Implementing Trustrank using Pregel framework

Yuvraj Shekhawat

May 16, 2023

1 Problem statement

In a business ecosystem, there are numerous dealers who frequently buy and sell goods to each other. Each dealer is identified by a unique id associated with it. To track the trustworthiness of these dealers, we are provided with a CSV file containing transaction details in the format of *(seller id, buyer id, transaction amount)*.

We are also provided with a list of bad dealers who have been flagged for exhibiting suspicious behavior. These dealers may have indulged in fraudulent activities or have a history of unreliable transactions.

We aim to identify the bad dealers who may have a negative impact on the overall trustworthiness of the ecosystem. For that we propose to use the TrustRank algorithm to assign a bad trust score to all dealers. The more the bad score of a dealer is, the more likely he is fraud.

The TrustRank algorithm is well-suited for this problem, as it can effectively evaluate the trustworthiness of nodes in a graph-based model. In our case, the dealers can be represented as nodes in a graph, and the transactions between them can be represented as edges. The TrustRank algorithm can then be applied to this graph to assign trust scores to all the dealers.

We are using Pregel, a distributed computing framework, to implement the TrustRank algorithm. Pregel can handle large-scale graphs and can efficiently process the graph data in parallel. We can use Pregel to propagate the bad trust scores through the graph, starting from the bad dealers, and updating the trust scores of all other dealers in the ecosystem.

In this paper, we will provide a detailed solution of the problem using the TrustRank algorithm and Pregel, and the experimental results obtained using a real-world dataset. We will also analyze the effectiveness of our solution

in identifying bad dealers and improving the overall trustworthiness of the ecosystem.

> The fundamental assumption in our algorithm is that bad dealers tend to sell to other bad dealers. The seed set we are given is of bad dealers. Therefore, we will be calculating the bad trust score of a dealer in our problem.

2 Description of dataset

- We have total of 130535 transactions each containing seller id, buyer id and transaction value.
- There are 799 unique ids which means there are 799 dealers.
- There are total of 130535 transactions but most of them are between same set of dealers. There are 5358 transactions between different dealers.
- Below is a small sample of data from our dataset.

	A	B	C
1	Seller ID	Buyer ID	Value
2	1309	1011	1225513
3	1309	1011	1179061
4	1309	1011	1119561
5	1309	1011	1200934

3 Algorithm

There are two ways to implement trustrank which have the same core idea but are executed in two different ways. We will give algorithm to do it both cases.

1. First way is to use transition matrix for propagation. This approach takes the whole graph in consideration at once.

2. The other way is to use Pregel framework. This approach is vertex-centric and we will need to write algorithm in terms of how each vertex propagates and gets score from its neighbours.

3.1 Using Transition matrix

Transition matrix T is a square matrix of size $N \times N$ where N is the number of vertices such that $T_{i,j}$ denotes the probability of transitioning from vertex i to vertex j through a random walk on the graph.// Below is the algorithm to calculate trustrank using transition matrix.

Algorithm 1 TrustRank Using transition matrix

Input: T: Transition matrix, N: Number of vertices, S: Set of bad vertices IDs, α_B: Decay factor, M_B: Number of iterations
Output: Trust scores for each vertex

Initialize $\mathbf{d}= 0_N$
for $i \leftarrow 1$ to L **do**
    if $\sigma(i) \in S$ **then**
        $d(\sigma(i)) \leftarrow 1$ where $\sigma(i)$ denotes vertex id of i^{th} vertex.
    end if
end for
$d = \frac{d}{\|d\|}$ to normalize the vector.
$t^* \leftarrow d$
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^*

The term $t^* \leftarrow \alpha_B \cdot T \cdot t^* + (1 - \alpha_B) \cdot d$ contains two elements:

1. $\alpha_B \cdot T \cdot t^*$: This term propagates trust from inlinks of a vertex to that vertex.

2. $(1 - \alpha_B) \cdot d$: This term propagates a constant value to all the bad vertices in the original set S since they are the one propagating the trust initially.

3.2 Using Pregel Framework (Think like a vertex)!

In Pregel, a graph is divided into a number of partitions, and each partition is processed by a worker node. The computation is divided into a sequence

of iterations, called supersteps, and each superstep consists of two phases: vertex update and message redistribution.// In vertex update, a vertex processes all the incoming messages from its incoming neighbours, does some computation and sends outgoing messages to its outgoing neighbours. Message distribution redistribute all these messages among vertices.

Algorithm 2 Trustrank using Pregel

Input: N: Number of vertices, S: Set of bad vertices IDs, α_B: Decay factor
Output: Trust scores for each vertex

1: Initialize Trustscores for all bad vertices to be $1/N$ and rest 0.
2: **for** $i \leftarrow 1$ to M_B **do**
3: **for** vertex $v \in$ set of all vertices **do**
4: $MessageSum$ = Sum of all the incoming messages to v.
5: **if** $v \notin S$ **then**

$$\text{Trust score(v)} = \alpha_B \times MessageSum$$

6: **else**

$$\text{Trust score(v)} = \alpha_B \times MessageSum + (1 - \alpha_B)/N$$

7: v sends outgoing messages to outlinks in proportion of edge weights.
8: **end if**
9: **end for**
10: **end for**
11: **return:** A vector containing updated Trustscores of all vertices.

This algorithm is inspired from the idea of random walker. Suppose our random walker is at some vertex at $step_t$. It could either jump to an outgoing link with probability α_B or jump to a bad vertex with probability $1 - \alpha_B$. For it to be on vertex v at $step_{t+1}$.

1. If vertex $v \notin S$, then it could have only crawled to v from an in-link with probability α_B.

2. If vertex $v \in S$, then it could have crawled to v either from an in-link with probability α_B or from a random vertex with probability $1 - \alpha_B$.

4 Data Correction and Algorithm in our question's scenario:

Algorithm 3 Trustrank using Pregel

Input: N: Number of vertices, S: Set of bad vertices IDs, α_B: Decay factor
Output: Trust scores for each vertex

1: Parse the `iron_dealers_data.csv` file to get all the dealers(vertices) and transactions(edges). Edges should be in format (*seller id*, *buyer id*, *total transaction amount*) where *total transaction amount* is the weight of edge. From `bad.csv` get a list of all the bad dealers, Let's call the set of vertices corresponding to bad dealers as S.
2: **for** $i \leftarrow 1$ to M_B **do**
3: **for** vertex $v \in$ set of all vertices **do**
4: $MessageSum$ = Sum of all the incoming messages to v.
5: **if** $v \notin S$ **then**

$$\text{Trust score(v)} = \alpha_B \times MessageSum$$

6: **else**

$$\text{Trust score(v)} = \alpha_B \times MessageSum + (1 - \alpha_B)/N$$

7: **end if**
8: v sends outgoing messages to its outlinks in proportion of edge weights.
9: **end for**
10: **end for**
11: **return:** A vector containing updated Trustscores of all vertices

The present algorithm may be insufficient in scenarios where a given vertex lacks any outgoing edges. This circumstance implies that the random crawler will exclusively transition to a bad node with probability $1-\alpha_B$, resulting in score loss as only a portion of the vertex's score is transmitted. In actuality, the crawler will transit to a vertex belonging to the set S with certainty. To guarantee this outcome, we may establish edges from said vertex to all bad nodes. Although counter-intuitive, this measure assigns a probability of α_B to the crawler's transition to an out-link of the vertex in question. In order to address the issue of vertices lacking outgoing edges, additional edges were incorporated into the graph.

Furthermore, the propagation of scores was executed in accordance with the weights assigned to the outgoing edges, which correspond to the values of the respective transactions. It is noteworthy that the supplementary edges introduced to ensure that all vertices possess outgoing edges hold equal weights, given that the crawler is equally likely to traverse any random bad node.

5 Results

We run the above algorithm on our dataset. Some of the observations are:

1. Most of the dealers have score 0.
2. Most of the bad dealers from the set S have high Trustscores.
3. Below is the table of statistics on final Trustscores.

Central tendency	**Value**
Mean	0.001251
Standard deviation	0.004000
Median	0

Table 1: Most of scores are 0, that's why median is 0

To visualize the scores, We printed a table. Below are some of the dealers which have highest Bad Trustscore:

Dealer ID	Trustscore
1088	0.0481873
1144	0.0464345
1007	0.0376523
1210	0.0245252
1034	0.0231957