Implementation of TrustRank Algorithm

Rahul Verma
AI23MTECH11008
Indian Institute of Technology, Hyderabad
Email: ai23mtech11008@iith.ac.in

Pratik Yawalkar
AI23MTECH11006
Indian Institute of Technology, Hyderabad
Email: ai23mtech11006@iith.ac.in

Sarang Kukade
EM23MTECH11008
Indian Institute of Technology, Hyderabad
Email: em23mtech11008@iith.ac.in

Mahesh Deshmukhe
CC23MTECH11003
Indian Institute of Technology, Hyderabad
Email: cc23mtech11003@iith.ac.in

Aniruddha Paradkar
AI23MTECH13001
Indian Institute of Technology, Hyderabad
Email: ai23mtech13001@iith.ac.in

Abstract

Done as a part of the coursework on Fraud Analytics Using Predictive and Social Network Techniques (CS6890). Code with dataset is available at this link.

1. Problem Statement

The objective of this work is to **pinpoint bad nodes** in a network of nodes, where edge weights indicate the strength of a transaction between a set of nodes. We aim to develop a method that effectively spots these ”bad” nodes by examining **how they connect and interact within the network**.

In this work, we utilize the **TrustRank Algorithm** to effectively **identify malicious nodes** within the network, by employing a known set of bad nodes as seed, provided by trusted oracles. Consequently, the resulting algorithm rankings will clearly distinguish bad nodes from good ones by **assigning higher scores to untrustworthy bad nodes and lower scores to trustworthy ones**. Thus, enabling accurate detection of untrustworthy nodes within the network.

As our method employs the TrustRank Algorithm to effectively identify bad nodes within a network by assigning them higher rankings, we can heuristically describe this approach as the **Implementation of Mistrust Rank**. This method utilizes a set of known bad nodes (provided by oracles) as the input to enhance detection accuracy.

2. Description of the Dataset

The dataset consists of two CSV files: `Payments.csv` and `bad_sender.csv`. The `Payments.csv` file contains three columns: ‘Sender’, ‘Receiver’, and ‘Amount’. Each row (except the column headings) denotes a transaction where the amount is transferred from a sender node to a receiver node. There are a total of 130,535 such transactions provided in this file. These transactions can be used to generate a directed graph where edges indicate transactions between the sender-receiver nodes. Each edge represents the strength of the transaction occurred between the sender and the receiver. In cases where there are multiple edges from sender to receiver, these edges are consolidated into one by summing up their values. For nodes with no outlinks, which can lead to leakage, synthetic links are created back to all the identified bad nodes given in `bad_sender.csv` file with equal weighting. It should also be noted that there are 800 unique sender/receiver nodes in `Payments.csv` file. For convenience, such as statistical interpretation, the numeric IDs of these nodes are mapped between 0 and 799 without any loss of generality.

The latter file consists of IDs of 20 bad nodes under the column heading of ’Bad_Sender’. We have used these 20 nodes as input bad seed in our method.

2.1. Statistics of the dataset

- `Payments.csv`:
  - ‘Sender‘: Sender Node.
  - ‘Receiver‘: Receiver Node.

- 'Amount': Amount transferred by the Sender to the Receiver.
  - There are cases where there exists multiple links between the same two nodes. In those cases, the links are consolidated into one.
- Counts:
  - Unique Sender/receiver: 799
  - Nodes with no outgoing link: 96

- `bad_sender.csv`: List of bad nodes for using as a seed set.
  - Bad nodes: 20

3. Algorithm Used

3.1. TrustRank

TrustRank is a **topic-specific variation of PageRank** that incorporates the notion of a trusted seed set of pages. The goal of TrustRank is to make use of these oracle-provided trusted pages to identify other trustworthy pages. The idea is to **begin with, this set of trusted seed nodes and iteratively propagate the trust score along the graph edges**. Equation 1 shows the TrustRank equation after taking into account both seed set and edge weights. We show a short pseudo code in Algorithm **??** for the reader's reference. For a much detailed code check link.

In our study, we harness the principles of TrustRank algorithm to identify suspicious nodes within a network. By employing a predetermined set of known malicious nodes as seed points, we adapt the TrustRank algorithm to prioritize nodes exhibiting untrustworthy behavior, thus establishing what we term as **MistrustRank**. This approach reverses the conventional TrustRank methodology, assigning higher ranks to nodes associated with deceptive or malicious activities and lower ranks to those deemed reliable. This adaptation enables us to pinpoint nodes that exhibit characteristics contrary to the expected norms of trustworthiness. Through iterative computations and propagation of mistrust signals, our algorithm effectively isolates and **highlights potentially harmful entities within the network with high mistrust ranks**, offering insights crucial for mitigating risks and enhancing the network's overall integrity and security.

$$R(u) = \sum_{v \in B_u} \rho_v * R(v) * \beta + (1 - \beta) * \frac{1}{d_u} \qquad (1)$$

where:
u = a node
B_u = the set of u's backlinks

Algorithm 1 TrustRank Algorithm Implementation

Input:
  T: transition matrix
  N: number of Nodes
  α_B: decay factor
  M_B: number of iterations
Output:
  t^*: MisTrustRank scores

procedure MISTRUSTRANK
  $d \leftarrow 0^N$ ▷ initialize d as zero vector of length N
  for all node ∈ Nodes **do**
    if node ∈ bad_senders **then**
      d(Nodes.index(node)) $\leftarrow 1$ ▷ High mistrust for bad senders
    end if
  end for
  $d \leftarrow \frac{d}{\|d\|}$ ▷ normalize static score distribution vector
  $t^* \leftarrow d$ ▷ compute MisTrustRank scores
  for $i \leftarrow 1$ to M_B **do**
    $t^* \leftarrow \alpha_B \cdot T \cdot t^* + (1 - \alpha_B) \cdot d$
  end for
  return t^*
end procedure

ρ_v = weightage of $R(v)$ based on the total transaction amount in forward links of nodes v and its corresponding edge value between node v and u

β = damping factor

d_u = node u's initialized value at start based on the provided bad seed set

3.1.1 Generating inputs for TrustRank Algorithm

In the context of our MisTrustRank algorithm, the preparation of input data is a crucial step that significantly impacts the effectiveness and accuracy of the resulting Mistrust scores. Our algorithm take bad senders(as seed) and Transition matrix (generated from a weighted directed graph) as input to return back Mistrust scores for each node. The bad senders seed is available in `bad_senders.csv` file. However, we need to generate a weighted directed graph from `Payments.csv` file to extract transition matrix. The generation of directed graph follows mentioned algorithm and is of utmost significane for extracting transition matrix.

The generated graph from the above algorithm can be viewed using networkx library. The output graphs looks like figure 1 below:

Algorithm 2 Algorithm for Generating a Directed Weighted Graph

procedure GENERATEGRAPH
  Initialize graph
  while entries in Payments dataset **do**
    Extract $(\text{sender}, \text{receiver}, \text{amount})$
    if sender $=$ receiver **then**
      Skip adding self-loop
    else
      Add nodes sender, receiver to graph if absent
      Add or update edge $(\text{sender}, \text{receiver})$ with weight amount
    end if
  end while
  Normalize outgoing edge weights so that $\sum_{\text{receiver}} w_{\text{sender,receiver}} = 1$ for each sender
  for each node in graph **do**
    if node is dangling node but not a bad node **then**
      if bad nodes are specified **then**
        Reconnect with: weight $= \frac{1}{\text{total bad nodes}}$ to each bad node
      else No bad nodes in network
        Connect uniformly: weight $= \frac{1}{\text{total nodes}}$ to all nodes
      end if
    end if
  end for
  Display graph properties
end procedure

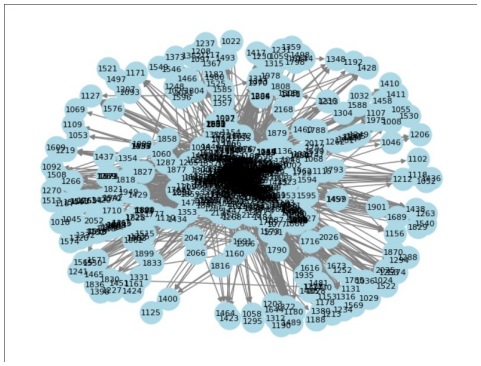

Figure 1. The directed graph does not have any self loops, also multiple edges in same directed are clubbed into one edge.

3.1.2 Potential Leakages in Trust Rank Algorithm

Nodes with no outlinks tend to accumulate the incoming values and cause leakage. In the implemented code, dangling nodes are effectively handled to mitigate the potential leakage in the TrustRank algorithm. Through meticulous consideration of network structure, the code identifies dangling nodes(those devoid of outbound links) and strategically incorporates them into the trust propagation process. Leveraging the concept of transition matrices, the code redistributes trust from dangling nodes by **connecting those nodes back to all the bad nodes with equal weightage**. That way, the accumulated values are propagated back to the bad nodes equally. In this code they are connected to all bad nodes with the weightage in accordance with **Algorithm 2** mentioned above.

3.1.3 Generating Transition matrix from the graph

The generate_transition_matrix function constructs a transition matrix from a directed weighted graph, essential for MisTrustRank Algorithm as mentioned in algorithm 3 below. It initializes a matrix with nodes as rows and columns, setting all initial values to zero. The function iterates through each node, calculates total outgoing weights, and distributes transition probabilities by normalizing the outgoing weights. For nodes without outgoing connections, known as dangling nodes, the function redistributes their weights evenly among predefined 'bad nodes' or, if none, uniformly across all nodes. This ensures the matrix remains stochastic, suitable for probabilistic network analysis.

3.2. Hyperparameters

- 'damping_factor' : 0.85
- 'iterations' : 100

4. Results

The code takes a directed graph(Transition Matrix and number of Nodes) as input and computes the scores for each node in the graph. The output of the algorithm is a list of **MistrustRank scores**, one for each node in the graph.

While there is no ground-truth data available for us to compare our results against, we can check the ranks of the oracle-provided bad node which must be high if the algorithm worked perfectly.

Algorithm 3 Generate Transition Matrix

procedure GENERATETRANSITIONMATRIX
  $nodes \leftarrow$ sorted(self.nodes.keys())
  $transition\_matrix \leftarrow$ DataFrame initialized with 0s, indexed by $nodes$
  for $sender$ in $nodes$ **do**
    $sender\_node \leftarrow$ self.nodes[$sender$]
    $total\_out\_weight \leftarrow$ sum($sender\_node.adjacent.values()$)
    if $total\_out\_weight > 0$ **then**
      for $(receiver, weight)$ in $sender\_node.adjacent.items()$ **do**
        $transition\_matrix[sender][receiver] \leftarrow \frac{weight}{total\_out\_weight}$
      end for
    else
      if self.bad_nodes$\wedge sender \notin$ self.bad_nodes **then**
        for bad_node in self.bad_nodes **do**
          $transition\_matrix[sender][bad\_node] \leftarrow \frac{1}{\text{len(self.bad\_nodes)}}$
        end for
      else
        $transition\_matrix.loc[sender] \leftarrow \frac{1}{\text{len(nodes)}}$
      end if
    end if
  end for
  return $transition\_matrix$
end procedure

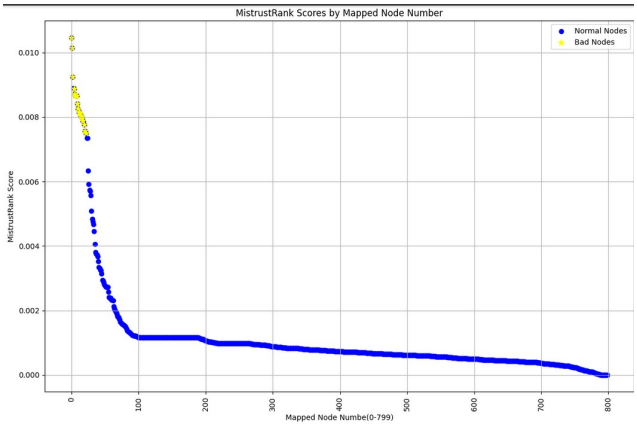

Figure 2. All nodes are arranged in the descending order of their Mistrustrank scores with bad sender seed nodes highlighted.

From figure **??** we can see that the majority of the ground-truth bad nodes have a higher score. This **shows that the algorithm has worked as intended**.

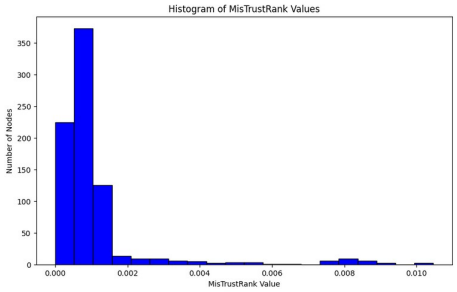

Figure 3. Histogram of the ranks of all the nodes.

Figure 3 shows that most of the nodes have ranks between 0.00 and 0.01 while a few nodes have higher ranks indicating that the majority of the nodes are good nodes excluding a specific few.

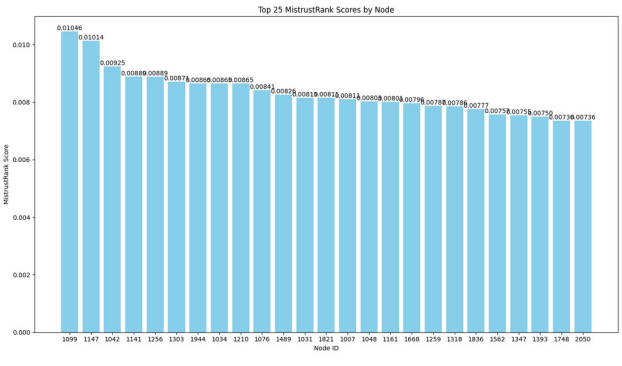

Figure 4. Top-25 Untrustworthy sender nodes.

Figure 4 Out of the top 25 nodes exhibiting the highest Mistrust values, our analysis reveals that 20 nodes coincide with the identified seed bad sender nodes, ranking within the top 22 positions of nodes exhibiting mistrust. This observation corroborates the efficacy of our algorithm, lending credence to its intended functionality and reinforcing its effectiveness in discerning nodes exhibiting suspicious behavior.**Additionally, we observe that some nodes possess considerably higher MistrustRank scores than the officially identified bad nodes. Such nodes are flagged as potential suspects by our algorithm and warrant further investigation for possible engagement in fraudulent activities.**