Implementation of TrustRank Algorithm

Morey Piyush Prabhakar
AI23MTECH14003

Dindorkar Mayuresh Rajesh
CS23MTECH14007

Sanyam Kaul
CS23MTECH14011

Shrenik Ganguli
CS23MTECH14014

1. ABSTRACT

The objective of this assignment is to identify 'bad' nodes in a dataset using the **TrustRank** algorithm.

2. Problem Statement

Our objective in this assignment is to identify the **malicious or 'bad' nodes** (senders) in a network of nodes formed from the Payments dataset. In this dataset, nodes are the **'Senders'** and **'Receivers'** and an edge between the two nodes indicates a transaction between the pair of sender and receiver. The weight of each edge between two nodes in the network is calculated by the amount of that transaction, thus indicating the strength of the transaction.

To achieve this objective, we implement the **Trust Rank** algorithm, which essentially assigns a **trust rank score** (or bad trust score) to each node based on its connections and transactions with 'bad' nodes. So even though ideally, trust rank scores are higher for good nodes, in our implementation, we have defined it such that higher trust rank scores indicates a greater likelihood of that sender being a 'bad' node. This can be thought of as **DistrustRank** algorithm.

3. Description of the dataset

The dataset provided comprises two distinct files: 'Payments.csv' and 'bad_sender.csv'

- **'Payments.csv'**: Consists of 130,535 rows and three columns - 'Sender', 'Receiver' and 'Amount', where each row denotes a transaction between a particular buyer and seller, whereas the columns denote the Buyer ID, Seller ID and the Transaction Amounts.
- **'bad_sender.csv'**: Consists of 20 rows and a single column - 'Bad Sender' which gives us the IDs for the 'bad' nodes (bad senders).

There are 799 unique nodes in the dataset, and as per the data provided to us, atleast 20 nodes (which are given to us) are malicious or 'bad'. Thus, for ease of computations, after extracting the unique nodes, we have indexed them from 0-798, without loss of generality.

4. Algorithm Used

4.1. PageRank

Although Trust Rank is the primary algorithm utilized in this project, in order to comprehend it, we must first understand the Page Rank algorithm, the concepts of which were then applied to the development of Trust Rank.

In the world of search engines, PageRank is an important algorithm that evaluates the relevancy of web sites using a number known as the **PageRank score**. This score determines where a page appears in search results, depending on the quantity and quality of inbound links. The idea at the heart of PageRank is that a page gains more weight as it gains links from credible sources. However, in this case, we depart from the traditional PageRank methodology and instead use a methodology called TrustRank, which is based on its core idea.

4.2. TrustRank

The **topic-specific** version of PageRank is called TrustRank. It is an algorithm whose primary goal is to identify websites that are trustworthy and those that are fraudulent. A trusted seed set of pages is incorporated into TrustRank, with the transitivity of trust serving as the base, the trustworthy pagess offered by these oracles are used to identify other trusted pages, which is the aim of TrustRank. Thus, rather than solely relying on the quantity and quality of inbound links, TrustRank harnesses a designated set of trusted seed pages, typically sourced from authoritative entities like government or academic institutions.

4.2.1 Constructing a graph from the dataset

The first step of this algorithm is to construct a directed graph from the 'payments.csv' dataset. In this graph, an edge from one node to another indicates a transaction between the two nodes. The weight of the edges is calculated as:

- If there are more than one transactions (edges) between the same pair of sender and receiver, we **sum up the amount across all these transactions** and make a single edge of the total amount.
- If a single sender node performs multiple transactions, we calculate the total amout sent by that node and then **divide each amount by this total sum.**

4.2.2 Avoiding trust leakage

For the nodes in the dataset which only act as receivers in all transactions involving them, i.e., **lack outgoing edges**, we extract these nodes and connect them to the 20 'bad' nodes by edges of weight equal to 1. Thus by doing this, we ensure that the **connected nodes have no outlinks back to bad nodes to avoid trust leakage.**

4.2.3 Constructing Transition Matrix

The second step of the algorithm is to create a transition matrix. This will be a $\mathbf{799 \times 799}$ **matrix** such that all nodes can be mapped to all other nodes in the graph. An element A_{ij} in the graph denotes the **weight of the edge from sender node i to receiver node j**. The weights are calculated as discussed above, and since the weights calculated are **normalised, the resulting transition matrix is also normalized.**

4.3. Constructing Initial Trust Rank Vector

In the third step, we create a **799-dimensional matrix**, where there is an initial value for each node. The initial value of the good nodes is set to 0, whereas for the bad nodes, the value is set to $\frac{\mathbf{1}}{\mathbf{20}} = \mathbf{0.05}$ (since there are 20 known 'bad' nodes). Thus, the sum of all the elements in this vector is 1.

4.4. Executing Trust-Rank algorithm

The fourth step is the execution of actual trust rank algorithm for 100 iteration with hyperparameters: **damping factor = 0.85 and number of iterations = 100** as shown in the figure 1. Here, the graph's **transition matrix** T**, the total number of nodes** N**, and a number of hyperparameters like** L**,** M_B**, and** α_B are given to the algorithm as input. The `SelectSeed` function is called upon by the algorithm in the first phase, and it produces a vector s that

function TrustRank
input
  $\mathbf{T}$ transition matrix
  N number of pages
  L limit of oracle invocations
  α_B decay factor for biased PageRank
  M_B number of biased PageRank iterations
output
  $\mathbf{t}^*$ TrustRank scores
begin
  // evaluate seed-desirability of pages
(1) $\mathbf{s} = \mathsf{SelectSeed}(\ldots)$
  // generate corresponding ordering
(2) $\sigma = \mathsf{Rank}(\{1,\ldots,N\}, \mathbf{s})$
  // select good seeds
(3) $\mathbf{d} = \mathbf{0}_N$
  for $i = 1$ to L do
    if $\mathsf{O}(\sigma(i)) == 1$ then
      $\mathbf{d}(\sigma(i)) = 1$
  // normalize static score distribution vector
(4) $\mathbf{d} = \mathbf{d}/|\mathbf{d}|$
  // compute TrustRank scores
(5) $\mathbf{t}^* = \mathbf{d}$
  for $i = 1$ to M_B do
    $\mathbf{t}^* = \alpha_B \cdot \mathbf{T} \cdot \mathbf{t}^* + (1 - \alpha_B) \cdot \mathbf{d}$
  return $\mathbf{t}^*$
end

Figure 1. The TrustRank algorithm

represents the "desirability" of each node as a seed node. The `Rank` function then produces a permutation of the vector x, represented as x_0, putting its members in decreasing order of their respective s**-scores**. As a result, the attractiveness scores given to each node are the basis for the reordering. In the next step, the static score distribution vector d is created by calling an oracle function on the L most suitable seed nodes.

Bad node entries are assigned to $\frac{1}{20}$, but all other entries stay at 0. The program first normalizes vector d to make sure all of its entries add up to 1. Then, it assesses TrustRank ratings. A **biased PageRank algorithm** is used to achieve this, substituting d for the uniform distribution. This stage is noteworthy because it includes trust dampening and splitting. Specifically, a node's trust score is split among its neighbors and dampened by α_B. After the algorithm runs, the vector t^* represents the TrustRank scores that are produced. The resulting scores in the given example, where $\mathbf{M_B} = 20$ **and** $\alpha_\mathbf{B} = 0.85$, show how trustworthy each node in the dataset is in relation to the others.

5. Results

To evaluate the performance of the Trust Rank algorithm, we plot the graphs to verify whether the 'bad' nodes have indeed been given higher bad trust scores.

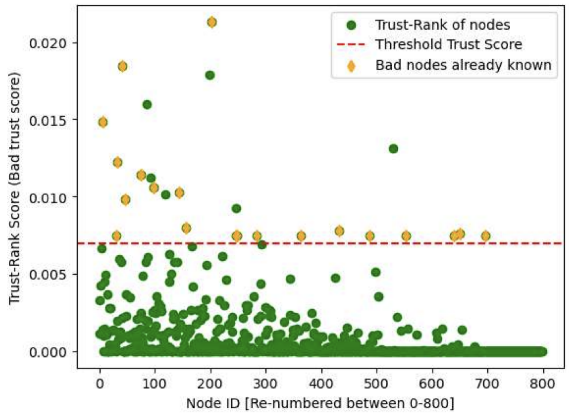

Figure 2. Scatter plot of Node ID vs Trust Score

Figure 2 Here, we can see that all the bad nodes (depicted as yellow coloured dots) have bad trust scores greater than 0.003. Here, the red dotted line depicts the threshold value for identifying the 'bad' nodes.

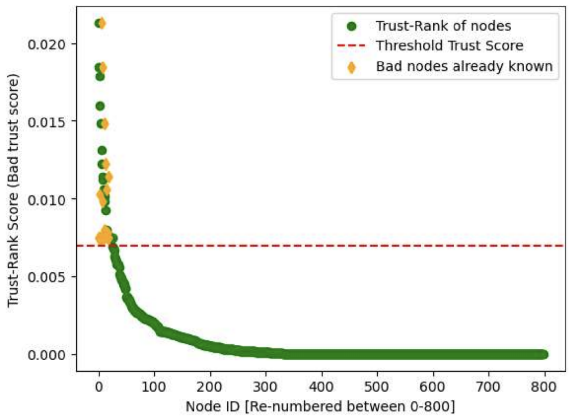

Figure 3. Plotting trust scores of nodes in descending order

Figure 3 From the above diagram we can observe that 0.003 is an effective threshold for differentiating between good and bad nodes.

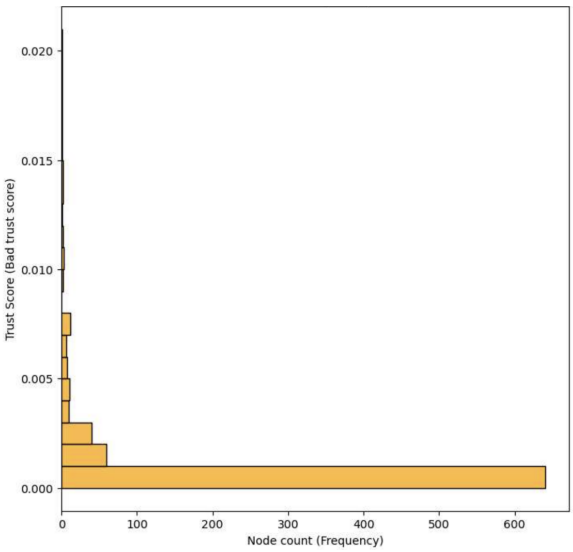

Figure 4. Trust Score vs Frequency of nodes

Figure 4 From the above plot, we can clearly see that most of the nodes are good with trust score ranging from 0.000 to 0.003.

6. Conclusion

- We can observe that, all the bad nodes have the trust score (bad trust score) above the threshold **(0.007)**.
- Bad nodes are highlighted by diamond symbol in above plots.
- Hence, all the good nodes (non-diamond nodes) that lie above the threshold are the **possible bad nodes**.