Bitcoin Alpha Trust Network Analysis

Arjun Narukkanchira Anilkumar, Harsh Verma, Kavyashree Prasad S P and Karampreeth Singh,
Department of Electronics and Computer Engineering, IUPUI

Abstract Weighted signed networks (WSNs) are networks in which edges are labeled with positive and negative weights. WSNs can capture like/dislike, trust/distrust, and other social relationships between people, thus forming a comprehensive network for analysis using machine learning. In this paper, user-to-user trust and distrust network, Bitcoin Alpha is studied. Behavior of important nodes in the network is examined by finding out the relationship among various centralities of these nodes. The centrality attribute of a node essentially captures how much it is liked/trusted by other nodes, and how much of a "central" role it plays in the network. Nodes are then partitioned based on their centralities. This enables studying the link between the number of partitions and their respective costs. Furthermore, a hypothesis on user trust in the platform after a fraudulent transaction is tested. This study lays the foundation to formulate a Long Short-Term Memory (LSTM) regression model to mimic user behavior to predict when they would make their next purchase after experiencing a fraudulent transaction.

***Index Terms*—Long short term memory regression, Bayesian interference model, Gaussian mixture model, Random forest regressor, Feature Agglomeration, Bayesian Interference Criterion and Silhouette score.**

I. Introduction

This paper illustrates three main objectives namely identifying relationship between various centralities calculated from the network, grouping centralities into different clusters and predictive modeling to determine when a bitcoin user would make the next transaction after undergoing a fraudulent transaction. The dataset used in this paper is Bitcoin Alpha. It is a trust network graph comprising of user ratings in Bitcoin platform. It is a signed edge weighted network where members rate other members from a scale of -10 to 10 in steps of 1. 10 indicating total trust and -10 indicating total mistrust. Each row in the dataset consists of source (node id of user), target (node id of trader), rating and timestamp at the time of rating measured in seconds since epoch. There are 3783 nodes and 24186 edges [1] [2] [3].

Centralities of a graph determine the relative importance of each node. This is inferred from the position occupied by a node within a graph. Determining the impact of the position vs the individual occupying the position helps gain a more intuitive understanding of this concept.

User behavior analysis is becoming essential in various fields spanning from online marketing to anomaly detection. Analysis based on user ratings gives information on users' approval of certain products. Using the Bitcoin alpha dataset an attempt is made to understand user behavior after a fraudulent transaction. This behavior is modeled into a LSTM (Long short-term memory) network to predict their next transaction after undergoing a fraudulent one.

II. METHODOLOGY

A. Relationship Among Various Centrality Measures

Four correlation measures namely degree, eigenvector, betweenness and closeness centrality was determined from the dataset. A crude initial exploration of data analysis is attempted using correlation. Correlation among various centrality measures was calculated and the results for the same are displayed below.

TABLE I
CORRELATION ANALYSIS FOR VARIOUS CENTRALITY MEASURES

	In-degree	Out-degree	Degree	Eigen vector	Betweenness	Closeness
In-degree	1	0.9704	0.99173	0.9081	0.8888	0.4620
Out-degree	0.9704	1	0.99336	0.8712	0.9080	0.4294
Degree	0.9917	0.9933	1	0.8952	0.9057	0.4481
Eigen vector	0.9081	0.8712	0.89529	1	0.6899	0.5652
Betweenness	0.8888	0.9080	0.9057	0.6899	1	0.2898
Closeness	0.4620	0.4294	0.44816	0.5652	0.2898	1

It can be inferred from the table that there is high linear correlation among six centrality measures namely in-degree, out-degree, degree, eigen vector and betweenness centrality. Closeness centrality has the least correlation with other measures. Overall degree centrality being a symmetric

measure exhibits the same behavior in comparison with its asymmetric counterparts (in degree and out degree centrality). This conveys the redundancy of direction in these measures. Hence, only degree centrality will be considered for further data analysis.

To understand the properties of this graph, each centrality measure was ranked from highest to lowest. 3 common trends were observed. They are listed below along with the possible insights they offer.

1. Low Eigenvector and Closeness centrality (nearly 1000 ranks below) compared to their Betweenness and Degree centrality:

This could mean that the node is enclosed in a cluster away from the rest of the network and it connects a small group of nodes to many others. In the Bitcoin dataset, nodes with these properties are acting as brokers in the network connecting a dense isolated cluster of nodes directly connected to it, to peripheral nodes of huge clusters with fewer ties [4].

2. Lower Betweenness centrality (700- 1000 ranks below) compared to other centrality measures:

This could mean that there are multiple paths in the network to reach various nodes. Nodes with this property, may have connections which circumvent the node. Hence the nodes' bridging attribute is redundant due to presence of multiple path [4].

3. Low closeness centrality (850-1000 ranks below) compared to other centrality measures:

Nodes with these properties connect small group of nodes to highly influential nodes (high eigen vector centrality) of bigger clusters[4].

Since closeness had a low linear correlation to other centrality measures, a nonlinear regression model was employed to discern the relationship of closeness to other measures. A polynomial regression model of degree 2 was used to model data from sklearn python[5] . The data was split into 70% training and 30% testing. A quadratic polynomial with 3 independent variables is as follows:

$$F(x,y,z) = ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j \quad (1)$$

Where F(x,y,z)= closeness centrality, x=degree centrality, y=eigenvector centrality and z= betweenness centrality.
The coefficients obtained are

a=0	b=-0.86108603
c=4.56851801	d=-4.12676703
e=144.29222642	f=137.5709008
g=-177.69390516	h=-10.79841045
i=235.69010375	j=-74.3001643

Root mean square (RMS) error for training data and testing data are 0.0275 and 0.0272 respectively. Hence, the model has not overfit or underfit the data. Mean value of ground truth observed for training and testing data are 0.2363 and 0.235 respectively. Sparsity Index (SI) was calculated to determine goodness of RMS error, which is RMS error of predicted values/Mean of true values. This resulted in a Sparsity Index (SI) less than one, indicating that the model has effectively obtained the relationships.

B. NODE PARTITIONING BASED ON CENTRALITIES

Grouping of nodes based on centralities will give more understanding on levels of importance upheld by various nodes in the network. This also helps in categorization of type of influential role played by each node. To accomplish this, a feature selection model is employed to envision the data in a lower dimensional space. Feature selection and dimensionality reduction are some of the major tools used in machine learning to visualize high dimensional features. The original feature space is converted into a reduced or new feature space with fewer dimensions. This reduction is achieved either by selecting a subset of original features(feature selection) or by mapping them into an entirely different feature space (dimensionality reduction) [6].These techniques help to remove redundant features and improve accuracy of the final model by letting it focus on only relevant information [7]. It also aids in interpretability of results obtained from the model.

In order to select important features from all the centralities computed two techniques were used namely:

1. Random forest regression: A random forest algorithm helps to weight the importance of each feature on the model. It does so by calculating the impurity a tree node using that feature induces across all trees in the forest. Hence, a random forest regressor was used to model centralities from sklearn package [8]. It was found that betweenness centrality had the least importance among others.

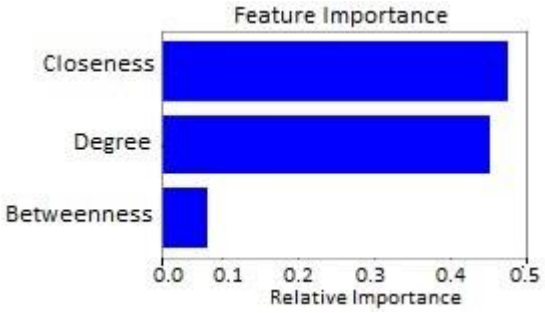

Fig. 1. Relative importance of features obtained after fitting regression model

2. Feature Agglomeration: Agglomerative clustering performs clustering by considering each sample as a cluster of its own and then recursively merging the samples into clusters. It uses a bottom top approach of clustering. Feature agglomeration uses this technique to merge similar features[9]. This was implemented on centralities which resulted in degree and betweenness centralities merged into their arithmetic mean as a single feature.

Hence, based on the above two observations it was deduced that betweenness centrality could be eliminated

from the features space. Also, the strong correlation between betweenness and degree centrality uplifts this inference.
The remaining three features were visualized as a scatter plot in 3D using MATLAB. From the 3d plot it was quite clear that there were uneven sized clusters, non-ellipsoidal, irregular shapes with flat geometry. Hence, to appropriately fit such a distribution Gaussian Mixture Model (GMM) clustering was used [10]. To estimate the number of clusters for this model, BIC (Bayesian Interference Criterion) and silhouette score was used as these two are quite efficient with GMMs [11].
Results for both these techniques are given below:

1. BIC: This estimate gives us a measure of how well the model fits the unknown distribution. Lower scores of BIC is an indicator of good fit. Following this criterion, it can be observed from the figure that more the number of clusters better is the fit.

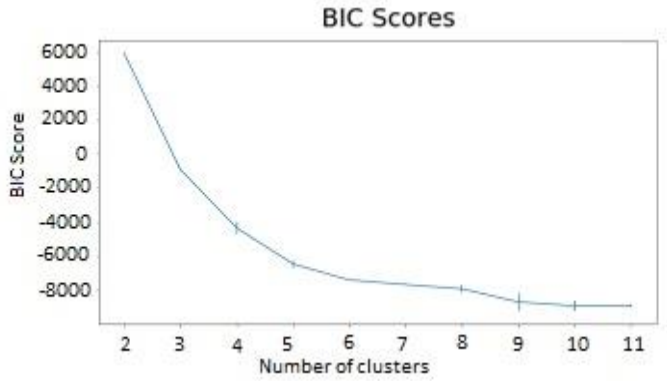

Fig. 2. Number of clusters vs their BIC scores

Hence, BIC does not penalize complex models from over fit in this case. Since the curve is monotonic with different slopes in different regions gradient of BIC can be used to understand the point of biggest slope change. Therefore, performing this suggest that after 5 clusters the slope is almost constant indicating that there is not much gain in increasing the number of clusters [11].

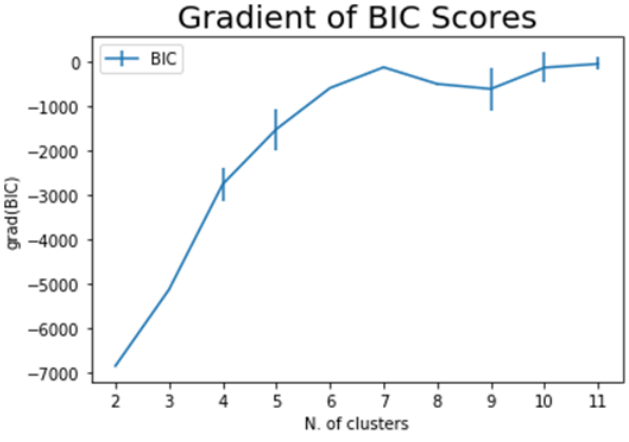

Fig. 3. Number of clusters vs gradient of BIC scores

1. Silhouette score: Silhouette score measures how compactly packed and well separated each cluster are. This is achieved by considering the mean distance between points in the same cluster and mean distance between points in next nearest cluster. Results obtained with this technique further suggested usage of 5 as the optimal number of clusters [11].

Fig. 4. Variation of Silhouette scores with respect to number of clusters

The effect on different clusters on cost function was explored. It can be clearly seen from the figure that as the number of clusters increase the cost function decreases. This is quite evident from the fact that as number of clusters increase the constraint of generalizing behavior of various samples into clusters reduces. Each sample preserves its own properties when number of clusters is increased to the number of data samples. Hence, the choice of number of clusters is always a trade of between reducing cost function and grouping of similar data together.

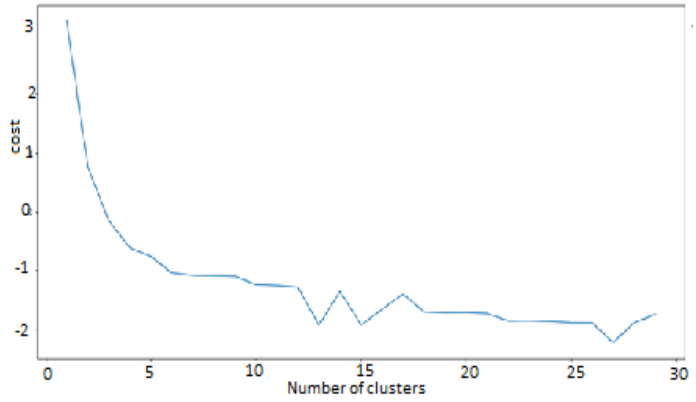

Fig. 5. Variation of cost of GMM clustering with respect to number of clusters

Using five as the optimal number of clusters based on analysis following observations can be inferred. Each centrality measure was ranked from highest to lowest to understand their significance in forming clusters.

Cluster1: In this cluster, all nodes have low values of centralities (high ranks) indicating that these nodes are not regular users of bitcoin making very few transactions. They are also not preferred users for trading among others.

Cluster 2: In this cluster, nodes have a low ranks of degree, high ranks of eigen centrality (around 1000-1500) and high ranks of closeness centrality (above 700-1000).This

suggest that mostly these nodes are enclosed in a cluster away from the network with direct connection to nodes which do not have connection to influential neighbors.

Cluster 3: Nodes in these clusters have high degree centrality with extremely low eigenvector (rank of 2000-3000) and low closeness (rank of 2000-3000) centralities. These nodes follow properties of cluster 2 more strongly.

Cluster 4: These are nodes with lower closeness values compared to their degree and eigenvector centralities [8]. Nodes with these properties are embedded in clusters away from the network but remain connected to many influential (high eigen vector centrality) and other users in the network.

Cluster 5: In this cluster, all nodes have low ranks of centralities (high centrality values) indicating that mostly these nodes are top influential nodes in the network. There is not much variation in ranks among their centrality measures.

C. ANALYSIS OF TRUST IN BITCOIN ALPHA PLATFORM

In Bitcoin Alpha platform, users rate traders based on genuineness of transaction on a scale of -10 (highly fraudulent transaction) to +10 (highly trustable transaction). It is hypothesized that, when a user is encountered with a fraudulent transaction, he/she takes quite a long time (approximately more than 2 months) to gain trust in the platform and make their next transaction. In some cases, may not return. One of the major obstacles in this study is the absence of labels for fraudulent users, who may rate falsely to bring down their competitors. Such users would usually create fake accounts and perform multiple transactions in a short span of time and provide false ratings. This issue could be tackled by filtering out fraudulent users from the dataset prior to evaluating the hypothesis. This is done using the BIRDNEST algorithm. This algorithm implements a Bayesian model to understand types of user rating behavior and the forming a likelihood estimation which measures how much a user deviates from the others[12]. After preprocessing, the data was inspected to test the hypothesis. It was found that Bitcoin users after undergoing a fraudulent transaction do not lose trust in the network and return on an average in 8 days.

D. PREDICTIVE MODELLING OF BITCOIN USERS

Since most Bitcoin users return within a short interval of time and not leave the platform after a fraudulent transaction, an attempt was made to predict their next transaction.To achieve this firstly the dataset was pre-processed to remove fraudulent users who constantly provide negative ratings to their competitors rapidly or at regular intervals [13] [12]. After removal of such users, data was formatted in suitable manner such that the features include rating before fraudulent transaction, fraudulent transaction rating and their corresponding time instants and their out-degree centrality. Output to the network was time instant of rating after fraudulent transaction. These parameters were fed to an LSTM regressor to predict their time of return. LSTM model was used as the data is a time series data and also it was preferred over RNN for more flexibility in controlling outputs[14].

III. RESULTS

The model was successfully trained with 70% training and 30% testing data. The model comprises of 3 hidden layers with 50 units each. An RMS error of 0.04 for training and 0.05 for testing was achieved. Mean of train and test scores were 0.453 and 0.457 respectively. Hence the Sparsity Index for RMS is less than 1 indicating that the model achieved a good fit for the data. Model adapted itself to the trend of ground truth value as evident in figure [7]. The figures of better clarity for the result obtained are attached in the appendix.

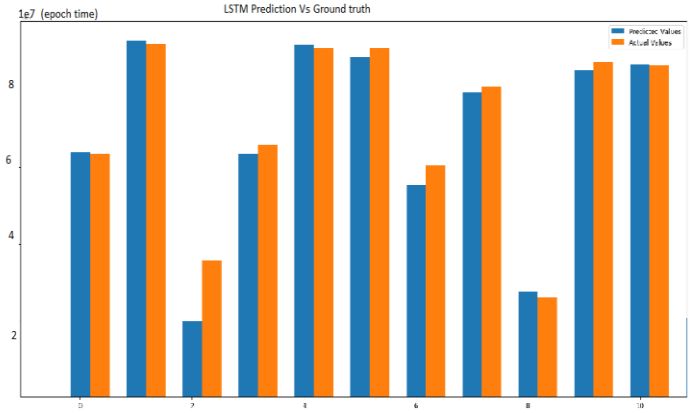

Fig. 6. LSTM Predicted values in epoch time (blue bars) and Ground Truth (orange bar plotted for 10 distinct users in testing dataset.

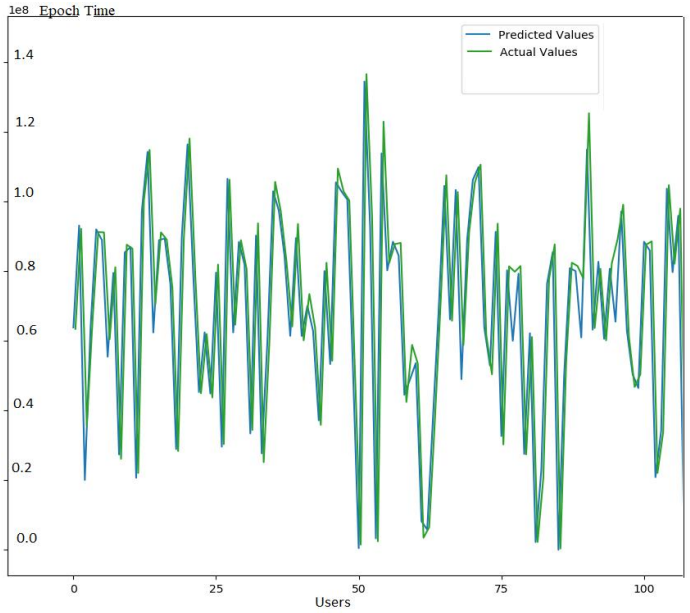

Fig. 7. Line plot of LSTM predicted value(Blue line) and actual value(Green line) plotted for 100 users in test dataset.

IV. CONCLUSION

Node centrality helps decipher the behavior of individuals and groups in the network. With the advent of big data in

graphs there is an accelerating demand to comprehend various statistical properties of the network [15]. Centrality is one of the important topological properties which is used to extract information regarding relative importance of a node in the graph. Using Bitcoin alpha, a signed edge weighted network comprising user ratings a centrality analysis was done and the relationship among centrality measures were observed. It showed that degree, eigenvector and betweenness centralities were well highly correlated with one another. These measures were not well correlated to their closeness centrality. Upon plotting them, it was seen that they follow a non-linear trend with this centrality measure. Hence, a polynomial regression model of degree two was used to compute their relationships. These measures were further clustered using a Gaussian mixture model to understand different patterns within the network. Five basic trends were observed after clustering.

The proposed hypothesis of users losing trust in the platform after encountering a fraudulent transaction was tested. It was found most users return back to the platform to make their next transaction within a few days. This was used as a motivation to develop a predictive model to predict when the user would make next transaction after undergoing a fraudulent one. An LSTM model was successfully trained to emulate this user behavior.

V. LIMITATIONS

The proposed model could be further enhanced to make more accurate predictions with the use of larger size dataset comprising of many genuine users. The proposed LSTM model was trained on users who had not been monitored long enough. Observing an individual over longer period of time would have given the model enough information to imitate user behavior better. By making the following modifications to the system, transaction platforms could understand how fraudulent users affect the trustworthiness of the platform in the minds of genuine users. Also it would give an insight on revenue estimation after an abnormal transaction.

ACKNOWLEDGMENT

We are very thankful to Dr. Dongsoo Stephan Kim for providing us the opportunity to carry out this project. It was a very good learning experience. We are also grateful to our teaching assistant Hamidreza LotfAlizadeh for guiding us whenever needed.

REFERENCES

[1] "SNAP: Signed network datasets: Bitcoin Alpha web of trust network." [Online]. Available: https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html. [Accessed: 04-Nov-2019].

[2] S. Kumar, F. Spezzano, V. S. Subrahmanian, and C. Faloutsos, "Edge Weight Prediction in Weighted Signed Networks," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Barcelona, Spain, 2016, pp. 221–230.

[3] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. S. Subrahmanian, "REV2: Fraudulent User Prediction in Rating Platforms," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining - WSDM '18*, Marina Del Rey, CA, USA, 2018, pp. 333–341.

[4] D. Du, "Social Network Analysis: Centrality Measures," p. 81.

[5] "sklearn.preprocessing.PolynomialFeatures — scikit-learn 0.22 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html. [Accessed: 12-Dec-2019].

[6] "Feature selection for dimensionality reduction." [Online]. Available: https://dl.acm.org/citation.cfm?id=2182379. [Accessed: 12-Dec-2019].

[7] "Intro to Feature Selection Methods for Data Science." [Online]. Available: https://towardsdatascience.com/intro-to-feature-selection-methods-for-data-science-4cae2178a00a?gi=873fc8413a4b. [Accessed: 12-Dec-2019].

[8] "3.2.4.3.2. sklearn.ensemble.RandomForestRegressor — scikit-learn 0.22 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. [Accessed: 12-Dec-2019].

[9] "2.3. Clustering — scikit-learn 0.22 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html. [Accessed: 12-Dec-2019].

[10] "2.1. Gaussian mixture models — scikit-learn 0.22 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/mixture.html. [Accessed: 12-Dec-2019].

[11] "Gaussian Mixture Model clustering: how to select the number of components (clusters)." [Online]. Available: https://towardsdatascience.com/gaussian-mixture-model-clusterization-how-to-select-the-number-of-components-clusters-553bef45f6e4. [Accessed: 12-Dec-2019].

[12] B. Hooi *et al.*, "BIRDNEST: Bayesian Inference for Ratings-Fraud Detection," in *Proceedings of the 2016 SIAM International Conference on Data Mining*, 2016, pp. 495–503.

[13] H. Li *et al.*, "Bimodal Distribution and Co-Bursting in Review Spam Detection," in *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, Perth, Australia, 2017, pp. 1063–1072.

[14] "Simple RNN vs GRU vs LSTM :- Difference lies in More Flexible control." [Online]. Available: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57. [Accessed: 12-Dec-2019].

[15] H. Li, "Centrality analysis of online social network big data," in *2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA)*, Shanghai, 2018, pp. 38–42.

APPENDIX

SOURCE CODE FOR CENTRALITY COMPUTATION

```
import pandas as pd
import networkx as nx
df = pd.read_csv("bitcoin.csv")
data = df.values
G = nx.DiGraph()
for i in data:
   G.add_edge(i[0],i[1],weight=-i[3])

indegree = nx.in_degree_centrality(G)
outdegree = nx.out_degree_centrality(G)
degree = nx.degree_centrality(G)
between = nx.betweenness_centrality(G)
closen= nx.closeness_centrality(G)
eigen = nx.eigenvector_centrality(G)
with open('values_weighted_analy.csv','w') as file:
   file. write ("Node, indegree,outdegree,degree,eigen,between,closen[i]\n")
   for i in eigen.keys():

file.write(str(i)+','+"%f"%indegree[i]+','+"%f"%outdegree[i]+','+"%f"%degree[i]+','+"%f"%eigen[i]+','+"%f"
%between[i]+','+"%f"%closen[i]+"\n")
```

SOURCE CODE FOR CORRELATION ANALYSIS AMONG CENTRALITY MEASURES

```
##correlation Analysis####
import numpy as np
import pandas as pd
INSTANCES = pd.read_csv('values_weighted_analy.csv')
g = INSTANCES.values
testcases_scaled_data= g[0:,3:7]
testcases_scaled_data1 = preprocessing.scale(testcases_scaled_data)
testcases_scaled_data2 = preprocessing.scale(g[0:,1:7])
DDATA = pd.DataFrame(testcases_scaled_data2)
corr = DDATA.corr()
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr,cmap='hsv', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,data_scaled2.shape[1],1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
```

```
ax.set_yticks(ticks)
ax.set_xticklabels(['in degree','out degree','degree','eigen vector','betweeness','closeness'])
ax.set_yticklabels(['in degree','out degree','degree','eigen vector','betweeness','closeness'])
plt.show()
```

SOURCE CODE FOR POLYNOMIAL REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
#Importing the dataset
INSTANCES = pd.read_csv('values_weighted_analy.csv')
g = INSTANCES.values
testcases_scaled_data= g[0:,3:7]
testcases_scaled_data1 = preprocessing.scale(testcases_scaled_data)
testcases_scaled_data2 = preprocessing.scale(g[0:,1:7])
nonlinear = PolynomialFeatures(degree=2)

X_train , X_test, y_train,y_test = train_test_split(testcases_scaled_data[:,:-1],testcases_scaled_data[:,-
1],test_size = 0.3)
X_poly = nonlinear.fit_transform(X_train)
reg_model = LinearRegression()
reg_model.fit(X_poly, y_train)
####training#####
y_pred1 = reg_model.predict(X_poly)
df = pd.DataFrame({'Actual': y_train, 'Predicted': y_pred1})
print('Mean Absolute Error:', metrics.mean_absolute_error(y_train, y_pred1))
print('Mean Squared Error:', metrics.mean_squared_error(y_train, y_pred1))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train,y_pred1)))
print('mean',np.mean(y_train),y_train.min())
#####testing#########
X_test = nonlinear.fit_transform(X_test)
y_pred = reg_model.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
print('mean',np.mean(y_test))
```

SOURCE CODE FOR FEATURE SELECTION

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.cluster import FeatureAgglomeration
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
INSTANCES = pd.read_csv('Norm_val_2.csv')
data_BITCOIN = INSTANCES.values
feature = preprocessing.scale(data_BITCOIN[0:,3:7])
den= pd.DataFrame({'degree':feature[:,0], 'eigen':feature[:,1],'between':feature[:,2],'close':feature[:,3]})
den=den.drop(['eigen'],axis=1)
sen=pd.DataFrame({'eigen':feature[:,1]}
model = RandomForestRegressor(random_state=1, max_depth=10)
df=pd.get_dummies(den)
model.fit(df,sen)
_1 = df.columns
importn = model.feature_importances_
indexex = np.argsort(importn)[-5:]  # top 10 _1
plt.title('Feature Importances')
plt.barh(range(len(indexex)), importn[indexex], color='b', align='center')
plt.yticks(range(len(indexex)), [_1[i] for i in indexex])
plt.xlabel('Relative Importance')
plt.show()

agglo=FeatureAgglomeration(n_clusters=3).fit_transform(feature)
```

SOURCE CODE TO DETERMINE OPTIMAL NUMBER OF CLUSTERS IN GMM MODEL

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from sklearn.mixture import GaussianMixture as GMM
from sklearn import metrics
from sklearn.model_selection import train_test_split
from matplotlib import rcParams
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Importing the dataset
datas = pd.read_csv('values_weighted_analy.csv',usecols=[3,4,6])
data = datas.values
data_scaled = preprocessing.scale(data)
find_data=data_scaled

gauss=GMM(5).fit(find_data)
a=gauss.predict(find_data)
b=a.reshape(-1,1)
c=[]
for i in range(b.shape[0]):
   c.append(b.item(i))
f=np.array(c)
import pandas as pd
pd.DataFrame(f).to_csv("clusterings.csv")
def checking(arr:list, X:int)->list:
   dx=np.argsort(arr)[:X]
   return arr[dx]
numbers=np.arange(2,12)
silt=[]
silt_errd=[]
iterations=20
for n in numbers:
   temporory=[]
   for _ in range(iterations):
      gmm=GMM(n, n_init=2).fit(find_data)
      labels=gmm.predict(find_data)
      sil=metrics.silhouette_score(find_data, labels.reshape(-1,1), metric='euclidean')
      temporory.append(sil)
   val=np.mean(checking(np.array(temporory), int(iterations/5)))
   errd=np.std(temporory)
   silt.append(val)
   silt_errd.append(err)

plt.errorbar(numbers, silt, yerr=silt_err)
plt.title("Silhouette Scores", fontsize=20)
plt.xticks(numbers)
```

```
plt.xlabel("N. of clusters")
plt.ylabel("Score")

numbers=np.arange(2, 12)
bitcoin=[]
glitches=[]
iterations=20
for n in numbers:
   temp1=[]
   for _ in range(iterations):
      gmm=GMM(n, n_init=2).fit(findings)

      temp1.append(gmm.bic(findings))
   val=np.mean(checking(np.array(temp1), int(iterations/5)))
   err=np.std(temp1)
   bitcoin.append(val)
   glitches.append(err)
plt.errorbar(numbers,bitcoin, yerr=glitches, label='BIC')
plt.title("BIC Scores", fontsize=20)
plt.xticks(numbers)
plt.xlabel("Numberof clusters")
plt.ylabel("Scores")
plt.legend()

plt.errorbar(numbers, np.gradient(bitcoin), yerr=glitches, label='BIC')
plt.title("Gradient of BIC Scores", fontsize=20)
plt.xticks(numbers)
plt.xlabel("Number of clusters")
plt.ylabel("gradient(BIC)")
plt.legend()
cost=[]
for i in range(1,100):
   gmm=GMM(i).fit(findings)
   cost.append(gmm.score(findings))


g=list(range(1,100))
real_cost = [ -x for x in cost]
plt.plot(g,real_cost)
plt.xlabel("Number of clusters")
plt.ylabel("cost function")
plt.show()
```

SOURCE CODE FOR ANALYSIS OF TRUST OF USERS IN BITCOIN PLATFORM

```
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import collections
import statistics
df=pd.read_excel('out.xlsx',sheet_name='out')
data=df.values
G = nx.DiGraph()
G.add_edges_from(((u, v, {'weight': d,'time' : t}) for u, v, d, t in zip(data[:,0],data[:,1],data[:,2],data[:,3])))
degree_sequence = sorted([(d,n) for n, d in G.in_degree() if(d>100)],reverse=True)
degree_sequence = sorted([n for n, d in G.in_degree() if(d>100)],reverse=True)
n = []
high_in = []
for i in degree_sequence:
        s= sum([i['weight'] for k,j,i in G.in_edges(i,data=True)])
        l= len([i['weight'] for k,j,i in G.in_edges(i,data=True)])
        n.append(("%0.3f"%(s/l),i))
        high_in.append(i)
n = sorted(n,reverse=True)
datas = []
for avg,node in n:
        Neighbours = [(n1,n2,k) for n1,n2,k in G.out_edges(node,data=True) if k['weight']==-10]
        for n1,n2,k in Neighbours:
                Values = sorted([(h['time'],h['weight']) for m1,m2,h in G.out_edges(n1,data=True)])
                BeforeNeg_Rate  = sorted([h['time'] for m1,m2,h in G.out_edges(n1,data=True) if
h['time']<k['time'] ])
                After_Neg_Rate  = sorted([h['time'] for m1,m2,h in G.out_edges(n1,data=True) if
h['time']>k['time'] ])
                if(len(BeforeNeg_Rate)>2 and len(After_Neg_Rate)>2):

                        dx = 1
                        dy_b = np.diff(BeforeNeg_Rate[-3:])/dx
                        dy_b/=86400
                        dy_a = np.diff(After_Neg_Rate[:3])/dx
                        dy_a/=86400
                        before_time     = BeforeNeg_Rate[-1]
                        next_time               = After_Neg_Rate[0]
                        time_diff =  next_time-k['time']
                        time_diff/=86400
                        var_b_rate              = statistics.variance(dy_b)
                        var_a_rate              = statistics.variance(dy_a)
                        datas.append(time_diff)
                        print(var_b_rate, " : ",var_a_rate, time_diff)
plt.plot(np.arange(len(datas)), datas)
plt.axhline(10,color='green', linestyle='--', linewidth=1.5)
```

```
plt.show()
```

SOURCE CODE FOR LSTM PREDICTION MODEL

```
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import time
numpy.random.seed(7)
scaler = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))
X=pandas.read_csv('../data/train.csv',usecols=[0,1,2,3,4])
Y=pandas.read_csv('../data/train.csv',usecols=[5])
X = scaler.fit_transform(X)
Y = scaler_y.fit_transform(Y)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
X_train = numpy.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test= numpy.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
modelLSTM = Sequential()
nooffeatures = 5
modelLSTM.add(LSTM(50, input_shape=(1,nooffeatures), return_sequences=True))
modelLSTM.add(LSTM(units=50, return_sequences=True))
modelLSTM.add(LSTM(units=50))
modelLSTM.add(Dense(1))
modelLSTM.compile(loss='mean_squared_error', optimizer='adam')
modelLSTM.fit(X_train, y_train, epochs=100, batch_size=1, verbose=2)
train_Predict = modelLSTM.predict(X_train)

test_Predict = modelLSTM.predict(X_test)
# invert predictions
trainScore = math.sqrt(mean_squared_error(y_train, train_Predict))
print('Train Score: %.2f RMSE' % (trainScore))
train_Predict = scaler_y.inverse_transform(train_Predict)
print('mean of train=',numpy.mean(y_train))
testScore = math.sqrt(mean_squared_error(y_test, test_Predict))
print('Test Score: %.2f RMSE' % (testScore))
print('mean of test=',numpy.mean(y_test))
```

```
train_Y = scaler_y.inverse_transform(y_train)
test_Predict = list(scaler_y.inverse_transform(test_Predict))
test_Y = list(scaler_y.inverse_transform(y_test))
train_Predict = list(scaler_y.inverse_transform(train_Predict))
test_Predict = [i[0] for i in test_Predict]
test_Y = [i[0] for i in test_Y]
N=len(test_Y)
plt.bar(numpy.arange(N), test_Predict - min(test_Predict), 0.35, label = 'Predicted Values')
plt.bar(numpy.arange(N)+0.35, test_Y - min(test_Y), 0.35, label = 'Actual Values')
plt.legend()
plt.show()
plt.plot(numpy.arange(N), test_Predict - min(test_Predict), 0.35, label = 'Predicted Values', color = 'red')
plt.plot(numpy.arange(N)+0.35, test_Y - min(test_Y), 0.35, label = 'Actual Values', color = 'green')
plt.ylabel(" epoch time ")
plt.xlabel(" test users ")
plt.legend()
plt.show()
```

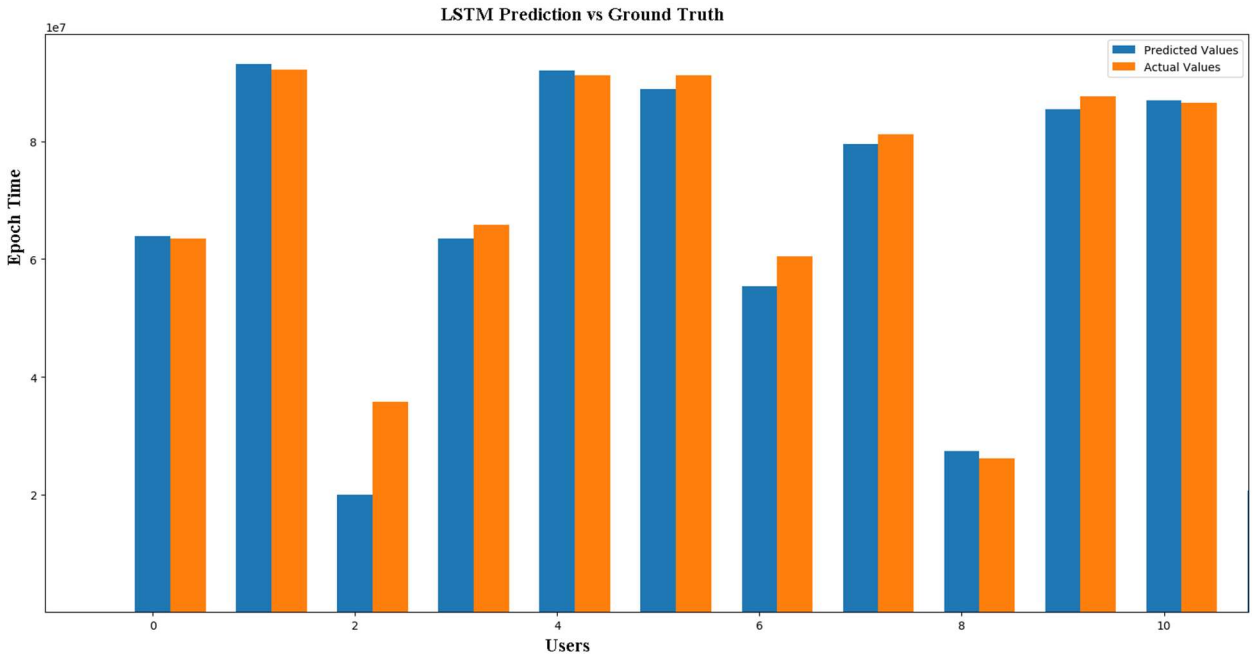

Fig [1] LSTM predictions vs ground truth (Bar plot) (Blue-Predicted values, Orange- Actual Values).

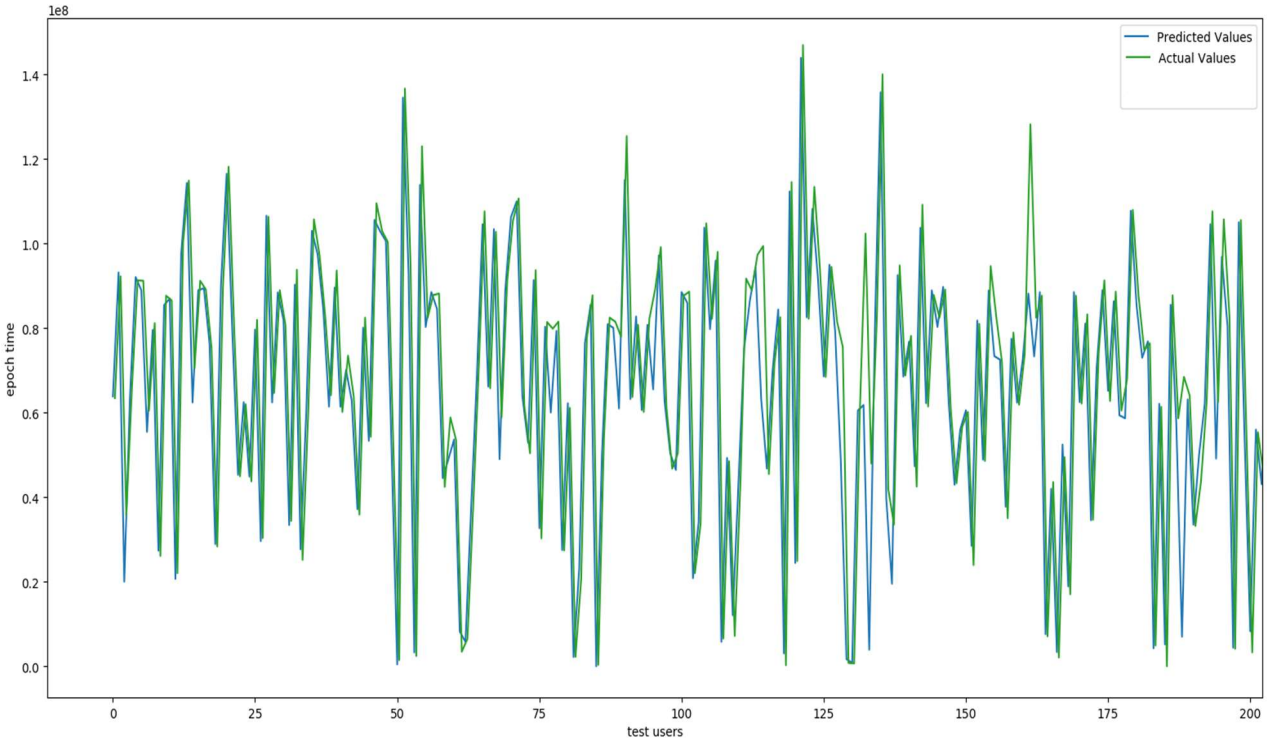

Fig [2] LSTM Model vs ground truth (Line Plot)