DEPARTMENT OF
STATISTICS

Anomaly detection in networks

Gesine Reinert

Department of Statistics
University of Oxford

Women in Networks, Leeds

Joint work with

Andrew Elliott and Mihai Cucuringu (Alan Turing Institute),

Milton Martinez Luaces and Paul Reidy (Accenture);

`https://arxiv.org/abs/1901.00402`

Motivation: Financial Fraud

Financial fraud losses have been estimated to at least GBP 768 million in the UK alone in 2015.

Financial fraud is a global challenge.

Here we are motivated by anti-money laundering.

We represent financial transaction data as a network, with accounts as nodes and transactions as directed edges, weighted by the transaction amount.

Known and unknown anomalies

Some fraudulent behaviour should be reflected in such a network, for example as

- long paths of large transaction amounts;
- rings of large payments;
- cliques of accounts that send money to each other.

The typical anomaly may be a new anomaly!

Aim: to detect novel anomalies while using labelled data if available.

Challenge: fast method to pick up small anomalies in large networks

Idea

Combine

- community detection to decompose the network into smaller units to deal with large networks;
- network comparison based on small subgraphs and spectral features to pick up small anomalies;
- and use these, as well as standard network summary statistics, as input features for random forests to obtain a fast method. In total we use 140 features.

To assess expected behaviour we use Monte Carlo tests,

- randomise the edges of the networks while keeping the degrees fixed (configuration model), and
- randomly shuffle the weights of the edges.

Data ?

Due to lack of financial transaction data sets with labelled fraudulent transactions, we resort to two synthetic network data sets:

1. a set of synthetic networks which we generated, based on a directed Bernoulli random graph with random weights and randomly planted anomalies;
2. a set of synthetic networks which the Accenture team generated (in discussions with the Turing team) and in which the anomalies were revealed to the Turing team only after their analysis.

A preview of our results

- Our method outperforms Oddball, a widely used method for anomaly detection in networks by Akoglu *et. al.* (2010);
- a simple sum of feature scores does as well as the random forest, give or take;
- the top 2 % of flagged anomalies contained on average over 90 % of the planted anomalies.

The data

Labelled data

To develop and measure the success of our methodology, we use two network models with known embedded anomalies.

The first model guided our method development stage with an eye towards the underlying ground truth.

The second model was developed by the Accenture team, following on from discussions with the Turing team; a network which was generated from this model was shared with the Turing team for performing anomaly detection.

The Turing team was oblivious to the model details; the details were only revealed to the Turing team when generating the final set of 100 test networks for inclusion in the publication.

Synthetic data set 1: weighted ER network

Our synthetic model is a weighted directed ER random graph model, in which small structures with unusually large weights are planted at random.

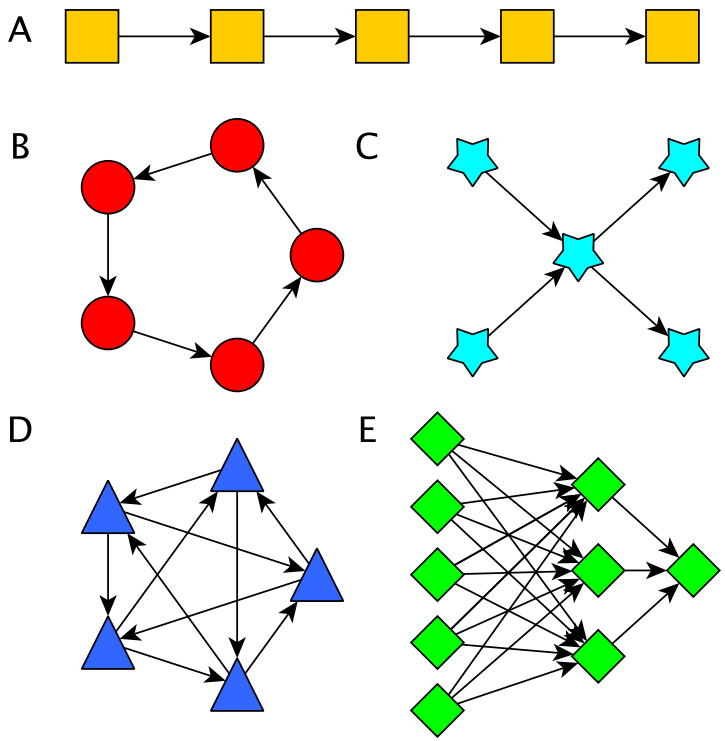
The edge weights are drawn independently and uniformly from the interval $(0, 1)$. Therefore, the distribution of the non-anomalous edges from node i to node j is that of

$$W_{ij} \sim \text{Bern}(p)\text{Uni}(0, 1),$$

where $\text{Bern}(p)$ is a Bernoulli trial with probability p, and $\text{Uni}(0, 1)$ is a uniform random draw from the interval $(0, 1)$ which is independent of the network and of all other draws.

Planted anomalies

We embed a random number of random sized anomalous structures of five different types at random locations throughout the network.



How anomalies are planted

1. Generate a directed (n, p) ER network
2. Add independent uniform$(0, 1)$ weight to each edge
3. Draw the number of anomalies from $[5, \ldots, 20]$
4. For each anomaly select the type and the number of nodes from $[5, \ldots, 20]$
5. Place the anomaly uniformly at random.
6. Place independent uniform$(w, 1)$ weights on the anomalous edges, where w is chosen such that the expected number of anomalies is less than 1 without planted anomalies.

Synthetic data set 2: Accenture network

The Turing team was given access to several samples from the model and they were informed that the anomalies would include heavy edges.

Due to lack of other information about the network structure, and inspired by a previous Accenture model, the Turing team used a configuration model as null model for the Monte Carlo tests.

During the development stage, the Turing team submitted a list of flagged anomalies to the internal Accenture team in order to assess the performance of the method.

The construction of the Accenture network

The number of nodes is fixed to $55,000$, and each node is assigned an in-degree and an out-degree.

The assignment is made by rounding down random draws from a normal distribution

- for the in-degree: $\mathcal{N}(21, 9)$
- for the out-degree: $\mathcal{N}(19, 4)$

If the sum of the in-degrees does not agree with the sum of the out-degrees, then the larger sum is reduced by deleting edges at random, until the totals match.

Each node is assigned an in-degree and an out-degree, and a corresponding number of stubs are created.

The stubs are then randomised and matched.

In the case of self loops, the algorithm randomly selects a set of edges of same length as the number of random edges, and swaps the 'sender' node of a self loop with the sender node of the randomly selected edge.

Multi-edges are permitted.

This process repeats until all stubs are matched.

Edge weights are then drawn from $\mathcal{N}(1000, 200)$, with all edge weights being positive.

To this network, the following anomalies were added:

- a clique of size 8, with exactly one edge between each pair of nodes,
- a similar clique of size 12,
- a ring of size 4, with constant weights,
- a similar ring of size 10,
- a heavy path of length 5, sampled from paths that already exist in the network and
- a similar heavy path of length 10.

The weights on the heavy edges are drawn as follows: draw a number uniformly at random from the interval $(0.99, 0.999990)$, and use as weight the value that would be required to obtain this number as percentile from $\mathcal{N}(1000, 200)$.

The method

Module 1: basic statistics, community detection, and paths

The first module

1. performs some basic tests for deviations of summary statistics from the null model, and converts the resulting p-values into scores which are used as features later;
2. divides the network into communities;
3. adds a path finding method to ameliorate the effect of splitting the network into communities.

1. Basic statistics

First we assess deviations from the configuration model for some basic summary statistics for the weights of the network.

The deviations are assessed by comparing to a configuration model with shuffled weights.

For example the geometric average of the weights of edges adjacent to a node is standardised by taking away the empirical mean and dividing by the empirical standard deviation, to give a z-score.

From this z-score we construct a feature as follows. If the p-value of the standard normal distribution is below 0.05, then we report the z-score, and otherwise we report 0.

2. Community Detection

Observing a small anomaly is difficult on a large network, and hence we divide the large network into smaller subnetworks.

A standard method for community detection is the Louvain based modularity optimisation, and we use this method here.

In this process we would like to avoid cutting through long heavy paths.

Hence we use an augmentation trick: we add extra edges along paths of length 2 when each edge in this path has a weight which exceeds the 99th percentile of the weights.

Features from community detection

For a community C with density ρ_C, geometric average of the weights γ_C, and size n_C we use

1. statistics based on the density of a community:
   - ρ_C / (density of the full network);
   - ρ_C / [n_C (density of the full network)];
   - use the Monte Carlo p-value for testing whether the density is typical (vs larger) compared to the configuration model;
2. statistics based on the edge weights:
   - γ_C/(geometric average of the weights across the network);
   - γ_C/[n_C(geometric average of the weights across the network)].

3. PathFinder

The separation of the network into communities can split long paths.

Hence we include a separate feature for finding long paths.

We start with paths of length 3 and extend them greedily until they have reached a length of 20 (if possible).

Then a score is assigned as before - compare to a configuration model with randomised weights, assign p-value, if below 0.05, we convert it into a z-score using the inverse standard normal c.d.f., otherwise we set it to 0.

Module 2: Features from network comparisons

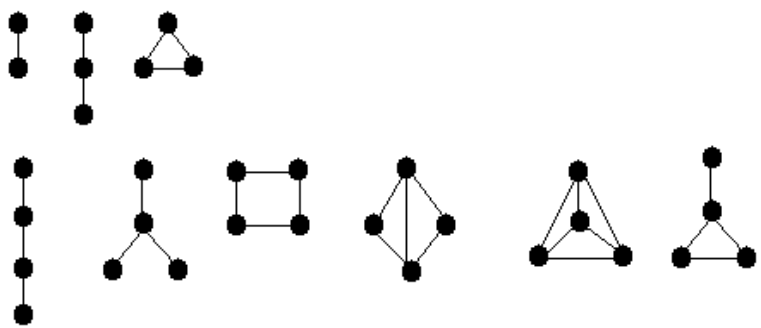
This module includes

1. the scores obtained from the community detection step;
2. node scores obtained from network comparison using *NetEMD*;
3. node scores obtained from spectral localisation statistics.

1. Network comparison using *NetEMD*

To capture small local anomalies we can use small connected graphs as unit for network comparison.

Here are small connected graphs on 2, 3 and 4 vertices:



There are 6 different graphs on 4 vertices.

Compare the count distributions for the two networks.

Intuition: the shape of the count/degree distributions is indicative for the network generating mechanism.

The underlying shape of a distribution should be invariant under translations and rescalings.

An example is given by the family of normal distributions.

The Earth Mover Distance

The Earth Mover Distance (EMD, 1-Wasserstein distance) between two univariate distributions with c.d.f. F and G is given by

$$EMD(F, G) = \int_{-\infty}^{\infty} |F(x) - G(x)| \, \mathrm{d}x.$$

If the distributions have and non-zero, finite variances $\sigma^2(F)$ and $\sigma^2(G)$, we define

$$EMD^*(F, G) = \inf_{c \in \mathbb{R}} EMD\left(F\left(\frac{\cdot}{\sigma(F)} + c \right), G\left(\frac{\cdot}{\sigma(G)} \right) \right).$$

EMD for networks: *NetEMD*

For two networks G and G' and given network feature t (small graph counts), we define the corresponding $NetEmd_t$ measure by

$$NetEmd_t(G, G') = EMD^*(F_t(G), F_t(G')),$$

where $F_t(G)$ and $F_t(G')$ are the c.d.f.'s of t on G and G' respectively (*Wegner et al, 2018*).

For a set $S = \{t_1, t_2, ..., t_m\}$ of network properties, the corresponding measure $NetEMD_S$ is

$$NetEMD_S(G, G') = \sum_{j=1}^{m} NetEmd_{t_j}(G, G').$$

Example: degree distribution

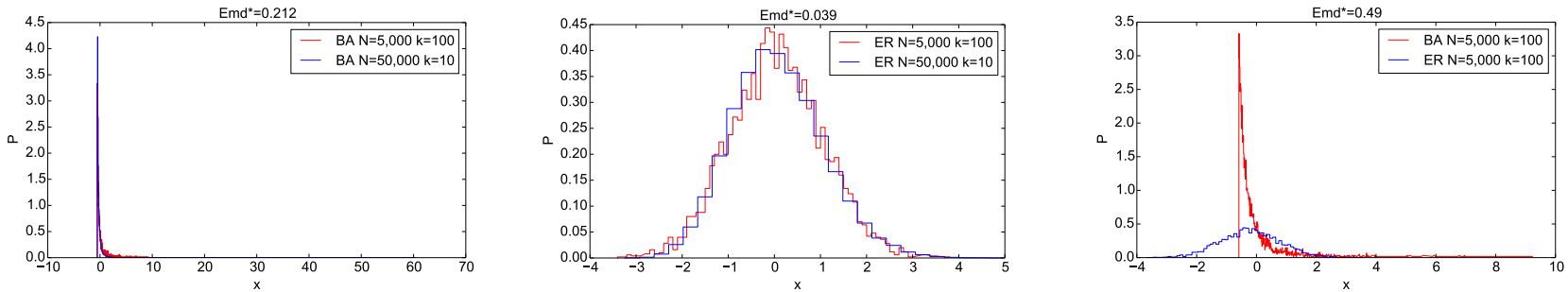


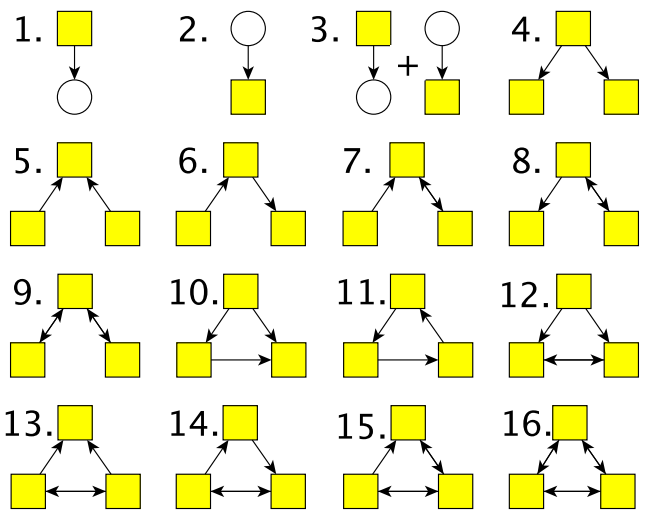
Figure: Plots of rescaled and translated degree distributions for BA and ER models with N vertices and average degree k:
a) BA $N = 5,000$, $k = 100$ vs BA $N = 50,000$, $k = 10$.
b) ER $N = 5,000$, $k = 100$ vs ER $N = 50,000$, $k = 10$.
c) BA $N = 5,000$, $k = 100$ vs ER $N = 5,000$, $k = 100$.

Implementation available at
`github.com/alan-turing-institute/network-comparison`

NetEMD features

Here we use the empirical distributions of

- in-degree, out-degree, and degree (sum of weights, also called *strength*);
- weighted directed subgraphs of size 3;
- the top 5 eigenvectors in the augmented network.



Again, for each statistic we take the Monte Carlo p-value of the *NetEMD* value compared to a configuration network with randomised weights.

For structures that are significantly different from the expectation under randomised weights, we assign a score to those nodes which have too large values or too small values.

Otherwise the score of the node is set to 0.

2. Spectral localisation statistics

Spectral localisation in networks is the phenomenon that a large amount of the mass of an eigenvector is placed on a small number of nodes.

In many cases, such nodes are different in some way compare to the remainder of nodes, and hence they may be good candidates for anomaly detection.

Laplacians

Let W be a weighted (possibly directed) adjacency matrix with non-negative entries, and let

$$W^s = W + W^T.$$

The Laplacians are defined as

$$\begin{aligned} \text{combinatorial Laplacian:} \quad & L_{Comb} = D - W^{(s)}, \\ \text{random-walk Laplacian:} \quad & L_{RW} = D^{-1} W^{(s)}, \end{aligned}$$

where D is the matrix with the degree of the symmetrised adjacency matrix of each node on the diagonal.

Eigenvectors are always normalised so that Euclidean norm 1.

We look for localisation in each of the communities separately, in the augmented network.

To this end, we look for localisation in the top 20 and bottom 20 non-trivial eigenvectors of the symmetrised adjacency matrix W^s, and the top 20 non-trivial eigenvectors of the combinatorial Laplacian and random-walk Laplacian of the augmented subnetwork corresponding to each community.

Whenever the network has fewer than 20 nodes, we consider as many eigenvectors as possible, and if there are ties (i.e., eigenvalues with multiplicity) we break ties randomly.

We compare the values with what we would expect at random.

In vectors with localisation, we assign a score to the nodes which carry most of the weights.

Module 3: Classification based on the features

Each of the network statistics described thus far produces a score.

One way to combine the scores is simply to sum them - we call this the Feature sum.

Following *Savage et al. (2016)*, we also combine the features using a random forest.

The training data for the Random forest are generated under our generating model, with $n = 10,000$ and the parameter sets (p, w) with $p \in \{0.001, 0.002, \ldots, 0.02\}$ and $w \in \{0.99, 0.991, \ldots, 1\}$ such that each anomaly has an expectation less than 1 in the network without planted anomalies; this yields 27 parameter sets.

For each parameter set we construct 100 networks and split them, with the first 70% as the training set and the remaining 30% as the test set. We run the entire pipeline on each of the 2700 networks.

We then perform feature selection, prioritising features that perform well across the ensemble of parameter regimes.

Finally, we combine the data from each of the parameter regimes, and fit a random forest on the reduced feature set to produce our final score.

Feature selection

We fit a random forest to the training data for each given parameter set using the scikit-learn `RandomForestRegressor` with default parameters in version 0.19.2 and use the feature importance vector from scikit-learn to construct a ranking.

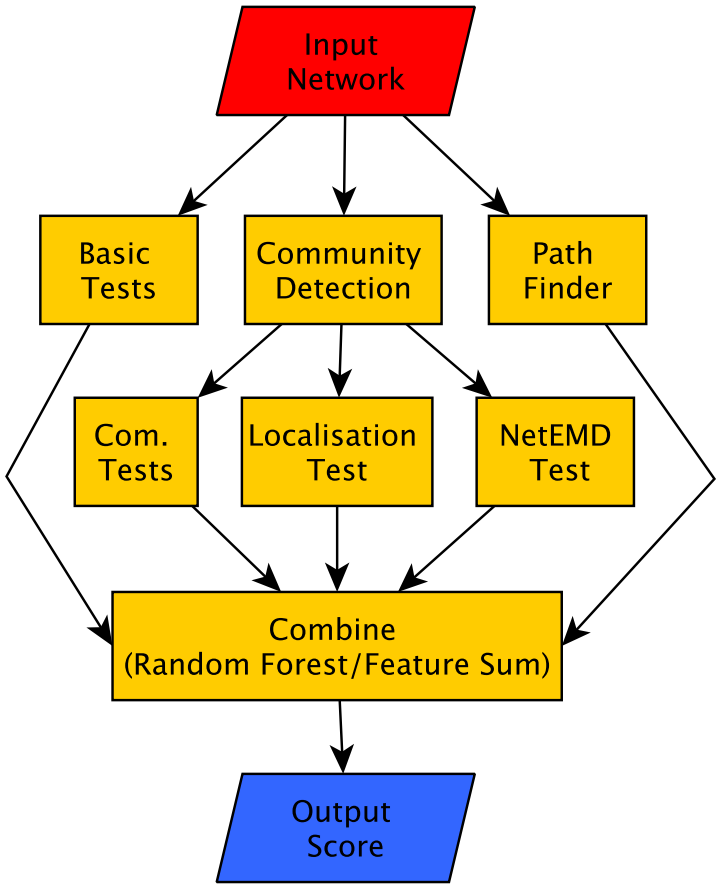
The average rank of each feature is

$$\text{FeatureRank}_i^{(\text{All})} = \frac{1}{27} \sum_{p,w} \text{FeatureRank}_i^{(p,w)},$$

where $\text{Feature Rank}_i^{(p,w)}$ is the rank of feature i in parameter set (p, w).

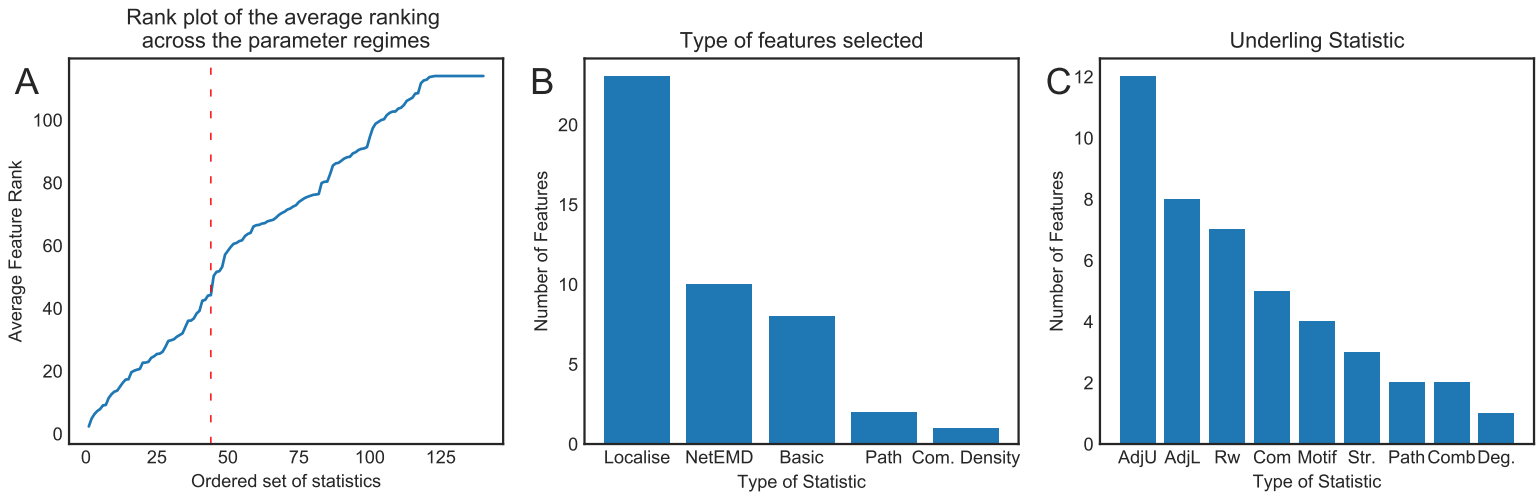
We select an appropriate cut-off by looking for a large jump in average rank in the rank plot of $\text{Feature Rank}_i^{(\text{All})}$.

The pipeline

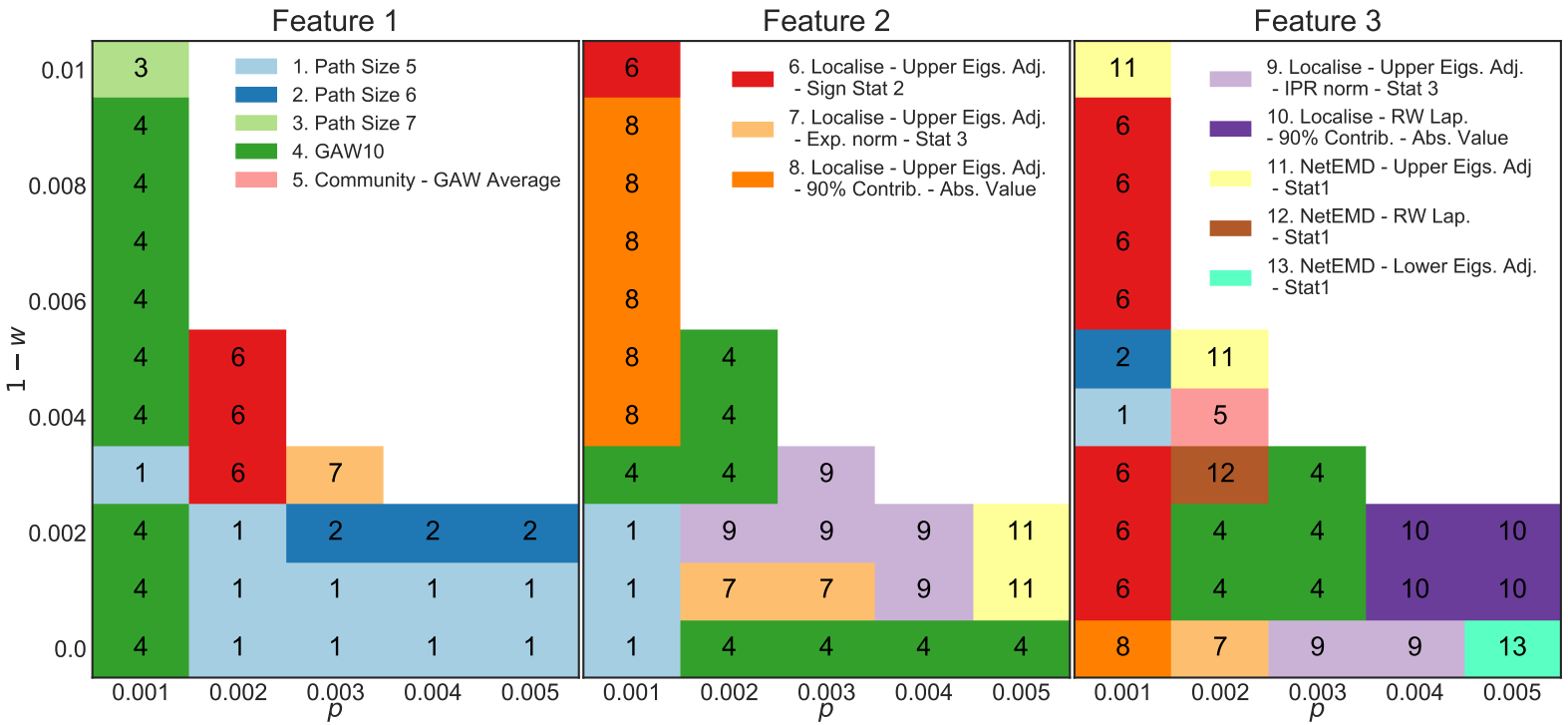


Results

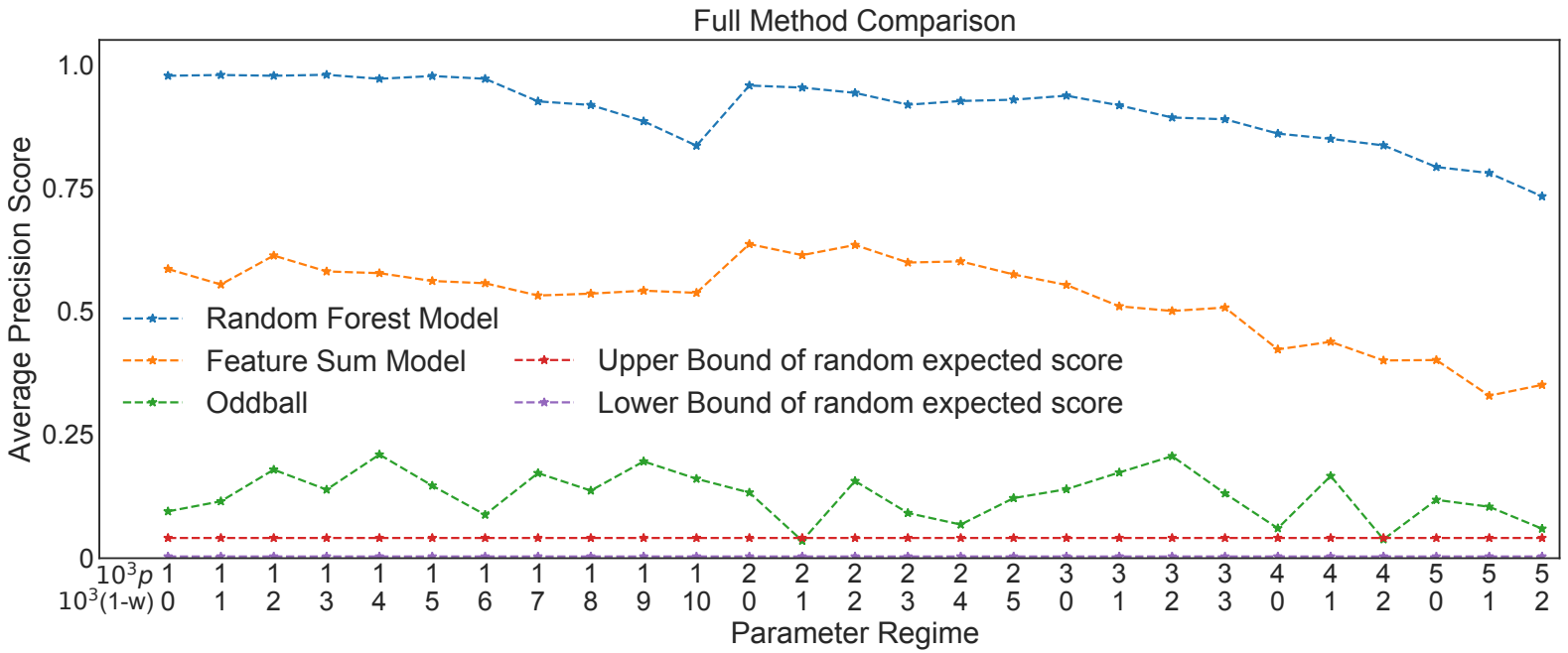
We select 44 features using the consensus rank over the parameter regimes.



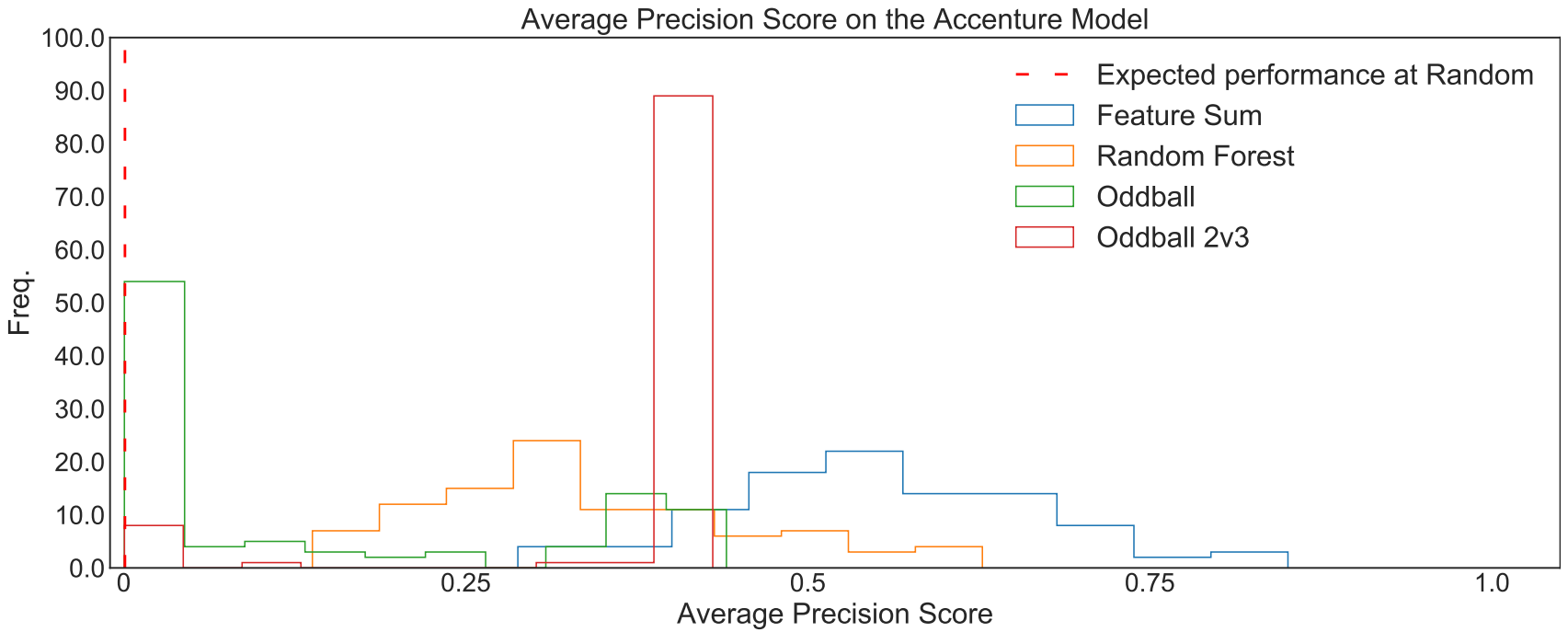
Most important features in each (p, w)- region



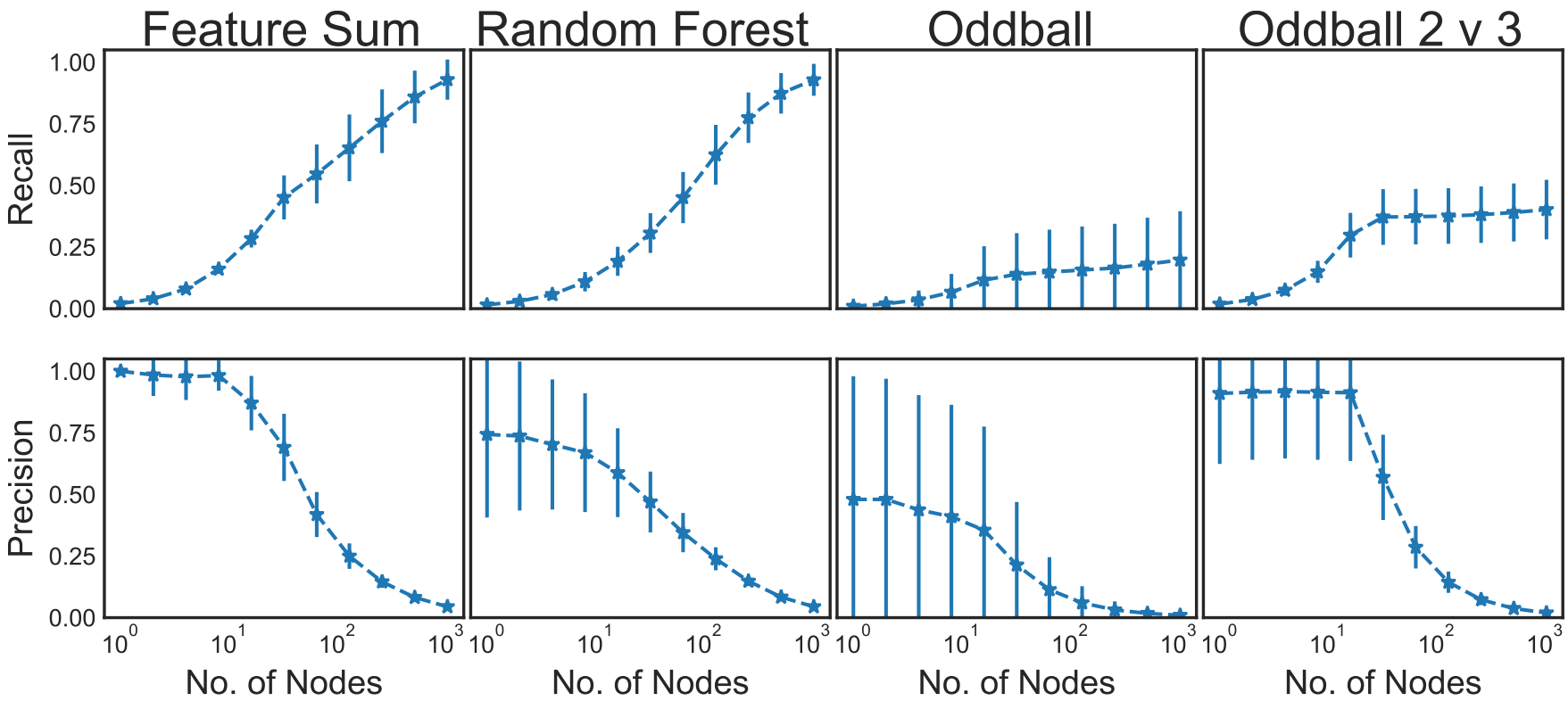
Performance of the classifiers



Average precision for the Accenture data



Recall and precision for the Accenture data



Conclusions

- We provide a novel anomaly detection method which uses network comparison and localisation resulting in 140 features.
- We construct a benchmark graph to develop and test the method.
- Our summation method and the random forest method both obtain good performance on this model, and outperform Oddball.
- We also test on a model, the generating mechanism of which we did not have access to during the main development.
- We perform well obtaining 93%+ of the embedded anomalies within the first 2% of nodes.

Future Work

- Real data (including applications to cyber datasets);
- Incorporate time dependency;
- Explore a Bayesian approach instead of a null model;
- Explore further the statistics that are driving the performance.