Research Report

Community detection on stock market

Ali Bagheri - Hesam Hosseini - AmirAbbas Afzali - Zahra Sorkhei

September 25, 2023

Contents

1 summery

The aim of our research is to analyze the stock market by detecting the community structure using various distance and clustering methods. We utilized Dynamic Time Warping (DTW) as a distance measure due to its ability to capture delays and time shifts and wraps. We also used a modified version of DTW that is not sensitive to singularities which are mostly noises, along with two other distance detection methods. We then constructed four graph based on these distances, where the edges represent similarity. another method used is to represent each stock time series in a low dimensional vector, to do that we extracted the feature vector using auto encoders.

We applied multiple classical clustering using four graph structural dataset and a voting mechanism between these clustering methods and dataset to obtain a more robust result. Furthermore, we employed deep learning techniques such as convolutional neural networks (CNN) to cluster stocks. And we are also working on graph neural networks (GNN) which is modified for graph structural input

we will explain all of the method introduced above in details

Our results demonstrate that the community structure in the stock market can be identified using DTW and other distance measures. The use of autoencoders and deep learning techniques in clustering and community detection showed promising results. Ultimately, our research contributes to the understanding and analysis of the stock market and its structures which can potentially be used in portfolio optimization to inform investment decisions

2 Data-Set

Each stock has multiple features, some of which are outlined below.

1. Open price: This is the price at which a stock begins trading at the start of a trading day.

2. Close price: This is the price at which a stock ends trading at the close of a trading day.

3. High price: This is the highest price at which the stock traded during a particular trading day.

4. Low price: This is the lowest price at which the stock traded during a particular trading day.

5. Volume: This is the total number of shares traded during a particular trading day.

6. Adjusted close price: This is the closing price of a stock adjusted for any corporate actions, such as stock splits or dividend payments.

7. Market capitalization: This is the total value of a company's outstanding shares of stock. It is calculated by multiplying the stock's current market price by the number of shares outstanding.

8. P/E ratio: This is the price-to-earnings ratio, which compares a company's stock price to its earnings per share.

9. Dividend yield: This is the annual dividend payment as a percentage of the stock's current price.

10. Beta: This is a measure of a stock's volatility relative to the overall market. A beta of 1 indicates that the stock is just as volatile as the market, while a beta greater than 1 indicates that the stock is more volatile than the market, and a beta less than 1 indicates that the stock is less volatile than the market

11. Moving averages: A moving average is the calculated average of the closing price for a specific period, typically 20, 50, or 200 days. Moving averages are often used to identify trends and help traders make decisions about when to buy or sell

12. Relative strength index (RSI): The RSI is a technical indicator that measures the strength of a stock's price action by comparing the magnitude of recent gains to recent losses. It ranges from 0 to 100 and is used to determine if a stock is overbought or oversold.

Two approaches for analyzing stocks are fundamental and technical analysis.

Fundamental analysis examines financial and economic factors affecting a company's stock price, while technical analysis analyzes the stock price movement by looking at patterns and trends. Fundamental analysis determines if a company is undervalued or overvalued, using metrics like P/E ratio, while technical analysis uses indicators like MACD to predict future price movements. Investors and traders can use either or a combination of both to make informed decisions.

Iran

Iran data-set consist of top 50 asset (based on Market capitalization) each consisting 5 time series feature including (Open-price,High-price, Low-price, Close-price, Final-price, Volume) The final price in the Iranian stock market is a unique case and is not commonly found in other stock markets. The final price in the stock market is determined by the price that increases in relation to the base volume, a unique feature of Iran's stock market. The purpose of the base volume and final price is to minimize irrational fluctuations and false prices, and instead base the share price on the volume of transactions and supply and demand.

Due to the fact that some assets have only been around for a short time, there are some Null value results. Additionally, some closures have occurred due to events such as the Annual General Meeting of Shareholders, Extraordinary General Meeting, or due to fluctuations in the closing price of more than 50 or 20%, or due to the disclosure of important information.

it's probably impossible to fill in the first kind of missing value but other closure types can be filled using interpolation in this case we used linear time interpolation, A difference between 'time' and 'linear' only occurs if the time index is not equally spaced

this interpolation only makes sense in the case of price features and as of now we have only used 'Adjusted - Final-Price' we are working to find a way to make good usage of the others as well

world

we have also gathered top 8000 world stocks (based on Market capitalization) including 74 countries

Value	Count	Frequency (%)
United States	3703	46.5%
India	562	7.1%
Canada	369	4.6%
Japan	348	4.4%
United Kingdom	260	3.3%
Germany	201	2.5%
Hong Kong	142	1.8%
Australia	129	1.6%
Other values (63)	1822	22.9%

as of now we have only used 'Adjusted-close-Price' we are working to find a way to make good usage of other features as well

3 Methodology

input of clustering algorithms

let's define some parameter:

$x : x_i$ is the ith stock in stock list vector
$A_D : A_D(i,j) = Distance(x_i, x_j)$ representing distance between each two stock
$A_S : A_S(i,j) = Similarity(x_i, x_j)$ representing similarity between each two stock
$X : X$ is feature matrix with dimension $N \times F$ which $X(i,j)$ represent j th feature of i th node (share).

the reason for capitalization is to avoid confusion since we use d and s for point to inner distance inside them

some algorithms take symmetric pair-wise distance matrix (A_D) , some take symmetric pair-wise similarity matrix (A_S) and some take feature matrix (X) as input.

converting A_D to A_S

1. the most general way to change between similarity and distance: a strictly monotone decreasing function $f(x)$. That is, with $f(x)$ you can make similarity = f(distance) or distance = f(similarity). It works in both directions. Such function works, because the relation between similarity and distance is that one decreases when the other increases. These are some well-known strictly monotone decreasing candidates that work for non-negative similarities or distances:

   (a) $f(x) = \dfrac{1}{1+x}$
   (b) $f(x) = 1 - x$
   (c) $f(x) = \exp(-x^a)$
   (d) $= \operatorname{arccot}(ax)$

2. Gaussian kernel

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

   The distance $\|x_i - x_j\|$ used in the exponent is $A_D(i,j)$ The kernel value is in the range $[0,1]$. There is one tuning parameter σ. Basically

if σ is high, $K(x_i, x_j)$ will be close to 1 for any $x_i - x_j$. If σ is low, a slight distance from x_i to x_j will lead to $K(x_i, x_j)$ being close to 0 .

3. graph learning algorithms:
   since Topological measures derived from smooth graphs are more reliable [2] smooth graph learning could be good potential
   In order to learn a graph from smooth signals,

$$\underset{W \in \mathcal{W}_m}{\text{minimize}} \quad \|W \circ A_D\|_{1,1} + f(W).$$

   $f(W)$ has to play two important roles: (1) prevent W from going to the trivial solution $W = 0$ and (2) impose further structure using prior information on W. This said, depending on f the solution is expected to be sparse, that is important for large scale applications.

   it turns out that solving this problem with a specific prior on the weights can result Gaussian kernel [3]

.1 Pair-wise distance/similarity detection

Clustering algorithms usually take two input either similarity or distance one of the method that has proven to be efficient is **DTW**:

DTW stands for Dynamic Time Warping, which is a technique used to measure the similarity between two sequences of feature vectors. It is widely used in speech recognition, handwriting recognition, and music analysis.

The DTW algorithm finds the optimal matching between two sequences by warping the time axis of one sequence to match the other sequence. The cost of the matching is determined by comparing the distance between each pair of corresponding feature vectors in the two sequences. The optimal matching is found using dynamic programming to minimize the total cost. Let X and Y be two sequences of feature vectors, where x_i is the ith feature

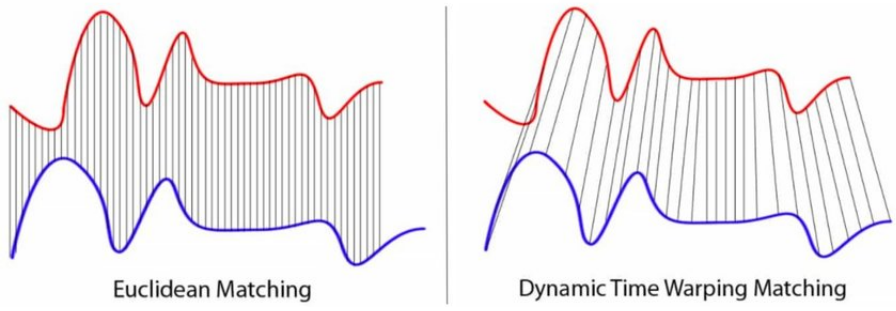

Figure 1. DTW vs euclidean distance

vector in sequence X, and y_j is the jth feature vector in sequence Y in our context i and j are time points. Let $\text{dist}(x_i, y_j)$ be the distance function that measures the distance between two feature vectors.

The DTW algorithm finds the optimal matching between X and Y using the following dynamic programming algorithm:

$$\text{DTW}(X, Y) = \min_{\text{path}} \sum_{(i,j)\in\text{path}} \text{dist}(x_i, y_j), \tag{1}$$

where the minimum is taken over all possible paths, and the path is defined as a sequence of pairs (i, j) that represents the corresponding feature vectors in X and Y. The dynamic programming algorithm computes the minimum cost of the path that matches each prefix of X to each prefix of Y.

The algorithm starts by initializing a cost matrix C of size $(m+1)\times(n+1)$, where m and n are the lengths of X and Y, respectively. The cost matrix $C_{i,j}$ stores the minimum cost of the path that matches $x_1, \ldots, x_i$ to $y_1, \ldots, y_j$.

The initialization is done as follows:

$$C_{0,0} = 0, \tag{2}$$
$$C_{i,0} = \infty, \quad \text{for} \quad i = 1, \ldots, m, \tag{3}$$
$$C_{0,j} = \infty, \quad \text{for} \quad j = 1, \ldots, n. \tag{4}$$

The first row and column are initialized to ∞ to ensure that only valid paths are selected in the minimum computation.

The rest of the cost matrix is computed using the following recurrence relation:

$$C_{i,j} = \text{dist}(x_i, y_j) + \min\{C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1}\}, \tag{5}$$

where $\text{dist}(x_i, y_j)$ is the distance between the ith feature vector in X and the jth feature vector in Y, and the minimum is taken over the three adjacent cells.

Finally, the optimal cost of the matching is given by $C_{m,n}$, which is the minimum cost of the path that matches the entire sequences X and Y. The optimal path can be found by backtracking from $C_{m,n}$ to $C_{0,0}$ using the adjacent cells with the smallest cost. The resulting path represents the optimal matching between the two sequences.

dist function :
the most simple choice is to use euclidean distance but as discussed by Zhao el al. in A similarity measurement for time series and its application to the stock market [4] they proposed DMPSM one thing they used in their study

is **Canberra distance** :
In order to overcome the shortcoming of Euclidean distance, many methods have been proposed. And one of the most effective approaches is Canberra distance (Bijnen, 1973), whose formula is as follows.

$$d^C(X,Y) = \frac{1}{n}\sum_{i=1}^{n}\frac{|x_i - y_i|}{|x_i| + |y_i|}, n = m$$

From above eq, we can see that Canberra distance is a dimensionless quantity and is insensitive to singularities. By taking this advantage of Canberra distance, Faisal et al. (2020) utilized Canberra distance into inter-centroid K-means and made a comparative analysis of Euclidean Distance and Manhattan Distance, find that the Canberra method is superior to Euclidean and Manhattan on their dataset.

Dynamic multi-perspective similarity measurement

Firstly, we assign weights to the segmented series for reflecting the time characteristic after segmenting the whole stock sequences into small pieces of time series. The principle of weight is that the closer the factor to the current time, the greater the weight. In other words, for the time series $X = (x_1, x_2, \cdots, x_n)$, each of the factors is given a weighting on a scale of ω_1 to ω_N, where $\omega_1 < \omega_2 < \cdots < \omega_n$ and $\omega_1 + \omega_2 + \cdots + \omega_n = 1$. That is,

$$X' = (\omega_1 x_1, \omega_2 x_2, \cdots, \omega_n x_n) = (x'_1, x'_2, \cdots, x'_n)$$

For example, suppose the length of sample stock series $X = (x_1, x_2, \cdots, x_n)$ is n, and the weight number is k, for the weight series,

$$W = \left(\frac{1}{1 + \sum_{i=2}^{n}(i+k)}, \frac{2+k}{1 + \sum_{i=2}^{n}(i+k)}, \cdots, \frac{n+k}{1 + \sum_{i=2}^{n}(i+k)} \right)$$

it is obvious that the greater the value of k, the greater the weight of the latter part, which can reflect the time characteristics. Secondly, Canberra distance is embedded into the DTW not only for eliminating the impact of the singularities, but also for coping with time shifts and warpings. Specifically, we utilize Canberra distance to construct the DTW matrix between the weighted series $X = (x'_1, x_2, \cdots, x'_n)$ and $Y' = (y'_1, y_2, \cdots, y'_m)$, that is,

$$D'(p_k) = \begin{bmatrix} d^C(x'_1, y'_1) & \cdots & d^C(x'_1, y'_m) \\ \vdots & \ddots & \vdots \\ d^C(x_n, y'_1) & \cdots & d^C(x'_n, y'_m) \end{bmatrix}_{n \times m}$$

where $d^C\left(x'_i, y'_j\right) = \frac{|x'_i - y'_j|}{|x'_i| + |y'_j|} (i = 1, 2, \cdots, n; j = 1, 2, \cdots, m)$. Then, the calculation formula of DMPSM can be described as

$$d^{\mathrm{DMPSM}} = \min \left\{ \sum_{k=1}^{k} \boldsymbol{d'}(\boldsymbol{p_k}) \right\}$$

where $d'(p_k) = d^c\left(x'_i, y'_j\right)$ (here, the meaning of the symbols in the equation is same with above).

The DMPSM has three major advantages. For better explanation, some examples are given in Fig. 4, where Series A and Series B are two different time series. In Fig. 4(a), it is obvious that Series B has a singularity, but $d^{\mathrm{DMPSM}}(A, B) = 0.35$, indicating that DMPSM can eliminate the impact of singularities. (2) In Fig. 4(b), Series B has time shifts and warpings with A, however, the calculating result of DMPSM shows that it can

cope with time shifts and warpings. (3) In Fig. 4(c), the latter part of the two series is more similar than that of the first half part, and by computing, the result of DMPSM is the minimum, which means that DMPSM can reflect the personalized characteristics of time series. In addition, since the DMPSM is flexible, there is no limit of range for our measurement, and different adjustments can be made according to different situations.

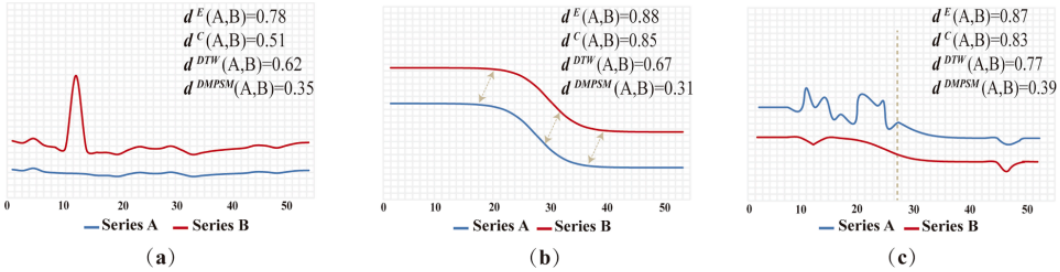

Fig. 4. The three advantages of DMPSM.

what we have done

firstly return is taken from the entire data frame,doing this make input dimension less and the comparison of shares does not depend on their price, because two shares may have different prices, but their trends are the same. this processing makes comparison more meaningful

$$r(t) = \frac{P(t) - P(t-1)}{P(t-1)} \times 100$$

calculating A type input

Algorithm 1: Calculate edge distances and populate adjacency matrix

```
Input: df_price_return
type ← 3;
foreach (i, stock_1) in (enumerate(en_stock)) do
    foreach (j, stock_2) in (enumerate(en_stock)) do
        if i > j then
            dist ←
             calc_edge(df_price_return[stock1], df_price_return[stock2],
             type );
            adj[i, j] ← dist;
            adj[j, i] ← dist;
if type == 1 then
    np.fill_diagonal(adj,1);
else
    np.fill_diagonal(adj,0);
```

before each of these algorithm first we find the most data that we can use for compression

The function getmost takes two series and returns the shorter series and its corresponding portion from the end of the longer series.

Algorithm 2: Get Most Function

```
Input: ser1, ser2
Output: s1, s2, s
s ← min(len(dropna(ser1)), len(dropna(ser2)));
s1 ← tail(ser1, s);
s2 ← tail(ser2, s);
return s1, s2, s
```

for some of algorithm we used this function

Algorithm 3: Calculate Property

Input: $s1$, $s2$, t
Output: *dist*, *mid*, $s1\_jumped$, $s2\_jumped$, *all_gone*
$dist \leftarrow abs(s1 - s2)$;
$mid \leftarrow \frac{s1-s2}{2}$;
$s1\_jumped \leftarrow s1 > 5$;
$s2\_jumped \leftarrow s2 > 5$;
$all\_gone \leftarrow (s1 > 4.5 \text{ and } s2 > 4.5 \text{ and } t > 3.5) \text{ or } (s1 < -4.5 \text{ and } s2 < -4.5 \text{ and } t < -3.5)$;
return *dist*, *mid*, $s1\_jumped$, $s2\_jumped$, *all_gone*;

we have created 4 data set using 4 algorithm explained below

1. simple DTW using dtaidistance library (A_D)

```
# distance_fast is implemented on C and needs C library
  dtw.distance_fast(s1.to_numpy(),s2.to_numpy())

```

2. modified DTW something like DMPSM but since we are not predicting we don't need to focus on recent event so we don't use weight for time point we also tuned parameter to extract better result (A_D)

   (a) window – Only allow for maximal shifts from the two diagonals smaller than this number. It includes the diagonal, meaning that only inner distance is obtained by setting window=1.

   (b) penalty – Penalty to add if compression or expansion is applied

   (c) Prune values based on inner distance. This is the same as passing *ub_euclidean*() to *max_dist*

   (d) only_ub – Only compute the upper bound

   (e) inner_dist – Distance between two points in the time series. One of 'squared euclidean' (default), 'euclidean'. When using the pure Python implementation (thus use_c=False) then the argument can also be an object that has as callable arguments 'inner_dist' and 'result'. The 'inner_dist' function computes the distance between two points (e.g., squared euclidean) and 'result' is the function to apply to the final distance (e.g., sqrt when using squared euclidean). You can also inherit from the 'innerdistance.CustomInnerDist' class.

```
1
2 dtw.distance(s1.to_numpy(),s2.to_numpy(),window=4,inner_dist
    =my_dist(),penalty=1,use_pruning=True,only_ub=True,
    max_step=4)/s
3
4 class my_dist(innerdistance.CustomInnerDist):
5 @staticmethod
6 def inner_dist(x, y):
7     if(x!=0 and y!=0):
8       return abs(x - y)/(abs(x)+abs(y))
9     return 0
10 @staticmethod
11 def result(x):
12     return x
13
```

3. this method unlike others will calculate A_S :

   If the difference in percentage change between two stocks is less than the threshold1, one point will be added to the link. Additionally, if the difference between the average changes of two stocks and the index of 50 stocks is greater than the threshold2, one point will be added as well.also if half of the market have moved together that day is not accounted

Algorithm 4: Semi-correlation calculation

Result: $semi\_cor/s$
$semi\_cor \leftarrow 0$
foreach i, j, k ***in*** $zip(s1, s2, totindex)$ **do**
  $dist, mid, s1\_jumped, s2\_jumped, all\_gone \leftarrow calc\_proprty(i, j, k)$;
  if all_gone **then**
    continue;
  end
  if $dist < thresh$ **then**
    $semi\_cor \leftarrow semi\_cor + 1$;
    if $abs(mid - k) > tresh2$ **then**
      $semi\_cor \leftarrow semi\_cor + 1$;
    end
  end
end
return $semi\_cor \ / \ s$;

4. (A_D) this method is mostly like previous method except the fact that instead of using thresholds it directly adds the distance unless a huge jump has occurred these jumps are most likely noises that should not be consider

```
Data: ser1, ser2, totindex
Result: tot_dist/s
Function calculateDistance(ser1, ser2, totindex):
    tot_dist = 0;
    for (i,j,k) in zip(ser1,ser2,totindex) do
        if not arenan (i,j,k) then
            dist ,mid,s1_jumped,s2_jumped,all_gone = calc_proprty
             (i,j,k);
            if all_gone then
                continue;
            if (not s1_jumped and not s2_jumped) then
                tot_dist += dist;
            else
                tot_dist += min(dist,5);
        else
            continue;
    return tot_dist/s;
```

calculating A using deep learning method

using a convolutional neural network similar to the model introduced in the article by ding et al. "A Novel Similarity Measurement and Clustering Framework for Time Series Based on Convolution Neural Networks,"[1] the adjacency matrix of different stock prices is obtained. The input to this network is the time series of stock prices, and the output of the network is the weighted adjacency matrix of stocks, which indicates the correlation between them. The difference between our implemented model and proposed model in this article is that in our model, each stock input consists of several time series (in this case, 5 time series), and various features including Close, Volume, High, Low, and Open are considered for each stock, and the correlation between two stocks is evaluated based on the sum of the weighted correlations of these five time series.

we call the resulted matrix $A_{S\text{deep}}$.

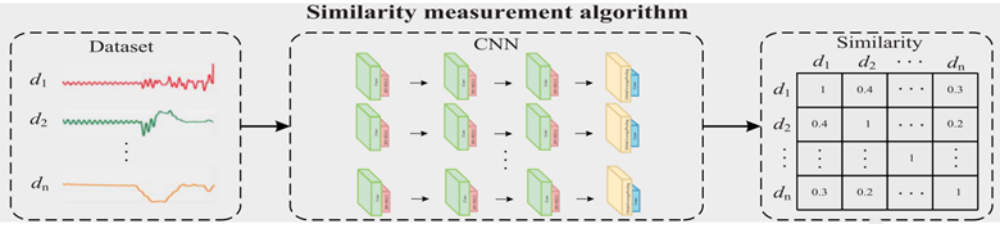

Figure 2. Similarity detection using CNN architecture

.2 Feature vector extraction

Auto Encoders

An autoencoder is a type of neural network that is commonly used for unsupervised learning, including feature extraction. It consists of two main components: an encoder and a decoder. The encoder takes an input x and maps it to a lower-dimensional representation, called a code or latent vector, z. The decoder then takes the code as input and reconstructs the original input x'.

The process of training an autoencoder typically involves minimizing a reconstruction loss, which measures the difference between the original input x and the reconstructed input x'. This is done by adjusting the weights and biases of the encoder and decoder, often using the backpropagation algorithm.

The loss function can be defined as:

$L(x, x') = ||x - x'||^2$

Where $|| \cdot ||$ denotes the L2-norm. The goal of the training process is to find the encoder and decoder weights that minimize the total reconstruction loss over a given dataset.

Mathematically, an autoencoder can be written as:

$$z = g(x; \theta_E) \tag{6}$$

$$x' = f(z; \theta_D) \tag{7}$$

Where g is the encoder function parameterized by θ_E, f is the decoder function parameterized by θ_D, and z is the code or latent representation of x. By extracting the latent vector generated by the encoder, you can reduce the dimensionality of your original feature space and feed it into a clustering algorithm to group similar data points together

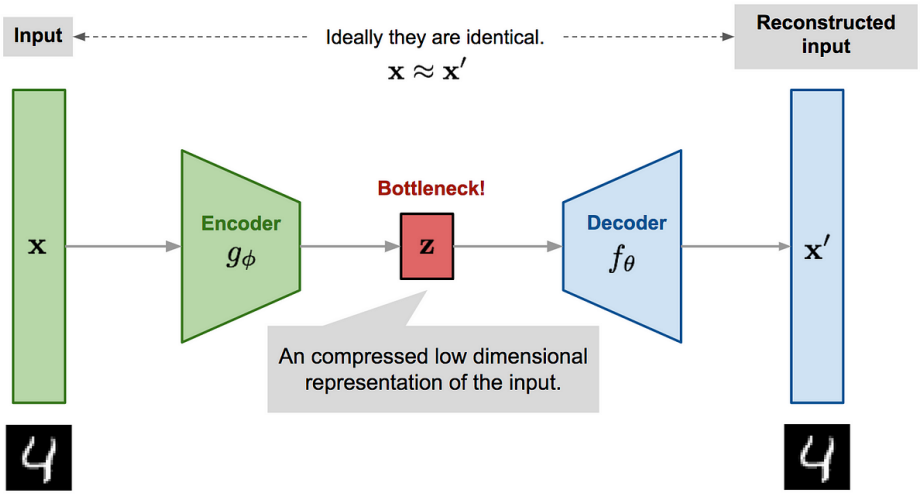

Figure 3. autoencoder architecture

what we have done :

we use an Convolutional Auto Encoder, which is a well-known architecture of neural networks, to perform this task. And finally, we evaluate the performance of the network with the top 50 shares of the US market. We use CAE to form feature matrix (X) with these 50 shares and finally perform clustering with $k-means$ (this time the input of $k-means$ is the feature matrix X, not the adjacency matrix A).

We used convolutional layers in our encoder and transposeConvolution layers in decoder of our model. We train the network with a batch size of 4 on 50 US shares. We repeat the training for a sufficient number of epochs to enable the network to produce output close enough to each input time series. Additionally, to ensure the number of epochs is sufficient, we first consider a high number of epochs and then use the Early stopping method in the training phase. The chart below shows the network's output (orange chart) for a specific input (blue chart) during training at different epochs. We observe that in later epochs, the network output becomes closer and closer to the input.

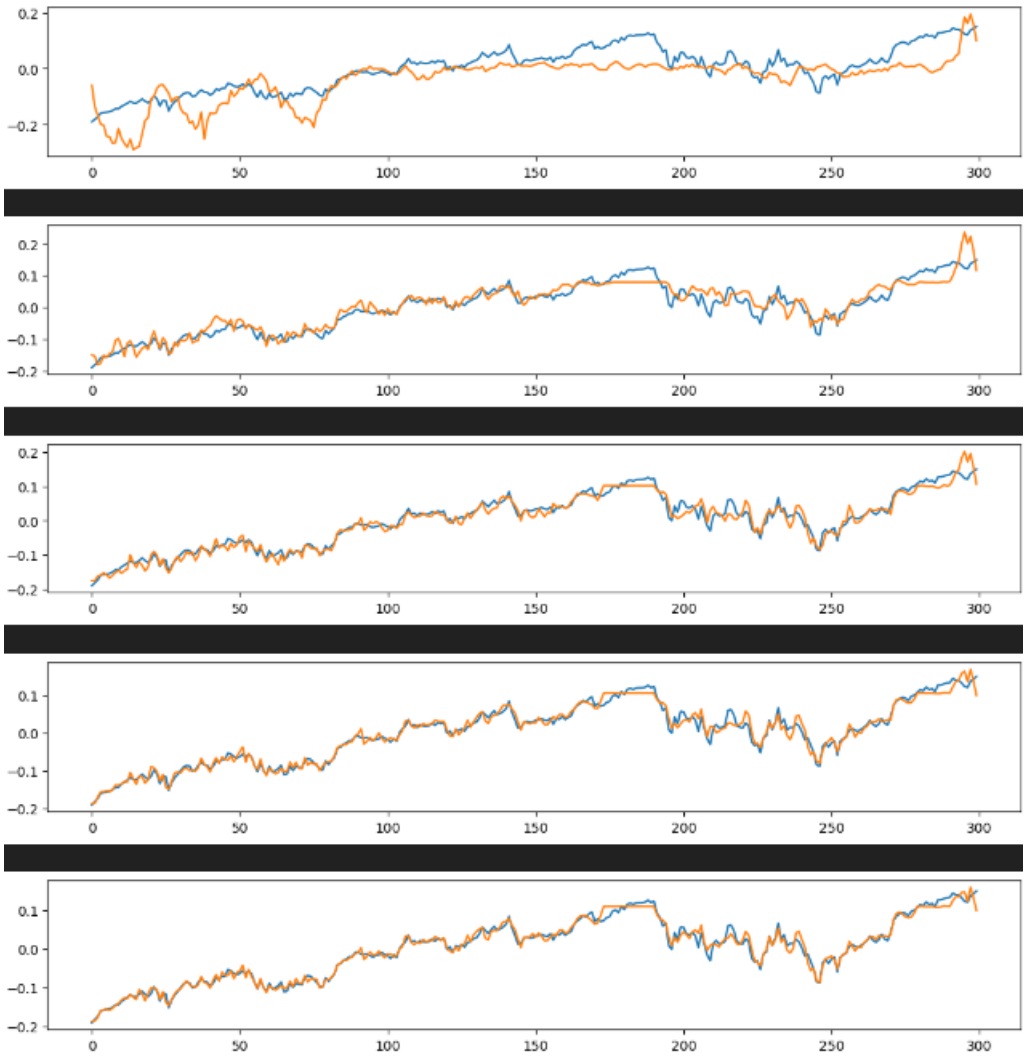

Figure 4. Comparison of network input and output in several consecutive epochs

The following figure shows the output of the network for different inputs after 200 epochs.

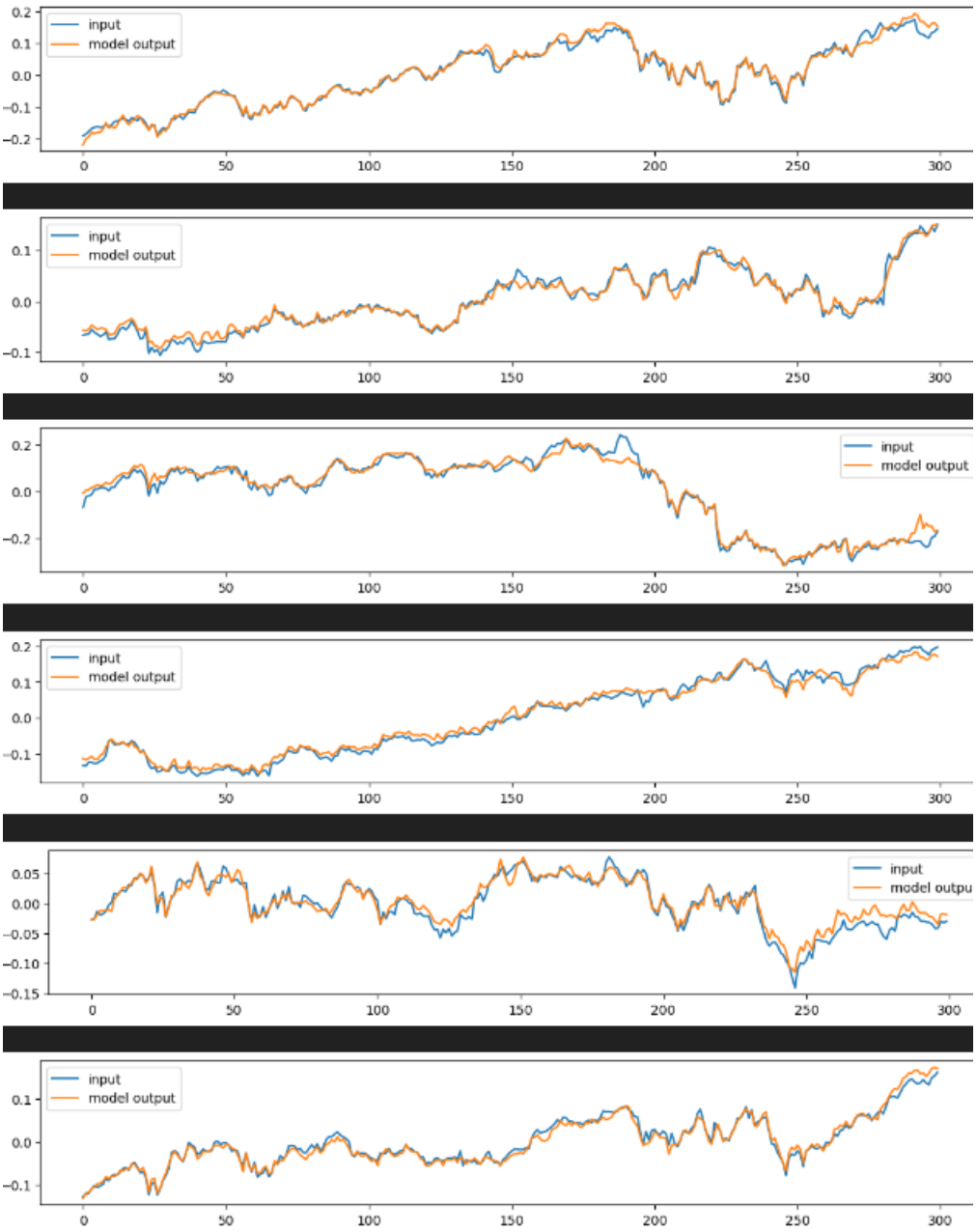

Figure 5. Comparison of network input and output for different inputs after 200 epochs

After completing the training phase, we disconnect the network decoder and use the output of the network encoder (latent vector z), which is a 64-dimensional vector, as the feature corresponding to each input for subsequent use.

Clustering

.1 Clustering with combination of classical methods using 4 type of A gathered from a stock market

In this section ,we regenerate adjacency matrix with 12 classical clustering algorithms like $K-means$ and $OPTICS$ method on 4 adjacency matrix witch obtained from 4 different method from Iran stocks and then the final clustering was performed with $K-means$ method on new adjacency matrix. The schematic of the algorithm is as follows:

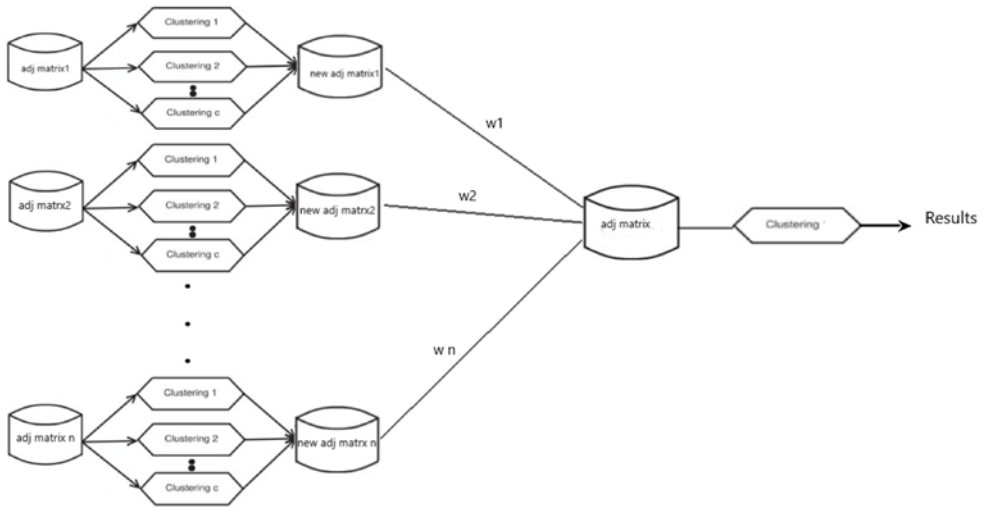

Figure 6. Architecture of (one-step) clustering mechanism . here $n = 4$, $c = 12$

the twelve clustering algorithm used is as follows

1. Hierarchical clustering
2. Agglomerative clustering
3. OPTICS
4. KMeans
5. Mean Shift Clustering
6. Spectral Clustering
7. Affinity Propagation
8. Birch

9. Gaussian Mixture Method
10. Ward's Method
11. SOM
12. Mini Batch KMeans

Hierarchical clustering is a method that aims to organize data into a tree-like structure, also called dendrogram, based on the similarity between samples. Agglomerative clustering is a type of hierarchical clustering that starts with each sample assigned to its own cluster and repeatedly merge the two most similar clusters until a stopping criterion is met.

OPTICS is another hierarchical clustering method that can handle non-monotonic structure and provides more flexibility in terms of cluster shapes.

KMeans is a partitioning algorithm that aims to divide data into a predefined number of clusters by minimizing the sum of squared distances between samples and their assigned cluster centers. Mean shift clustering is another centroid-based clustering method, but it does not require the predefinition of cluster number and can automatically estimate the optimal number of clusters.

Spectral clustering is a graph-based method that converts similarity measures into a weighted graph and applies spectral decomposition to obtain the clustering solution. Affinity propagation is a graph-based clustering method that assigns each sample to a representative sample, called exemplar, via message passing. Birch is a hierarchical clustering method that partitions the data into subclusters using a hierarchical tree-like structure.

Gaussian Mixture Method assumes that the samples are generated from a mixture of Gaussian distributions and estimates the parameters of these distributions to obtain the clustering solution. Ward's method is another hierarchical clustering method that aims to minimize the sum of squared differences between each sample and the center of its assigned cluster.

Self-organizing map (SOM) is an unsupervised learning method that reduces high-dimensional data into a low-dimensional map of neurons, where similar samples are assigned to adjacent neurons on the map. Mini Batch KMeans is a modified version of KMeans that uses a subset of samples at each iteration to accelerate the algorithm.

In summary, Hierarchical clustering and agglomerative clustering are suitable for exploring the hierarchical structure of the data, while KMeans and Mean Shift Clustering are efficient for clustering a large amount of data into a pre-defined number of clusters. Spectral clustering and Affinity Propagation are good choices when the data can be modeled as a graph, while Gaussian Mixture Method is suitable for data modeled as a mixture of Gaussian distributions. Ward's Method is suitable for finding clusters with tight boundary and SOM is useful for visualizing the data in a low-dimensional space. Mini Batch KMeans is an efficient variant of KMeans for large datasets.

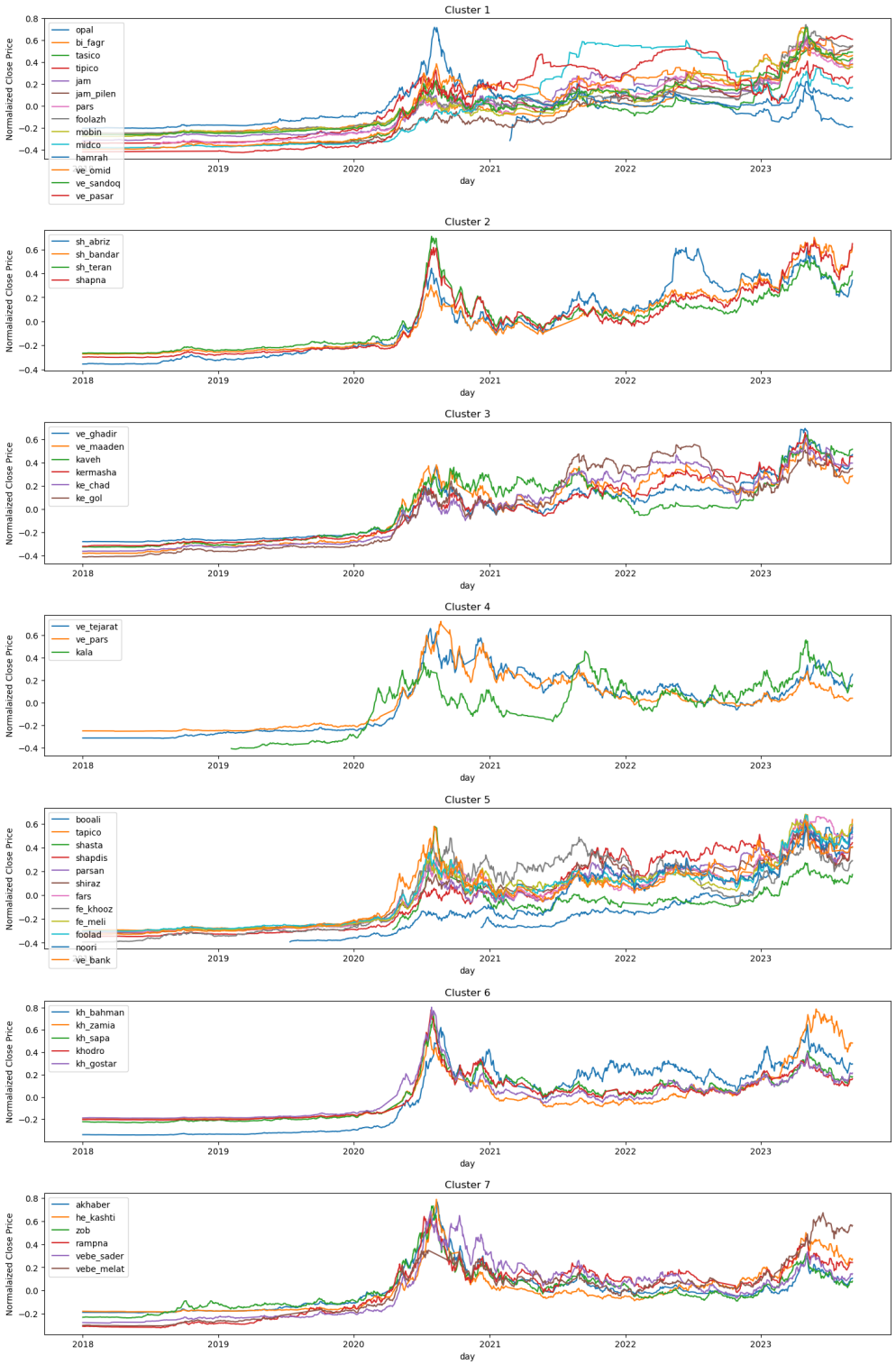

Figure 7. One-step clustering results (Final clustering was done by Kmeans method)

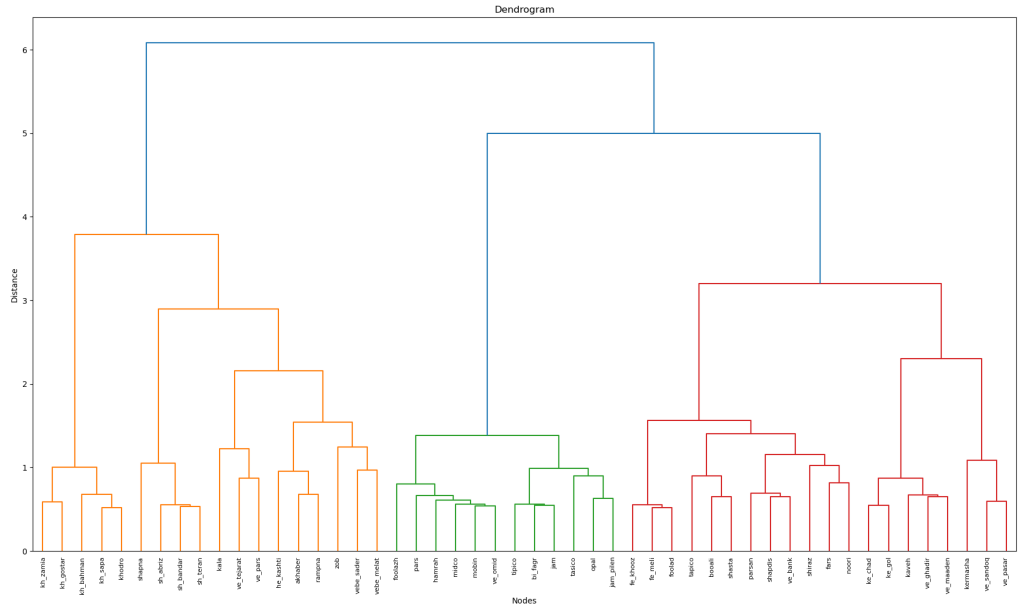

Figure 8. One-step clustering results (Final clustering was done by Ward's method)

In the following, we will repeat the algorithm twice as follows:

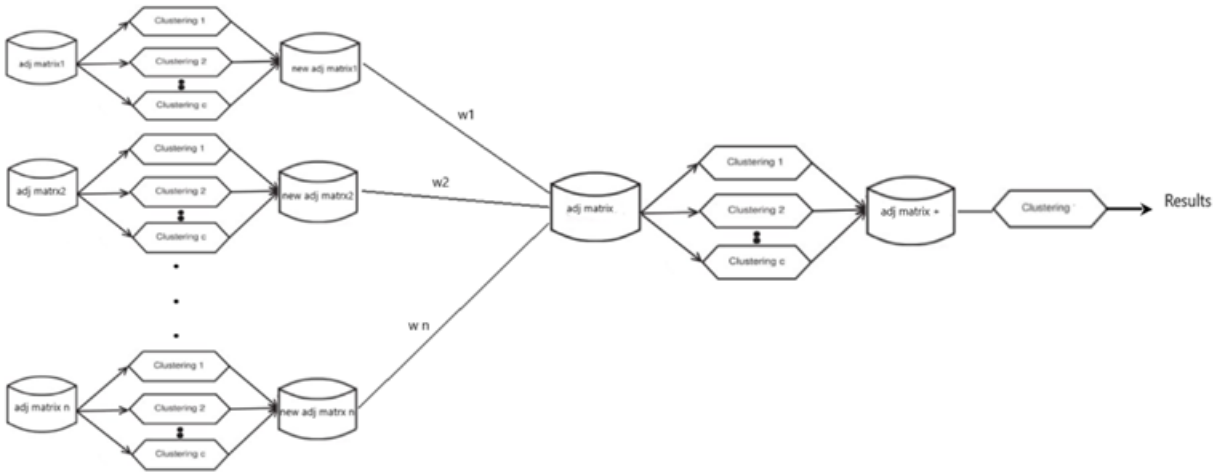

Figure 9. Two-step clustering architecture

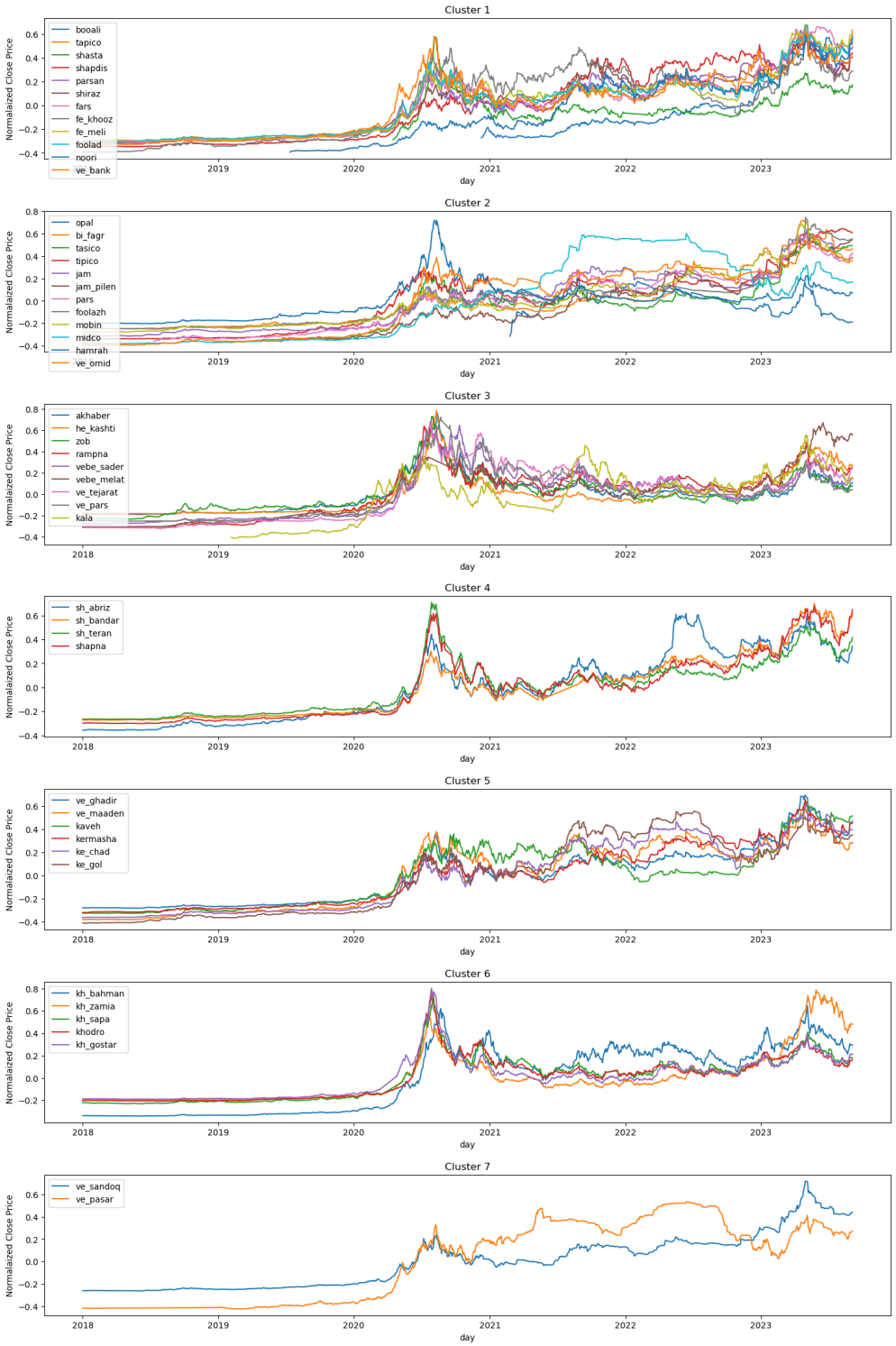

Figure 10. Two-step clustering results (Final clustering was done by Kmeans method)

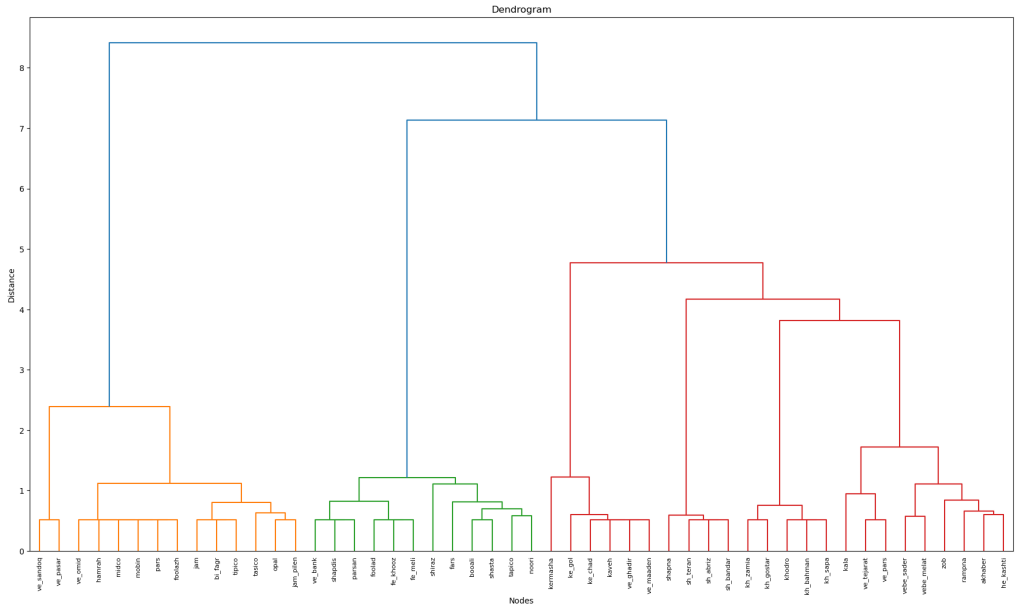

Figure 11. Two-step clustering results (Final clustering was done by Ward's method)

.2 Clustering using $A_{S\mathbf{deep}}$

The following results are obtained by performing k-means clustering using the adjacency matrix obtained from deep neural network on the top fifty large market shares of the US stock market.

Figure 12. clustering US stocks using $A_{S\text{deep}}$ with normalization (Final clustering was done by Kmeans method)

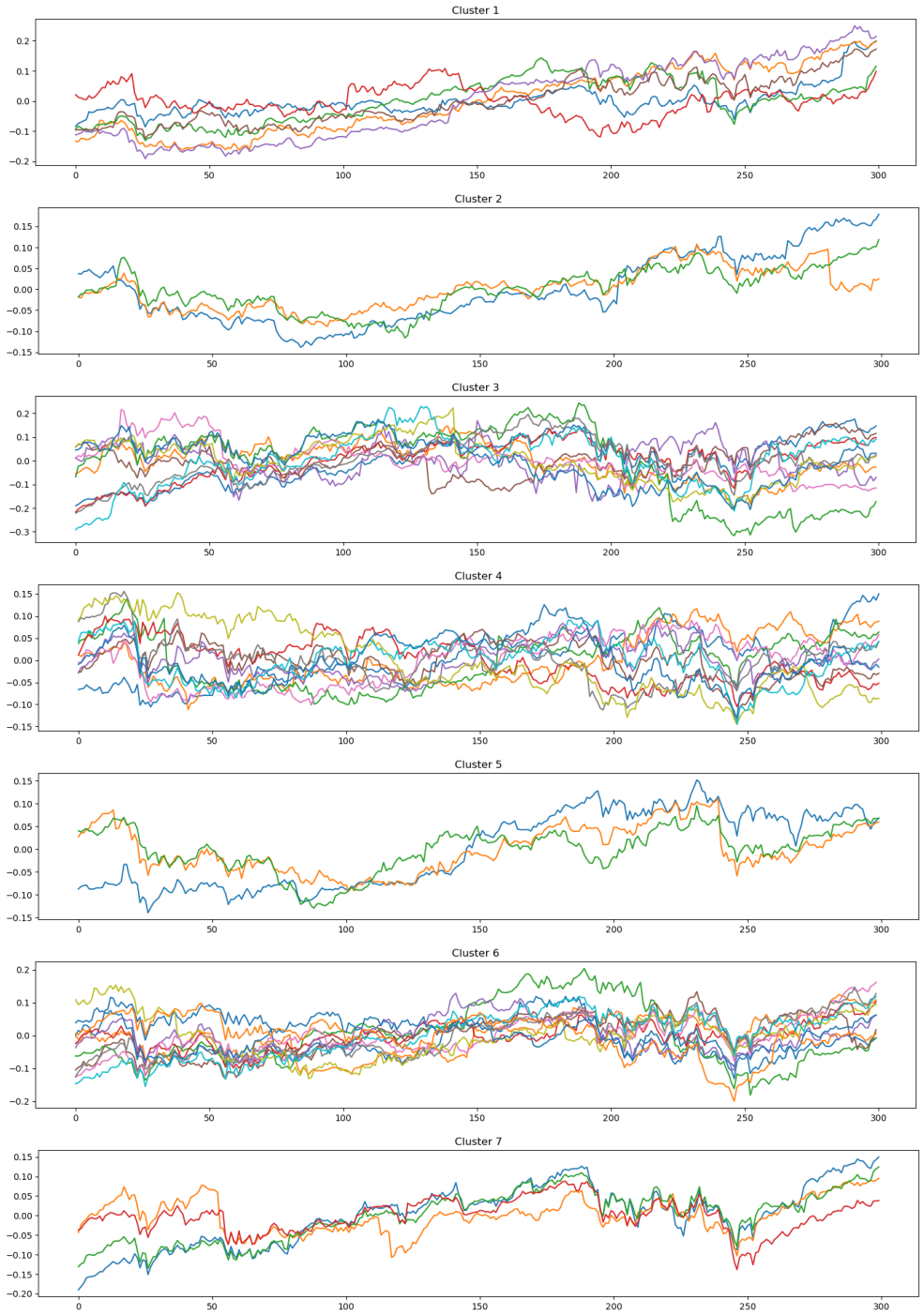

Figure 13. clustering US stocks using $A_{S\text{deep}}$ with return (Final clustering was done by Kmeans method)

.3 Clustering using feature matrix X

in this section, we performed clustering using $Kmeans$ method and extracted feature matrix (X) with a CAE which discussed in previous section .

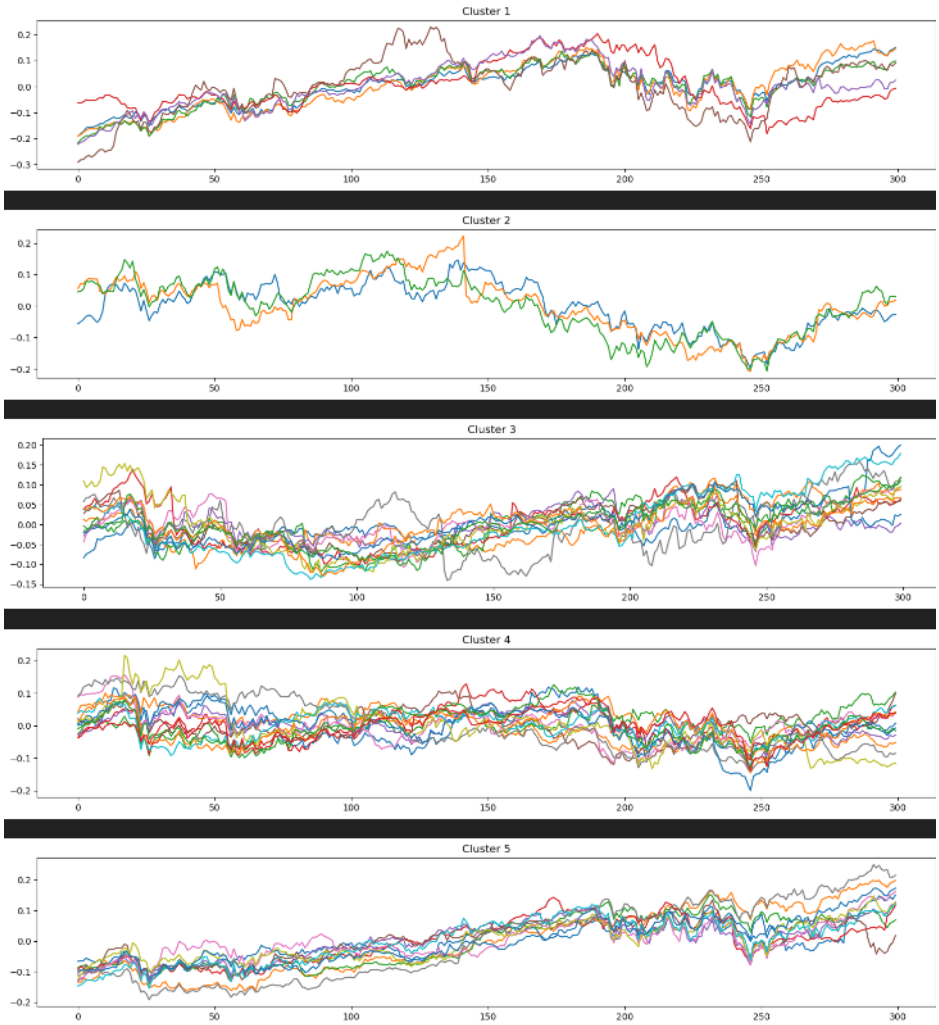

Figure 14. clustering top 50 US stocks using $Kmeans$ method and extracted feature matrix (X) with Convolutional Auto Encoder

It should be noted that the computational order of the CAE (for feature extracion) is $\mathcal{O}(n)$, unlike the convolutional network (introduced earlier) which is $\mathcal{O}(n^2)$, which is more suitable for clustering with a high number of shares.

4 future work and challenges

problems

.1 large scale input

We aim to resolve this issue by performing clustering and community detection on a large number of shares (approximately 6,000 to 8,000 shares) using two neural network architectures, GAN and GNN, which we will briefly introduce later. Our ultimate goal is to perform the initial clustering that is done on a large number of shares using these networks, and then perform clustering more accurately using the algorithms we have introduced so far, on the clusters created in the initial clustering stage, so that at the end, we can divide a large number of stock market shares into clusters with very close patterns to each other based on their correlation levels in the two-stage clustering.

.2 Miss-clustered share in result of clustering with similarity matrix extracted from the proposed CNN architecture

By clustering various stock markets, it was observed that in some clusters, there are some unrelated outlier time series (outlier) that, of course, is very few in number (in some cases one or two cases are not suitable for the appropriate category). Our idea to solve this problem is as follows:

We form initial clustering (with $Kmeans$) from the adjacency matrix obtained from the above network, and then from each category, we find several time series that have a similar pattern (using an algorithm used in the early stages of the project to find the distance between two stocks), and separate the time series that were not selected in this step. Then, we use the above network again with a minor change in its structure and its training :

This change is that instead of a linear layer in the output, we put a softmax layer that has the same number of neurons as the initial clusters we obtained from the output of $kmeans$ method. Now we give the separated time series (which currently belong to no category) to the trained network, and the final clustering is also performed by a convolutional neural network, whose results are also very desirable and expected.

$$\text{softmax layer: } f_j(\mathbf{s}) = \frac{\exp(\mathbf{s}_j)}{\sum_k \exp(\mathbf{s}_k)}$$

where $\mathbf{s}_j$ is the j-th element of $\mathbf{s}$

algorithms

.1 Generative Adversarial Network(GAN)

GAN stands for Generative Adversarial Network, which is a type of deep learning model that consists of two neural networks: a generator and a discriminator. GANs are used for generating new data that resembles a given training dataset.

The generator network takes random noise as input and generates synthetic data samples. Its goal is to generate data that is indistinguishable from the real data in the training set. The discriminator network, on the other hand, takes both real and generated data as input and tries to classify them correctly as real or fake.

During training, the generator and discriminator networks are trained simultaneously in a competitive manner. The generator tries to improve its ability to generate realistic data by fooling the discriminator, while the discriminator tries to improve its ability to distinguish between real and generated data.

In the context of community detection, GANs can be employed to learn a latent representation of the input data that captures the underlying community structure. The GAN consists of two main components: a generator and a discriminator.

The generator is responsible for generating synthetic data samples that resemble the real data. In the case of community detection, these synthetic samples can represent nodes or individuals within a community. The generator learns to generate samples that follow the same distribution as the real data, but with variations that allow for the exploration of different community structures.

The discriminator, on the other hand, aims to distinguish between real and synthetic data samples. It learns to differentiate between the real data samples, which represent nodes from different communities, and the synthetic samples generated by the generator. The discriminator's feedback is used to train the generator to generate more realistic samples that can fool the discriminator.

By training the GAN on a dataset that represents a network or graph, the generator can learn to generate synthetic samples that exhibit similar community structures as the real data. These synthetic samples can then be used for clustering or community detection purposes.

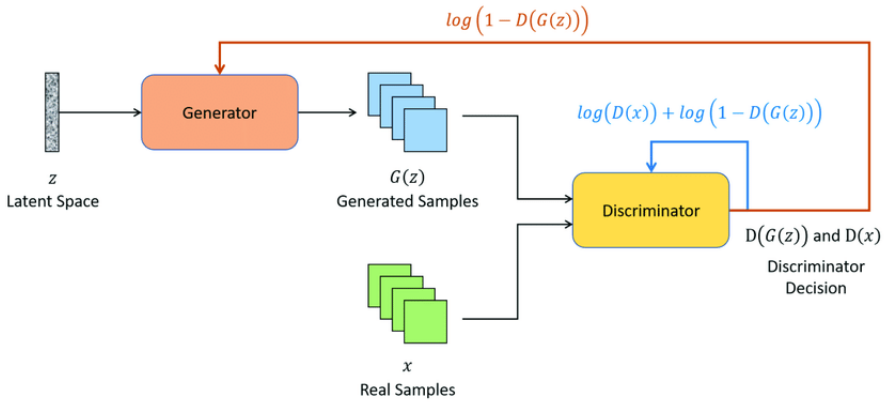

Figure 15

.2 Graph Neural Network(GNN)

GNNs are neural network architectures designed to process and analyze graph-structured data. They leverage the relational information present in graphs to capture complex dependencies and patterns. GNNs have shown great promise in various applications involving graph data, enabling more effective analysis and decision-making.

Node clustering using Graph Neural Networks (GNNs) is a method for grouping nodes together in a graph based on their features and connectivity patterns. The idea is to use GNNs to learn node embeddings that capture important structural and semantic attributes of the graph, and then use these embeddings to cluster nodes into groups that are similar to each other. In high-order node clustering, the goal is to group nodes together not only based on direct connections but also based on higher-order relationships, such as common neighbors or subgraphs. This can be achieved by extending the GNN architecture to incorporate higher-order features and features of the node neighborhoods, such as their degree or clustering coefficient. One approach for high-order node clustering using GNNs is to use a variant of the graph attention mechanism, which can capture both local and global information about the graph. In this method, the GNN iteratively updates the node embeddings based on the features of the nodes and their neighbors, while also considering higher-order relationships between nodes. Overall, high-order node clustering using GNNs is a powerful technique for analyzing complex graphs with rich structural and semantic information. It has potential applications in a wide range of fields, including social network analysis, recommendation systems, and bioinformatics.

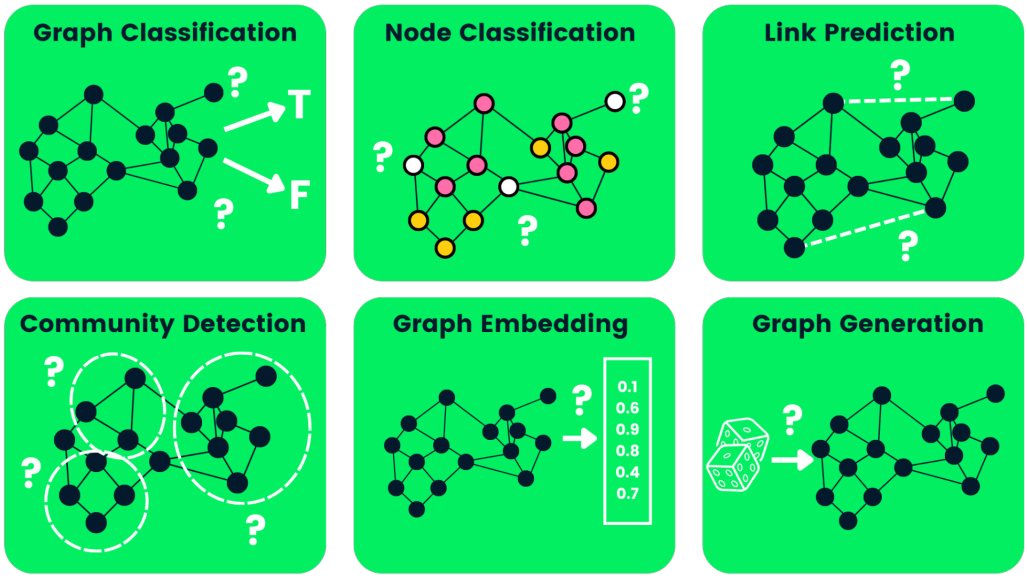

Figure 16. GNN aplication

References

[1] Xin Ding, Kuangrong Hao, Xin Cai, Xue-Song Tang, Lei Chen, and Haichao Zhang. A novel similarity measurement and clustering framework for time series based on convolution neural networks. *IEEE Access*, 8:173158–173168, 2020.

[2] Siyuan Gao, Xinyue Xia, Dustin Scheinost, and Gal Mishne. Smooth graph learning for functional connectivity estimation. *NeuroImage*, 239:118289, October 2021.

[3] Vassilis Kalofolias. How to learn a graph from smooth signals, 2016.

[4] Feng Zhao, Yating Gao, Xinning Li, Zhiyong An, Shiyu Ge, and Caiming Zhang. A similarity measurement for time series and its application to the stock market. *Expert Systems with Applications*, 182:115217, November 2021.