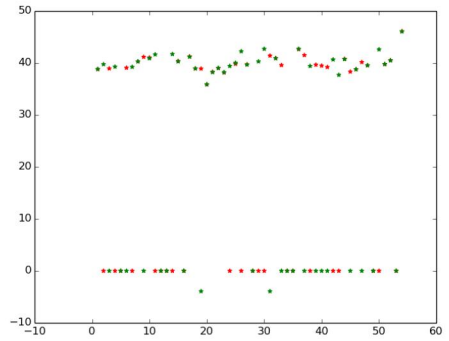

Figure 7. The humidity reading for all the 54 sensors at two time steps (red and green). No clear spikes are observed.

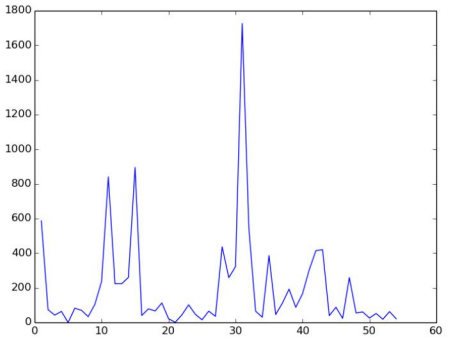

Figure 8. DNODA scores for all 54 motes for a particular epoch. As seen, the node 31 (Type 2 anomaly) has an aggregate reading that is very different from it's neighbours. DNODA also captures anomalies in nodes 12 and 15.

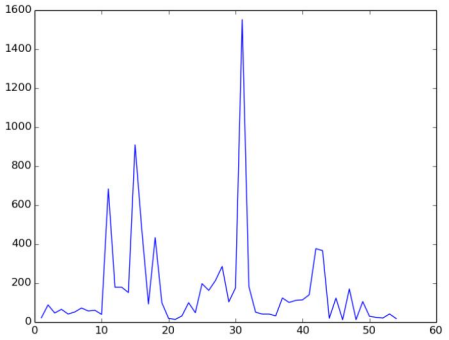

Figure 9. CNA scores for all 54 motes for a particular epoch. As seen, the nodes 12, 15 and 31 are also captured here. CNA also manages to capture the anomaly in node 18 (Type 3 anomaly) that was missed by DNODA.

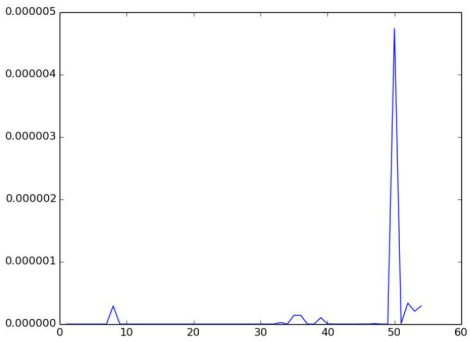

Figure 10. IcLEOD scores for all 54 motes for a particular epoch. IcLEOD manages to capture the Type 4 anomaly in node 50 that the other algorithms miss

2. IcLEOD performs very well in anomaly detection tasks with temporal anomalies.

3. Denoising Auto-encoders can potentially perform well if the data set is huge and has a lot of features. For data with fewer points or features, they fail due to lack of learning and overfitting.

4. Isolation Forests boost the performance of Anomaly Detection Algorithms.

8. Division of Work

The theoretical part was done equally by both team members. The implementation was divided as follows:

1. Dataset Creation and Curation: Juhi

2. DNODA: Juhi

3. CNA: Juhi

4. Isolation Forests: Juhi

5. IcLEOD: Kratarth

6. Denoising Auto-encoders: Kratarth

7. Results evaluation: Kratarth

References

Akoglu, Leman, McGlohon, Mary, and Faloutsos, Christos. Oddball: Spotting anomalies in weighted graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 410–421. Springer, 2010.

Buitinck, Lars, Louppe, Gilles, Blondel, Mathieu, Pedregosa, Fabian, Mueller, Andreas, Grisel, Olivier, Niculae, Vlad, Prettenhofer, Peter, Gramfort, Alexandre, Grobler, Jaques, Layton, Robert, VanderPlas, Jake, Joly, Arnaud, Holt, Brian, and Varoquaux, Gaël. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

Ji, Tengfei, Yang, Dongqing, and Gao, Jun. Incremental local evolutionary outlier detection for dynamic social networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 1–15. Springer, 2013.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Vengertsev, Dmitry and Thakkar, Hemal. Anomaly detection in graph: Unsupervised learning, graph-based features and deep architecture. 2005.

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. URL http://doi.acm.org/10.1145/1390156.1390294.