Research Article

An autoencoder-based deep learning approach for clustering time series data

Neda Tavakoli[1] · Sima Siami-Namini[2] · Mahdi Adl Khanghah[3] · Fahimeh Mirza Soltani[3] · Akbar Siami Namin[4]

Received: 4 November 2019 / Accepted: 23 March 2020 / Published online: 20 April 2020

Abstract
This paper introduces a two-stage deep learning-based methodology for clustering time series data. First, a novel technique is introduced to utilize the characteristics (e.g., volatility) of the given time series data in order to create labels and thus enable transformation of the problem from an unsupervised into a supervised learning. Second, an autoencoder-based deep learning model is built to model both known and hidden non-linear features of time series data. The paper reports a case study in which the selected financial and stock time series data of over 70 stock indices are clustered into distinct groups using the introduced two-stage procedure. The results show that the proposed methodology is capable of achieving 87.5% accuracy in clustering and predicting the labels for unseen time series data. The paper also reports an important finding in which it is observed that the performance of both techniques (i.e., autoencoder and Kmeans) are comparable. However, there are a few instances of time series data that are classified differently by the autoencoder-based methodology compared to the Kmeans algorithm. The results may indicate that the proposed deep learning-based approach is taking into account additional hidden features that might be overlooked by conventional Kmeans. The finding raises the question whether the explicit features of data should be analyzed for clustering or more advanced techniques such as deep learning need to be adapted by which hidden features and relationships are explored for clustering purposes.

Keywords Kmeans clustering · Financial data analysis · Time series clustering · Deep learning · Encoder–decoder · Unsupervised learning · Supervised learning · Encoder–decoder · Multi-layer perceptron

1 Introduction

An important step prior to performing any detailed data analysis is to understand the characteristics of a given data set. There are several statistical techniques that can help in creating different level of abstractions, each representing the data set from different angles. A very basic and prevalent technique is descriptive statistics such as mean and standard deviation, which are often utilized by data analysts in order to grasp the trend and variation of observations and thus capture a big picture of the data. These types of metadata can describe and, more specifically, "*featurize*" data. Hence, in practice and theory, data analysis refers to the identification, selection, and analysis of features of data sets.

A popular approach to conducting data analysis is through the conventional clustering problem, in which the given dataset is divided into subgroups. The goal is to maximize the similarity of the data observations grouped together; while maximize the dissimilarity of

✉ Akbar Siami Namin, akbar.namin@ttu.edu; Neda Tavakoli, neda.tavakoli@gatech.edu; Sima Siami-Namini, sima.siami-namini@ttu.edu; Mahdi Adl Khanghah, adl.mahdi1365@gmail.com; Fahimeh Mirza Soltani, soltani.fahimeh1364@gmail.com | [1]Department of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA. [2]Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409, USA. [3]Department of Computer Science, University of Debrecen, Debrecen, Hungary. [4]Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA.

SN Applied Sciences (2020) 2:937 | https://doi.org/10.1007/s42452-020-2584-8

the observations clustered in distinct groups. A simple and typical clustering algorithm (e.g., KMeans) takes as input a numerical vector representing the original data and measures the distances between data items using a simple distance metric (e.g., Euclidean). The assignment of data observations to different groups is then optimized with respect to the adjustment and optimization, which is a repetitive process. It is also possible to extend the basic clustering algorithm to address a more general multi-objectives problem where more than one feature, or characteristics, of datasets are taken into account for clustering.

1.1 The time series clustering (TSC) problem

As a very important and prevalent data type, time series data are playing a key role in different application domains such as social and human sciences including psychology, economics, business, and finance as well as engineering, quality control, monitoring and security. What makes time series data unique is the addition of another dimension to the complexity of data (i.e., time) and thus the elevation of the complexity involved in performing the analysis. The high dimensionality and additional complexity introduced by time makes the analysis of such data types very challenging. Therefore, it raises the concern of whether conventional data analysis techniques are suitable in exploring all "possible" features and factors as well as the causalities of such complex and inter-related data type. There are fundamentally several challenges associated with the problem of Time Series Clustering (TSC) including:

(1) *Unlabeled data* It is hard to identify or automatically label time series data. As a result, existing clustering techniques in supervised learning are less applicable. While there are great benefits associated with unsupervised learning (e.g., no need to know the exact number of desired clusters), the absence of labels in time series data would cause overlooking more accurate clustering techniques based on supervised learning, in which the labels of time series are known. As a result, it would be computationally more feasible to predict the cluster labels of time series data.
(2) *High dimensionality* Due to the inclusion of the time factor in the analysis besides some other features, the dimensionality of time series data and thus the number of features is increasingly high. Hence, it is of utmost importance to identify salient features that contribute significantly to the characteristics of the time series data and at the same time reduce the effects of nuisances that exhibit themselves as false features.
(3) *Hidden features* The most critical issue is the possibility of existence of some hidden features that may not be apparent and thus might be missed when conducting direct data analysis. Examples of such hidden factors might be exogenous and even some endogenous factors in the given data sets. The conventional data mining and even machine learning techniques are less effective in capturing these existence but hidden features. As a result, more advanced and rigorous methods and techniques are needed to take into account these possible features when modeling the clustering solutions.

This paper introduces a novel approach to time series clustering in which the problem is transformed into a supervised learning by automatically generating cluster labels for the time series instances. The cluster labels generated then are utilized to train a deep learning-based autoencoder to learn about the features of each time series data and thus cluster them with respect to the explored features.

1.2 General time series clustering approaches

A typical whole time series clustering consists of four major components: (1) dimensionality reduction, (2) distance measurement, (3) a clustering algorithm, and (4) a prototype definition and evaluation. As a special type of *whole time series clustering*, in feature-based clustering features of interest are extracted and the time series points are transformed (i.e., mapped) into a set of features. Feature extraction is used to compress large data sets using dimensionality reduction. In fact, in the feature-based clustering raw time series are transformed into a feature vector of a lower dimension. Then, a conventional clustering algorithm is applied on the lower dimension feature vector. The extracted features are usually application dependent, which implies that one set of features that are useful for an application might not be relevant and useful for another one. In some studies, other feature selection methods are performed to further reduce the number of feature dimensions after feature extraction [1]. As the notion of shape cannot be precisely defined, dozens of similarity (i.e., distance) measures have been proposed [2–5]. In this paper, we also employ similar techniques in order not only to cluster but also to utilize cluster labels for further analysis and more specifically for supervised learning. The time series clustering techniques are generally classified into six categories:

– *Hierarchical* In this approach, a hierarchy of clusters is generated using either agglomerative (i.e., bottom–up) or divisive (i.e., top–down) approaches. In the agglom-

erative method, each item is considered as a cluster and then appropriate clusters are merged together; whereas, in the divisive approach all the items are included in one cluster and then the cluster is split into multiple clusters. In hierarchical clustering, once the hierarchy is generated, it cannot re-adjust with any further changes. Therefor, the quality of hierarchical clustering is weak and some other clustering approaches are leveraged to remedy this issue.

- *Partitioning* In this approach, *k* groups of clusters are generated. One of the most common algorithms of partitioning clustering is called KMeans (i.e., *k*-means) clustering [6], where *k* clusters are generated and the mean value of all the elements within a cluster is considered as a cluster prototype.
- *Density-based* In this approach, a cluster is defined as a subspace of dense objects. One of the most common algorithms of density-based clustering is called DBSCAN [7], where a cluster is extended if its neighbors are dense.
- *Grid-based* In the grid-based clustering, the space is divided into a finite number of cells, called grids. Then the clustering is done on the grids. STING [8] and Wave [9] are two common grid-based clustering algorithms.
- *Model-based* In this approach, a model is used for each cluster. Then the best fit of data for the model is discovered. In the model-based clustering approaches, either statistical approaches or neural network methods can be used. An example is Self-Organizing Maps (SOM), which is a model-based clustering approach based on neural networks [10].
- *Multi-step* The Multi-step time series clustering refers to a combination of methods (also called a hybrid method), which is employed to improve the quality of cluster representation [11, 12].

1.3 The introduced two-stage time series clustering approach

This paper introduces a two-stage methodology for time series clustering. The clustering presented in this paper is a hybrid approach in which (1) partitioning-based, (2) model-based, and (3) multi-step time series clustering techniques are adapted with the goal of improving the clustering.

The first stage of the methodology targets the problem of "*unlabeled data*" in time series. The goal of this stage is to transform an unsupervised learning problem into a supervised learning and then be able to perform clustering and prediction of labels for time series data through supervised learning techniques. The basic idea is to derive the prospective cluster labels through utilization of characteristics and features of the given time series. Once the characteristics and features of time series data are "*vectorized*" (i.e., numerically calculated and represented), a conventional K-means clustering can be applied to cluster the feature vector data. The generated clusters and the associated label for each cluster can then be utilized to label the original time series data enabling the transformation of the problem to a supervised learning.

The second stage targets the problems of "*high dimensionality*" and dealing with "*hidden features*" of time series data. The proposed approach is to utilize deep learning to capture and take into account the effects of hidden layers. Deep learning is capable of optimizing a prediction model by iteratively learning and modeling new features through various internal neural networks and the neurons incorporated at each layer. To address both problems simultaneously, an autoencoder-based deep learning algorithm is utilized, in which the autoencoder not only takes into account the hidden features but also preserves the features that are salient for computation and prediction. Through changing the architecture of neural networks, including the shape of the input data, the number of internal layers, the number of neurons on each layer, and the activation and optimization functions it is possible to enhance the accuracy of the prediction, and more specifically, the supervised learning-based clustering problems through deep learning.

1.4 Contributions of the paper

The key contributions of this paper are as follows:

(1) Introduce a two-stage methodology to address the time series clustering problem. In the first stage, a methodology is introduced to create cluster labels and thus enable transforming a unsupervised learning problem into a supervised learning for time series data. In the second stage, an autoencoder-based deep learning algorithm is built to model clustering time series data.
(2) Demonstrate the performance of the proposed two-stage methodology through a case study performed on clustering time series data of over 70 stock indices. According to the results of the case study, the proposed two-stage clustering technique achieves an accuracy of 87.5% in correctly predicting the cluster labels of time series data.
(3) Compare the clustering results obtained by the conventional KMeans algorithm and the proposed two-stage methodology. It was observed that there are some discrepancies between the clustering results obtained by these two approaches implying that the deep learning-based approaches take into account

additional hidden features when clustering time series data.

1.5 Structure of the paper

This article is organized as follows: Sect. 2 highlights the key characteristics of time series and in particular financial time series data. A brief overview of artificial neural networks is represented in Sect. 3. The general picture of the two-stage model is presented in Sect. 4. The encoder–decoder deep learning model is described in Sect. 5. The introduced algorithms are presented in Sect. 6. The autoencoder-based model is evaluated and the results are reported in Sect. 7. Section 8 concludes the paper and highlights the future research directions.

2 Feature vectors of time series as data labels

This section first reviews the general characteristics of time series that can be used as features, and then explores financial time series and their unique characteristics along with a short description of the most representative features of financial time series data: *volatility* and *return*.

2.1 Common general components of time series data

General time series data are often analyzed with respect to certain features and components. This section briefly presents some known features of time series:

- *Seasonality* Seasonality is a periodical pattern observed for time series. It is the effects of seasons such as months or fiscal year on the volatility and the volume traded within a period of time. For instance, it is expected that the price of crude oil usually is elevated in the beginning of cold seasons.
- *Cycle* Cycle is a dynamic pattern observed over a period of time (e.g., year). For instance, it is expected to observe some cyclic behavior during harvesting time (e.g., cotton harvesting time).
- *Trend* It is a long-term movement in a given time series without considering time or some other external influential factors. For instance, it is expected that the number of individuals, who purchase new Apple product increases. However, this trend will be slowly disappearing over time if another new or better product introduced into the market.
- *Irregular features* These types of components are unpredictable. These features are often calculated or retrieved after trend-cycle and seasonal components are removed from the time series. The remaining parts are unpredictable, since they only represent non-cyclic and the characteristics that are unique to the underlying time series.

These features are the major tools for analyzing general time series data. More special time series data such as those related to financial markets have their own unique features, which are discussed in the following section.

2.2 Common features of financial time series data

The financial time series data can be characterized through certain features and patterns [13]:

- *Dependence* There exists a positive autocorrelation in stock return indices, but this autocorrelation is largely insignificant.
- *Distribution* The annual returns follow a normal distribution. Likewise, the security returns are non-stationary and also follow a normal distribution with fat tails.
- *Heterogeneity* The distributions of financial returns are non-stationary. Moreover, the standard deviation of returns is not constant over time.
- *Non-linearity* The models built for time series are mostly non-linear in mean and variance.
- *Scaling* Unlike physical objects, there are no constants or absolute sizes in economics. As a result, there is no characteristic scale in economics and finance and thus financial markets demonstrate non-trivial scaling properties.
- *Volatility* It is the standard deviation of the change in the values of a financial time series data. Volatility is often used to demonstrate the risks associated with stock indices.
- *Volume* It refers to the level of trading of a stock index over a given time period in the market. This feature may have some correlations with calendar and seasonal effects.
- *Calendar effects* The seasonal or calendar effects are periodical anomalies or patterns that are observed in returns. There are several different types and flavors of calendar effects such as the weekend effect, the January effect, the holiday effect, and the Monday effect.
- *Long memory* There is a chance that the stock market's returns and volatility exhibit long memory properties indicating that the observed returns are dependent over time. The chance highly depends on the type of the market.
- *Chaos* This feature exists when a dynamic system exhibits some sensitivity to initial conditions and thus reacts to unpredictable long-term behavior. There exists some

small evidence of low-dimensional chaos in financial markets.

The classical data mining techniques and even primitive machine learning algorithms might not be able to capture all these features and thus they may generated a model that might be less accurate. As discussed and presented in this paper, more advanced techniques such as deep learning approaches are better well-positioned to model these features through layers of learning.

2.3 Financial time series feature vector: <*volatility, return*>

In finance, volatility, also known as swings, refers to the degree of variation of a trading price series such as S&P 500 index over time. It is calculated by the standard deviation of logarithmic returns. More specifically, volatility shows the frequency and severity, in which the market price of an investment fluctuates. The stock volatility shows uncertainty of the future of the economic and financial series. The expectation of the future of economic and financial behaviors highly contributes in changing the stock volatility.

For calculating volatility, we first need to provide returns. The return of a stock in a given time period can be defined as the natural logarithm of the closing price (or other series such as opening or adjusting price) at the end of the period divided by the closing price of the stock at the end of the previous period. The general equation for calculating return is as follows:

$$r_t = ln\left(\frac{C_t}{C_{t-1}}\right) \tag{1}$$

where

- r_t is the return of a given stock over the period,
- ln is the natural log function,
- C_t is the closing price at the end of the period, and
- C_{t-1} is the closing price at the end of the last period.

For calculating the volatility, we need to calculate the standard deviation of the returns. Standard deviation is the square root of variance, which is the average squared deviation from the mean as follows:

$$\sigma = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_t - \mu)^2} \tag{2}$$

where

- r_t is the return of a given stock over the period,
- μ is the average of the returns, and
- σ is the square root of variance.

As an example, VIX (Chicago Board Options Exchange Market Volatility Index) is a popular measure of the implied volatility of S&P index options. If there is a wide range of fluctuations in the prices over short time, it means that there is high volatility and vice versa. On the other hand, if the price moves slowly, there is low volatility [14].

The importance of volatility and returns and the trade-off between these two stock indicators has received tremendous attentions. In practice, investors invest in the stock markets with an expectation of getting returns, which in turn involves risks or the volatility of asset returns. In fact, the trade-off between return and risk is the conceptual framework in the asset-pricing models.

There has been a large body of literature on transmission of stock returns and volatility. Most asset-pricing models indicate a positive trade-off between expected returns and volatility. On the other hand, there are some research studies, in which empirical evidence supports a negative relationship between returns and volatility [15–17]. For example, Chung and Chuwonganant [18] found that market volatility affects returns through stock liquidity, suggesting that liquidity providers play an important role in the market–return relationship in the United States. Sen and Bandhopadhyay [19] evaluated a dynamic return and volatility spillover from the US stock market into the Indian stock market. These conflicting results warrant further estimation by using appropriate techniques and algorithms.

The volatility clustering is the main feature of volatility of asset prices and the volatility shocks can affect the expectation of volatility in future [19]. The volatility clustering means the large changes of prices (variance of return) for a period.

There is a double relationship between volatility and returns in equity markets. The long run fluctuations of volatility show risk premiums and therefore it establishes a positive relation to returns. On the other hand, short run volatility indicates news effects and shocks to leverage, and thus produces a negative volatility-return relation. The leverage effect explains how the volatility rises when the asset prices reduces. While long run volatility is related with a higher return, the opposite appears in the short run volatility.

3 Artificial neural networks: a brief review

There are several different types of deep learning-based neural networks including convolutional neural networks (CNN) and recurrent neural networks (RNN). In particular, the Long Short-Term Memory (LSTM) network architecture

[20–22] can be of interest for analyzing time series data. This paper provides a general background related to general concept of ANNs and, more specifically, autoencoders.

3.1 Artificial neural network (ANN)

A typical neural network consists of different layers: (1) an input layer, (2) one or more hidden layers, and (3) an output layer. The nodes or neuron on each layer usually represent the number of features and thus the dimensionality of the dataset. The neurons are mapped through some links called "synapses" to the nodes created in the hidden layers and then to the output layer. The synapses links are associated with some weights that represent the significance of the value hold by every node. The weights help in decision making in order to decide which feature should be considered and thus should pass through the next layers. The weights also demonstrate the strength of the features to the hidden layer. A neural network is capable of adjusting the weight for each synopsis, a process that is usually called learning through optimization.

The nodes in the internal layers utilize some activation functions such as *sigmoid* or tangent hyperbolic (*tanh*) on the weighted sum of inputs and then transform or map the inputs to the outputs that hold the predicted values. Once the weights are adjusted, the output layer creates a vector of probabilities for different outputs and chooses the one with minimum error rate. In the case of multi-labels clustering problem, i.e., clustering with more than two outcomes, a *SoftMax* function can be utilized, in which it minimizes the differences between the expected and predicted values.

The learning process is an iterative task by which the assignments and weights are repeatedly adjusted to with the goal of minimizing the errors obtained through the network training. To find the most optimal values for errors, the errors are "*back propagated*" into the network from the output layer towards the hidden layers. As a result, the weights are adjusted. The procedure is repeated several times with the same observations and the weights are re-adjusted until there is an improvement in the predicted values and subsequently in the cost. When the cost function is minimized, the model is trained.

3.2 Encoder–decoder

An autoencoder is a type of neural networks that transforms input data into their output. An autoencoder consists of two parts in this transformation [23]:

(1) The *Encoder* that transforms the high dimensional inputs into a smaller set of dimensions while keeping the most important features, and

(2) The *Decoder* that uses the reduced set of features to reconstruct the initial input data.

The output of an encoder, referred to as "*latent-space representation*", is a compressed form of the input data, in which the most influential and important features are kept. The output of the encoder is then utilized to reconstruct the initial input data given to the autoencoder.

From a mathematical point of view, an autoencoder network is a composition of functions $(fg)(x)$. More specifically, an encoder is a function f that takes x as input and maps x into h, or the latent-space representation (i.e., $h = f(x)$). On the other hand, a decoder is a function g that takes the output of the encoder (i.e., h) and produces r (i.e., $r = g(h)$). The objective is to make r as close as possible to x.

The key objective of autoencoders is not just to copy the input into the output. In fact, through the training of an autoencoder and transformation of the input into the output, it is aimed that the produced latent-space representation (i.e., h) holds only unique and important properties and features of the dataset that can be inspected for further analysis. In order to extract the only important features of the given dataset in the form of latent-space representation, a set of constraints can be defined on the function that generates h so that the resulting compressed form of the dataset has smaller dimensions than initial dataset x. As a result, the quality of detecting most salient features of the dataset x heavily depends on the constraints defined on h. There are different variations of autoencoders [23]:

(1) *Basic autoencoder*, in which there are three layers: a) an input layer of size $|x|$, b) a hidden layer of size $|h|$ (i.e., $|h| < |x|$), and b) an output layer of size $|r|$ (i.e., $|r| = |x|$) where size refers to the number of nodes incorporated and designed in the underlying layers.
(2) *Multilayer autoencoder*, in which the number of hidden layers is increased to more than one. This type of autoencoders is useful when additional internal hidden layers are required to extract the hidden features and train the model.
(3) *Convolutional autoencoder*, in which the input data is filtered for the goal of extracting only some parts of it. These types of autoencoders are particularly very effective in image processing applications and conversions from 3-D images into smaller dimensions of filtered images.
(4) *Regularized autoencoder*, in which the extraction and training stages are performed in accordance with some other factors such as loss functions than solely based on defining hidden layers.

3.2.1 The mathematical background of encoder–decoder

In an autoencoder neural network, the network is split into two segments: the encoder and the decoder.

$$\begin{aligned} &\phi : \mathcal{X} \rightarrow \mathcal{F} \qquad (Encoder) \\ &\psi : \mathcal{F} \rightarrow \mathcal{X} \qquad (Decoder) \\ &\phi, \psi = arg \quad min_{\phi,\psi} ||X - (\psi \quad o \quad \phi)X||^2 \end{aligned} \tag{3}$$

The encoder function ϕ maps the original data $\mathcal{X}$, to a latent space $\mathcal{F}$ of reduced dimensionality. On the other hand, the decoder function ψ maps (i.e., reconstruct) the latent and reduced space $\mathcal{F}$ to the output. In our case, the output is the same as the input data $\mathcal{X}$. More specifically, the encoder–decoder pair is trying to reconstruct the original data and its shape after performing and capturing some generalized non-linear transformation of the data.

The encoding part of the network can be represented by the standard neural network function passed through (1) an activation function σ, (2) a bias parameter b, and (3) the latent dimension z.

$$z = \sigma(Wx + b)$$

In an analogous way, the decoding part of the neural network can be represented similarly, but with different activation functions, bias, and weight.

$$x' = \sigma'(W'z + b')$$

The loss function $\mathcal{L}$ for this combined neural network can be expressed in terms of these encode and decoder network functions.

$$\mathcal{L}(x, x') = ||x - x'||^2 = ||x - \sigma'(W'(\sigma(Wx + b)) + b')||^2 \tag{4}$$

Accordingly, this loss function $\mathcal{L}$ will be used to train the neural network through the standard back propagation procedure. The goal of the autoencoder is to select the encoder and decoder functions such that minimal information are encoded that be can regenerated by the decoder with minimal loss.

4 A synergic method for time series clustering

Figures 1 and 2 depict the proposed two-stage methodology for time series clustering. The methodology first enables supervised learning by generating cluster labels for the given time series data using conventional KMeans clustering, and then it uses the generated labels for the purpose of clustering. The steps of the two-stage synergic methodology are as follows:

(a) Stage I: Label Generation

(1) Capture the characteristics and descriptive metadata, as features, and build the feature vectors $< f_1, f_2, ..., f_n >$ for each time series data.
(2) Apply the conventional KMeans clustering on feature vectors and identify cluster groups.
(3) Utilize the class groups and their identifications (i.e., tags) as labels for each time series data.
(4) Provide the time series data, their feature vectors, and their generated labels to Stage II and thus transform an unsupervised learning to a supervised learning problem.

(b) Stage II: Autoencoder-based Clustering

(1) Build an autoencoder-based deep neural network with some hidden layers and neurons, i.e., nodes, in which:
   - The number of nodes on the inner most layer represents the number of clusters,
   - The number of nodes on the input layer represents the feature vector and its size,
   - The number of nodes on the output layer represents a probabilistic value showing the clustering label for each data set.
(2) Split the constructed "labeled" time series data into test and train datasets.
(3) Train the autoencoder-based neural network with the train dataset.
(4) Cluster and predict the labels of the test dataset using the trained neural network.

The encoding part of the model is responsible for identifying important features of the time series data. The encoder then reduces the number of features from the whole set to the selected and most important features of the time series data. On the other hand, the decoder takes the reduced set of important features and tries to reconstruct the initial values without losing any information significantly. The pair of < encoder, decoder > then will form a mechanism to reduce the dimensionality of the time series data for the purpose of clustering. In fact, the encoder and decoder pair only reduces the feature space and then the selected and reduced features are utilized to perform clustering.

In the following sections, we provide an in-depth description of the synergic methodology proposed for clustering time series data. First, we focus on the architecture of the designed autoencoder and then provide in-depth discussion of the algorithms developed.

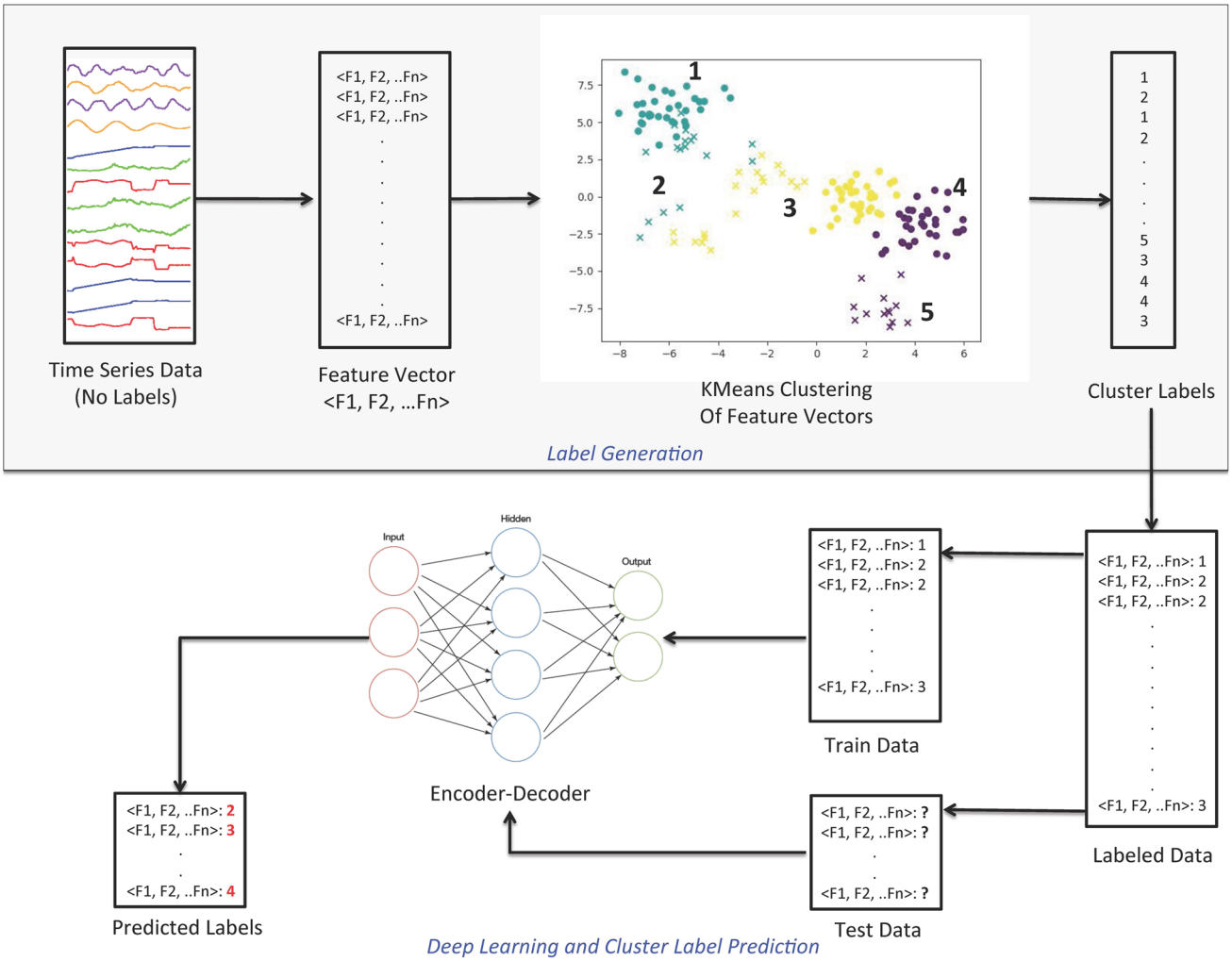

Fig. 1 A synergic methodology for time series clustering

5 An encoder–decoder for learning features of time series data

Figure 3 illustrates the architecture of the encoder–decoder neural network developed for feature learning of time series data.

The network includes two layers representing input x and output r, respectively. The input layer is designed with two neurons (i.e., nodes) where the neurons in the input layer represent the volatility and return for each stock index (i.e., $< Volatility, Return >$, the selected features for our financial case study); Whereas, the output layer is designed with one neuron, by which a probabilistic value will be calculated to represent the cluster label.

On the encoding part, there are three internal layers L1, L2, and L3, each with the number of devised neurons of 100, 50, and 20, respectively. Since the ultimate goal of an encoder is to reduce the dimensionality of a given input, the number of nodes incorporated in these internal layers is in descending order implying the reduction of the features and preserving only those that stand out and are salient. Note that the explicit features given to the authoencoder are in the form of $< Volatility, Return >$. However, the purpose is to detect and take into account hidden features that might exist even within volatility and return when modeling the deep learning-based clustering.

Since an autoencoder is a symmetric neural network, the number of layers and nodes on each layer of the decoder should be symmetric with the number of layers and neurons in the encoder side. As a result, there are three hidden layers on the decoder side (i.e., L4, L5, and L6) with the increasing number of nodes of 20, 50, and 100, respectively, in which an ascending order of the number of neurons is apparent. The decoder side explicitly reconstructs the original inputs using the reduced features with the exact shape for both input and output. The layer h is exactly where the number of prospective clusters for clustering data is taken into consideration. In our financial case

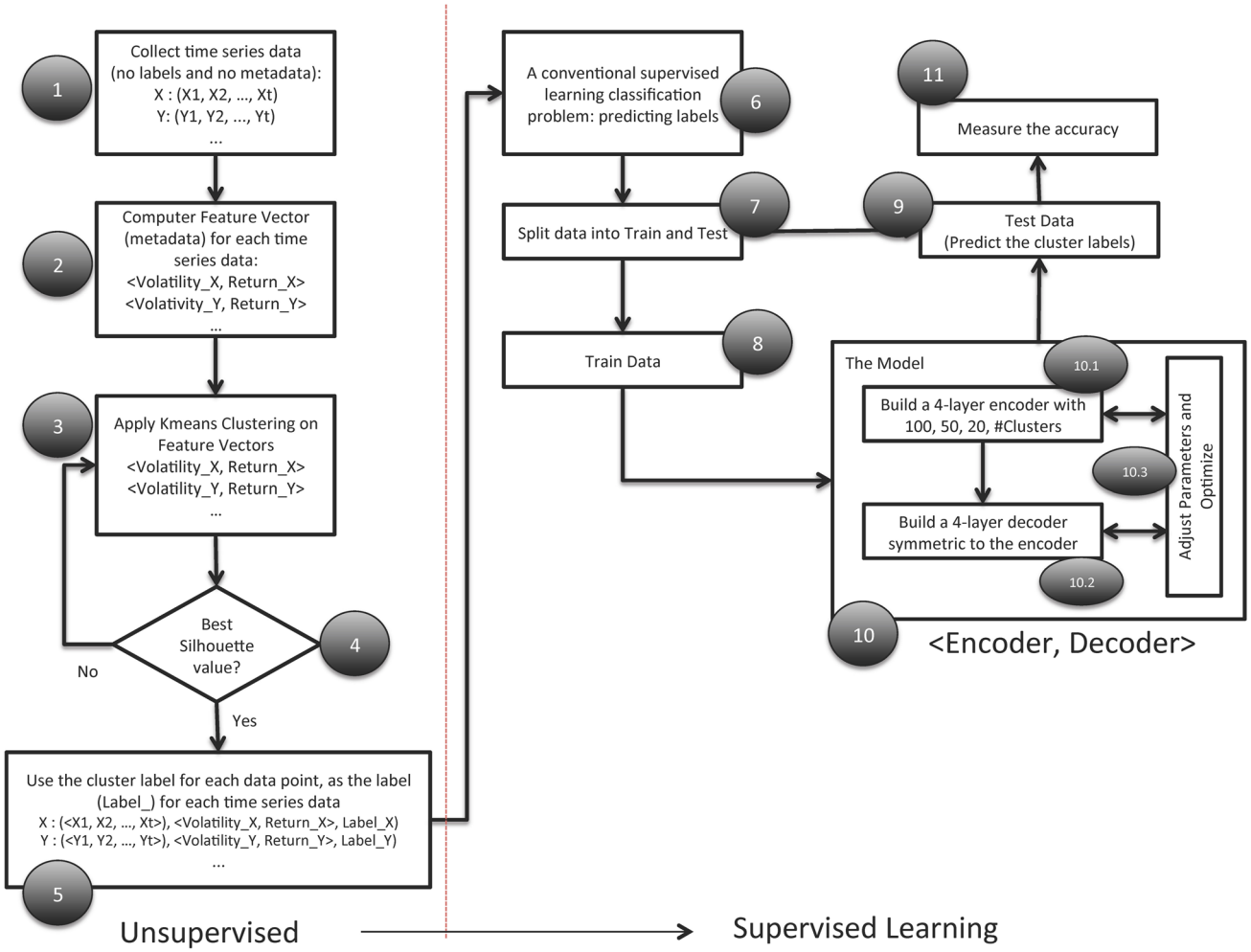

Fig. 2 The flowchart of the introduced timer series clustering

study, the optimal number of clusters is four (See Sect. 7 for analysis on capturing the optimal number of clustering using Silhouette value) and thus the number of nodes on this layer (i.e., *h*) is also considered four.

As it is apparent from Fig. 3, through the number of nodes and layers defined for the encoder part, the most salient features are captured and through the decoder side, which is symmetric to the encoder side, the exact shape of the input data is reconstructed. The adjustment of weights for the internal layers and their nodes are decided and optimized in a repetitive manner, where the loss function on the output (i.e., reconstructed input) is used as a means to measure the accuracy of the clustering.

The activation function incorporated on the layer *h* is in the form of a "*sigmoid*" function. A sigmoid function is used to predict the probability values, since its values range between (0 to 1). Once the model is trained on training set, the model (i.e., the output layer) produces a "*floating*" value in the range of $[1 - C, C - 1]$ where C is the number of desired clusters (i.e., four in our case study). A simple application of rounding (i.e., `np.rint` function in Python) and absolute (i.e., `np.absolute` function in Python) functions to the output generated by the model will produce a "positive integer" value between $[0, C - 1]$ that represent the class label of the underlying stock index data for which the output has been generated.

6 The algorithms

The introduced autoencoder-based deep learning methodology for time series clustering is represented through two algorithms: (1) Transforming unsupervised data into supervised through building feature vectors and characterizing time series using descriptive metadata (i.e., volatility and return), and (2) Building an autoencoder-based deep learning to predict cluster labels of supervised stock data. The standard form of K means and encoder–decoder based algorithms

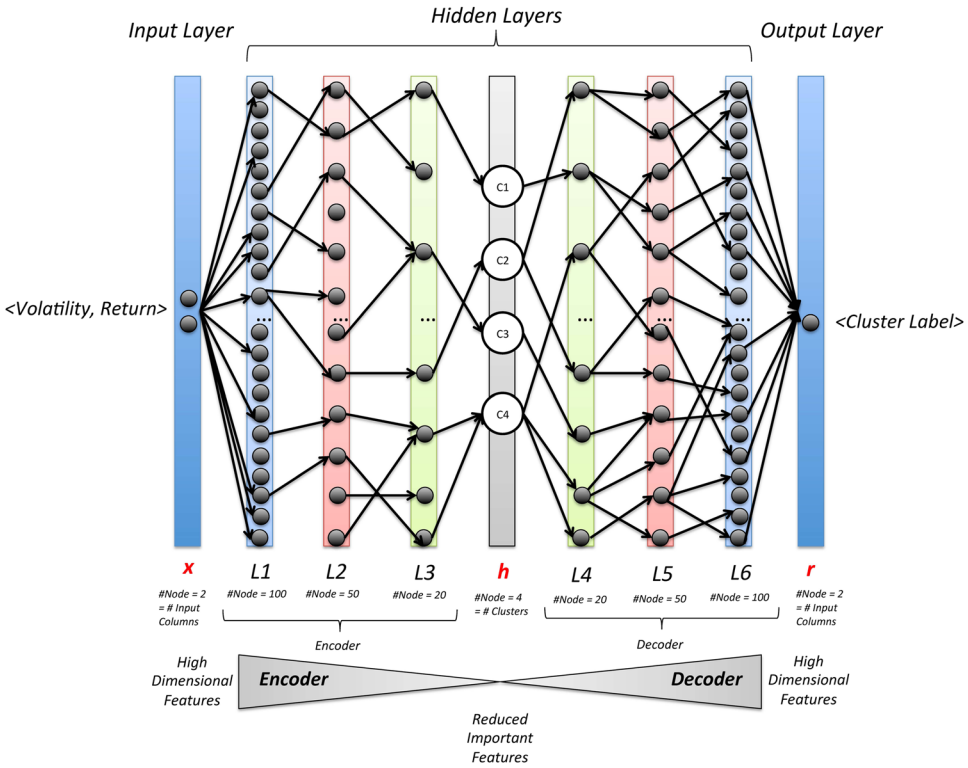

Fig. 3 The designed encoder–decoder architecture

are adapted to solve the time series clustering problem. These algorithms are well known and they have been implemented and integrated together for the purpose of addressing the time series clustering problem. In the following sections, we describe each algorithm in further details.

6.1 Algorithm 1: Supervised learning through characterizing time series and utilizing metadata as labels

Algorithm 1 presents the first stage of the methodology in which the conventional KMeans clustering algorithm is used to determine the cluster label for the time series data. The algorithm utilizes two descriptive concepts to characterize financial time series data: (1) volatility, and (2) return. Therefore, a vector of $< volatility, return >$ for each array of stock prices captured for each stock index and for a given period of time will be computed and constructed.

Algorithm 1 takes as inputs (1) a URL to scrap and enumerate stock indices, (2) the desired number of clusters for clustering stock indices, (3) the number of stock indices to analyze and cluster, and (4) the start date of stock prices. The algorithm then labels each stock index with respect to the cluster the underlying stock index belongs to by utilizing characteristic of time series data (i.e., volatility and return) and then transforming unsupervised time series data into supervised data. The first stage of the methodology (i.e., Algorithm 1) consists of several part itself.

Algorithm 1: Transforming unsupervised to supervised learning (metadata of time series as labels).

Result: The cluster label for each stock ticker.

```
initialization and setting;
retrieving online data of the target stock tickers;
compute volatility and returns for each stock ticker;
build <volatility, returns> vector;
while No. Clusters <= 10 do
    apply Kmeans(No. Clusters) on <volatility, returns>;
    measure Silhouette value for each clustering;
end
select clustering with the best Silhouette value;
use cluster label for each data point as data label;
report the cluster label for each stock ticker;
```

First the setting variables are initialized followed by the declaration of a few data structures to hold captured data. The algorithm then proceeds with scrapping the given URL and listing the stock indices (i.e., tickers) in order to perform cluster analysis. Once the list of stock tickers is prepared, the "Adjusted Close" price of each stock index is retrieved for a given time period (i.e., from the date indicated as input until the present date). For our case study, we retrieved data for the start date of January 1, 2019 to April 15, 2019 (the day of running this experiment and capturing the data).

As the two key characteristic features of time series, the volatility and return values are computed for each time series of prices for each stock index and the computed values along with the stock indices then are preserved in a data structure.

A series of KMeans-based clustering algorithm with respect to the number of desired clusters is then built and the best clustering is identified, with respect to the Silhouette value. The captured $< volatility, return >$ are then given to the clustering model in order to cluster and then create cluster labels. The labels for each stock index is created, representing the cluster they belong to (i.e., 0, 1, 2, 3). The data are then fed to Algorithm 2 to build an autoencoder. The codex of the algorithm is given in "Appendix A" Listing 1 for interested readers.

6.2 Algorithm 2: Predicting cluster labels of time series data through autoencoder-based deep learning

The second part of the methodology builds an autoencoder-based deep learning for clustering stock indices. The algorithm takes as inputs: (1) training labeled time series data, (2) testing unlabeled data, (3) number of clusters (i.e., neuron or node) to encode, (4) the shape of the input data (i.e., 2 in our case $< volatility, return >$, (5) the shape of the output data (i.e., 1 in our case, a floating value), and (6) the number of iterations or epochs. The second part of the algorithm is presented in Algorithm 2.

The algorithm starts with initiating setting variables including: 1) the number of clusters to project, 2) the number of batch size to retrieve and feed the autoencoder, 3) the shape of the input data, which is the number of input columns entered to the model ($< volatility, return >$)), 4) the output shape (i.e., in out case is 1, an output with one column, which is a floating variable representing the cluster label), 5) the train and test size, and 6) the number of epochs for iterative training . The algorithm then loads previously saved data that were captured through Algorithm 1. The loaded data are then split into two data sets of train (68%) and (33%) for test sets, respectively.

The exact building of the autoencoder starts with specifying the shape of the input data. In our case, the shape of the input data is a vector with two columns $< volatility, return >$. The creation of different layers of the autoencoder starts where the input shape is given to build the *x* part of the autoencoder model (as specified in Fig. 3). The input layers of the autoencoder are then built where the shape of the input is given to the first layer with 100 neurons, and the built first layer and its output is given to the second with 50 nodes or neurons, and the third with 20 neurons or nodes. The activation function for building these layers is "*relu*" which returns a value between (0 to 1). The *h* part of the autoencoder (Fig. 3) is built where the number of cluster labels is specified. The activation function here is "*sigmoid*."

The encoding part of the autoencoder and the encoding layers are then built where a decreasing number of neurons or nodes on each layer indicates filtering important features of data and preserving them for further analysis for the next layer (i.e., feature reduction).

Conversely and in a similar manner, the decoder part of the autoencoder intends to reconstruct the initial input data using the encoded data. The first layer of the decoder takes the output of the "encoder" with 20 neurons. The additional decoder layers are then built symmetrically with respect to the layers incorporated for the encoder part with a similar activation function. Eventually, the *r* part of the autoencoder (see Fig. 3) is built, where a floating

Algorithm 2: Deep learning-based encoder-decoder model for predicting cluster labels.)

Result: A clustering model based on an autoencoder-based deep learning.

1 initialize;
2 Splitting labeled time series data into test and train;
3 Build an encoder-decoder architecture model on training data;
4 Predict the label for the test data using the encoder-decoder model;

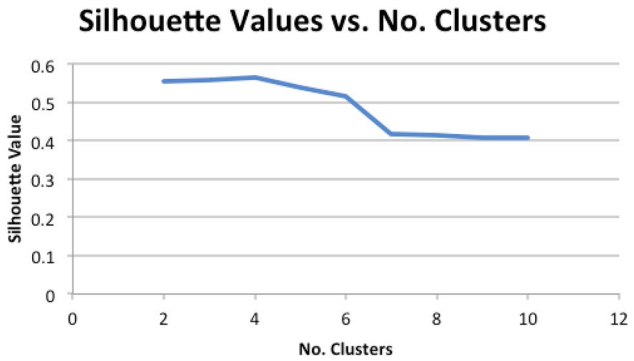

Fig. 4 Optimal number of clusters

variable is estimated to show the cluster label of the input data $< volatility, return >$.

The built autoencoder maps the input to the decoded and reconstructed output and the model itself is built. The model is then compiled using "*adam*" optimizer and mean square error (i.e., MSE) as a metric to assess the precision of the prediction. The built model is then given the training data set with a given number of epochs and bath size and eventually the test data are provided to the model for the purpose of prediction of their cluster labels. In the end, the absolute and the round value of the floating output value is reported as the predicted cluster label. The codex of the algorithm is given in "Appendix B" Listing 2 for interested readers.

7 Case study and evaluation

This section reports the results of a case study performed and thus it evaluates the introduced two-stage synergic methodology to cluster financial time series data.

7.1 Development platform

The authors implemented the algorithms in Python 2.7.13, the anaconda version. The deep learning portion of the algorithms was developed using tensorflow and keras, the open source Python implementations of deep learning and neural networks. The experiments were executed on a MacBook Pro computer with OS X El Capital 10.11.2 operating system with 2.8 GHz Intel Core i7 and 16GB 1600 MHz DDR3.

7.2 Data collection

The authors collected the indexes and ticker symbols for 70 companies listed by S&P 500. The ticker symbols were scarped from the URL of the Wiki page of the S&P 500[1]. The *read_html* Python library was used to automatically scrap and extract the required data from the given Web page. Once the thicker and symbol of the selected companies are identified, a Python script captured the time series data and more specifically the "Adjusted Close" for the selected stock symbol. The adjusted close value data were captured for the period of January 1, 2019 to April 15, 2019 on a daily basis.

7.3 Experimental results

This section reports the results of different analysis performed on the time series data along with the performance obtained using the introduced two-stage clustering methodology.

7.3.1 Detemination of the optimal number of clusters

The determination of the optimal number of clusters is essential in improving the precision and accuracy of the proposed methodology. An optimal clustering groups time series data with respect to an optimization metric and assigns the best label for each time series data that can be used in later stages of the algorithms for training and testing. There are several known methods to determine the optimal number of clusters that best clusters data with respect to the optimization metric. The elbow method, the average Silhouette method, and the gap statistics method are a few techniques to help finding the optimal number of clusters.

The authors used the average silhouette method to decide about the optimal number of clusters. To do so, the conventional KMeans clustering algorithm with a desired number of clusters between 2 and 10 was applied to the feature vector data set. Figure 4 illustrates the obtained Silhouette value for each clustering with different number of clusters. Accordingly, the optimal clustering is achieved when the Silhouette value is maximized.

As Fig. 4 shows the best optimal Silhouette value is produced when the number of clusters is set to 4 (i.e., Silhouette Value = 0.564). Therefore, the authors set the number of clusters to 4 for the remaining part of the case study.

[1] https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

Table 1 The result of KMeans clustering based on four clusters

	Cluster "0"			Cluster "1"			Cluster "2"			Cluster "3"		
	Index	Vol.	Ret.	Index	Vol.	Ret.	Index	Vol.	Ret.	Index	Vol.	Ret.
1	ACN	0.179	0.909	MMM	0.201	0.512	ABBV	0.254	−0.234	AMD	0.71	1.651
2	ADBE	0.223	0.713	ABT	0.207	0.468	ABMD	0.398	−0.416	ALXN	0.298	1.229
3	AES	0.172	0.909	AAP	0.29	0.516	ATVI	0.46	0.154	ALGN	0.427	1.431
4	A	0.2	0.781	AMG	0.296	0.537	ALK	0.259	−0.000	APC	0.696	1.402
5	APD	0.162	0.741	AFL	0.109	0.328	ABC	0.283	0.07	APTV	0.291	1.478
6	AKAM	0.203	0.982	ALB	0.333	0.317	AMGN	0.209	0.04	ANET	0.377	1.634
7	ARE	0.136	0.969	ALLE	0.178	0.582	ANTM	0.338	0.035			
8	AMT	0.123	0.866	AGN	0.31	0.299						
9	AMP	0.256	1.05	ADS	0.317	0.612						
10	AME	0.181	0.888	LNT	0.137	0.503						
11	APH	0.208	0.978	ALL	0.136	0.65						
12	ADI	0.285	1.08	GOOGL	0.223	0.557						
13	ANSS	0.204	1.036	GOOG	0.225	0.573						
14	AON	0.254	0.766	MO	0.278	0.584						
15	AOS	0.201	0.918	AMZN	0.284	0.689						
16	APA	0.336	1.157	AEE	0.14	0.483						
17	AIV	0.127	0.708	AAL	0.379	0.317						
18	AAPL	0.308	0.894	AEP	0.125	0.553						
19	AMAT	0.4	0.997	AXP	0.152	0.571						
20	ADSK	0.287	1.076	AIG	0.276	0.617						
21	ADP	0.169	0.851	AWK	0.123	0.6						
22	AZO	0.202	0.866	ADM	0.178	0.253						
23	AVB	0.102	0.708	ARNC	0.396	0.489						
24	AVY	0.186	0.959	AJG	0.157	0.439						
25	BHGE (BKR)	0.278	0.879	AIZ	0.165	0.258						
26	BLL	0.163	0.963	ATO	0.14	0.457						
27	BAC	0.256	0.733	T	0.182	0.443						
28	BAX	0.152	0.721	BK	0.183	0.417						
29				BBT (PNC)	0.213	0.426						
Mean		0.212	0.896		0.218	0.484		0.314	−0.050		0.466	1.470
STD		0.069	0.127		0.081	0.119		0.089	0.200		0.190	0.131

7.4 Building feature vector: capturing descriptive metadata

The stock market data and their time series can be characterized through two concepts: (1) volatility, and (2) return. In addition to some other relevant concepts, volatility and return can be utilized to capture and summarize the trend and certain behavior of time series. This section describes how these two characteristics are calculated and used in the clustering of time series data.

(a) Annualized Stock's Volatility[2]. To calculate annualized stock's volatility, the standard deviation of the price should be multiplied by the square root of 252 assuming that there are 252 trading days in a given year.

(b) Annualized Stock's Return. The annualized stock's return is computable in a similar fashion. However, instead of standard deviation, the mean value of the prices should be multiplied by the square root of 252.

7.5 The creation of time series clusters using KMeans

Once the annualized stock's volatility and return values are computed for each stock data, the values will be given to a conventional KMeans clustering algorithm with a desired number of clusters identified ealier (i.e., 4). The KMeans algorithm will group the stock's data with respect to volatility and return using the "Euclidean" distance measure.

[2] https://www.fool.com/knowledge-center/how-to-calculate-annualized-volatility.aspx

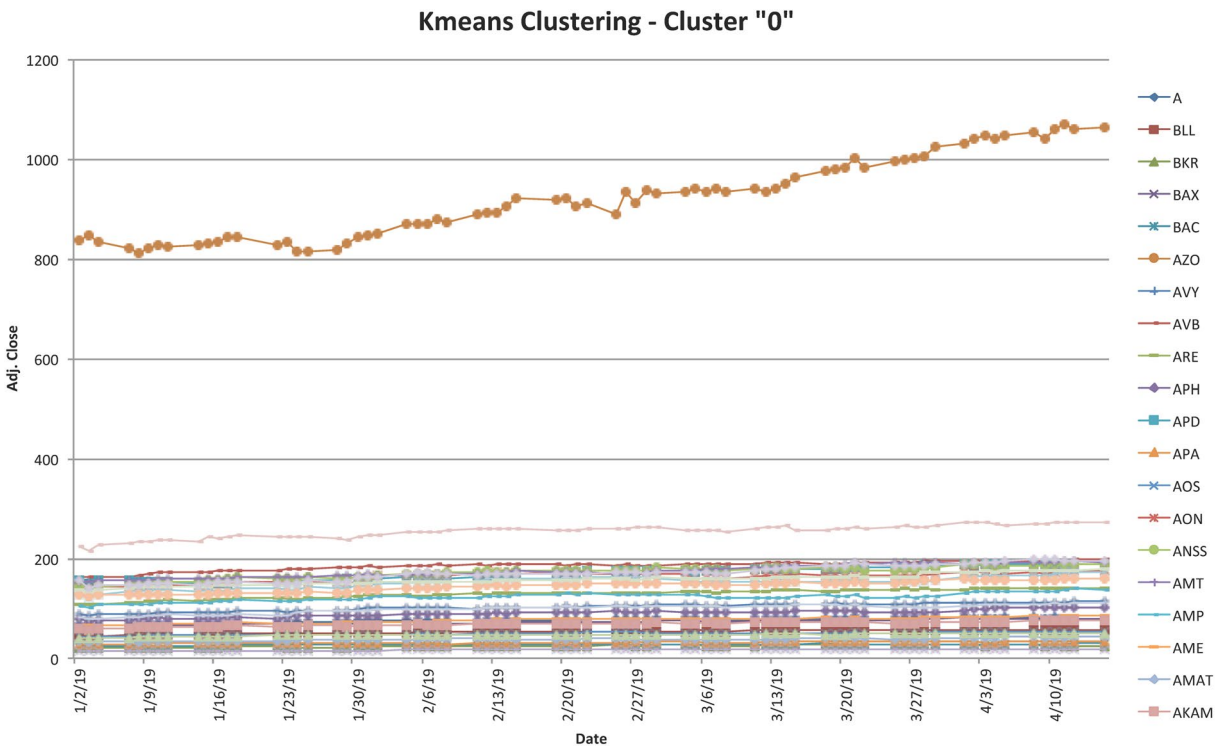

Fig. 5 KMeans clustering: Cluster "0"

The label of clusters formed by the KMeans algorithm will be used as the label for each time series data resulting in the clustering problem of unlabeled data (i.e., unsupervised learning problem) to be transformed into a clustering problem with labels and thus an instance of a supervised learning problem will be created. Table 1 lists the exact values for volatility and return for each member along with the mean and standard deviation values of these features for each cluster.

As Table 1 reports clusters 0, 1, 2, and 3 have 28, 29, 7, and 6 members, respectively. The mean values for the pair of $< volatility, return >$ for each cluster 0, 1, 2, and 3 are $< 0.212, 0.896 >$, $< 0.218, 0.484 >$, $< 0.314, -0.050 >$, and $< 0.466, 1.470 >$, respectively.

To help understanding the results of the KMeans clustering, we visualize the time series data for each member along with the range of volatility and return for each cluster. Figures 5, 6, 7 and 8 illustrate the time series data clustered together. As the Silhouette analysis indicated the optimum number of clusters is four, the figures show the exact time series of members of each cluster for the period of January 1, 2019 and April 15, 2019.

Let us take a look at the clusters and the stock indices grouped together. Figures 9, 10, 11 and 12 illustrate the range of volatility and return values of the stock indices that are clustered together. The stock indices are clustered with respect to two descriptive variables $< volatility, return >$. As a result, the trend of these two variables is similar to the other stock indices clustered in the same group. The figures visualize the range of volatility and returns computed for each member of clusters produces by KMeans clustering for the period of January 1, 2019–April 15, 2019.

7.6 Tuning parameters of the encoder–decoder

Table 2 lists the number of layers, the output shape (i.e., the number of nodes or neurons) of each layer along with the number of parameters estimated at each layer. This output is produced by the the neural deep learning module within the encoder–decoder module.

As highlighted earlier, the authoencoder takes as input a feature vector of size 2 on row 1 of the table (i.e., its shape which is of the form <Volatility, Returns>). It then propagates the input to the internal layers devised for the encoder (i.e., row numbers of 2–5 in Table 2) and decoder (i.e., row numbers 6–9 of Table 2) parts. At the level of *dense*_4 (i.e., row number 5 of Table 2) the shape is in the form of 4, the number of desired clusters. The exact number of layers and nodes are built in a reverse order by the decoder and eventually an output shape with one column (i.e., the predicted label for each time series) is produced.

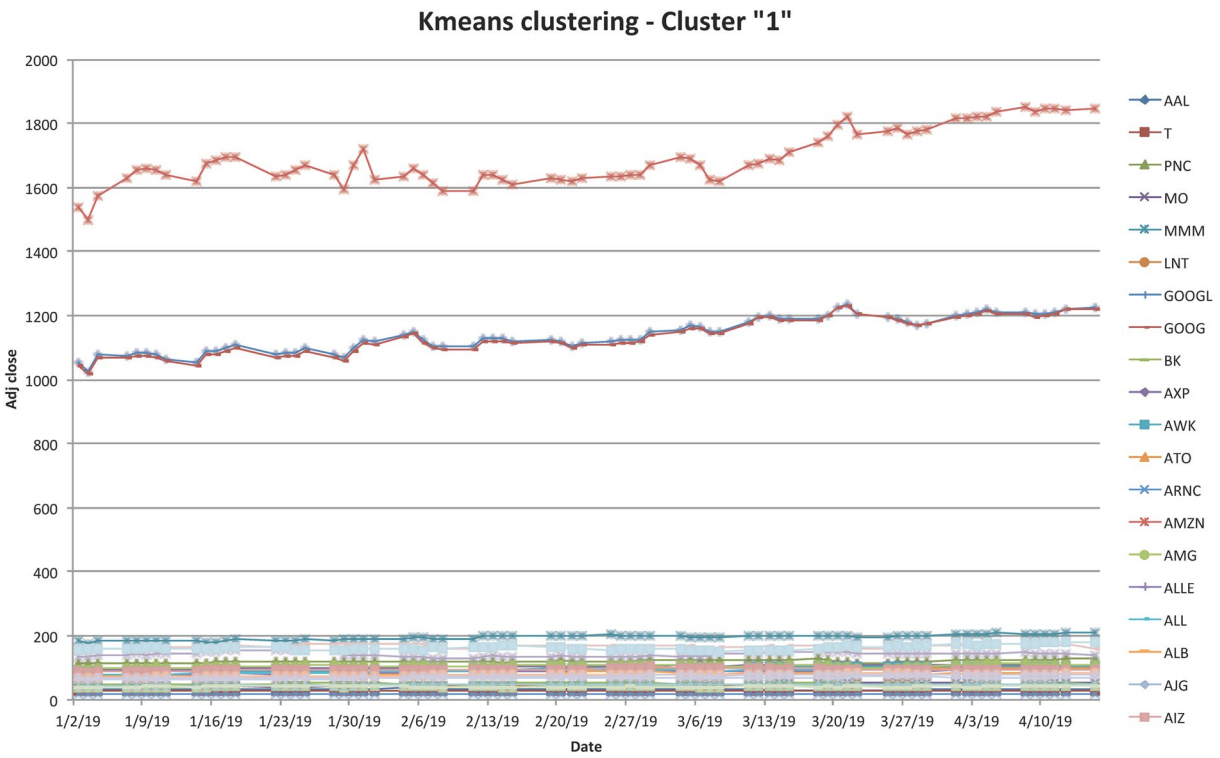

Fig. 6 KMeans clustering: Cluster "1"

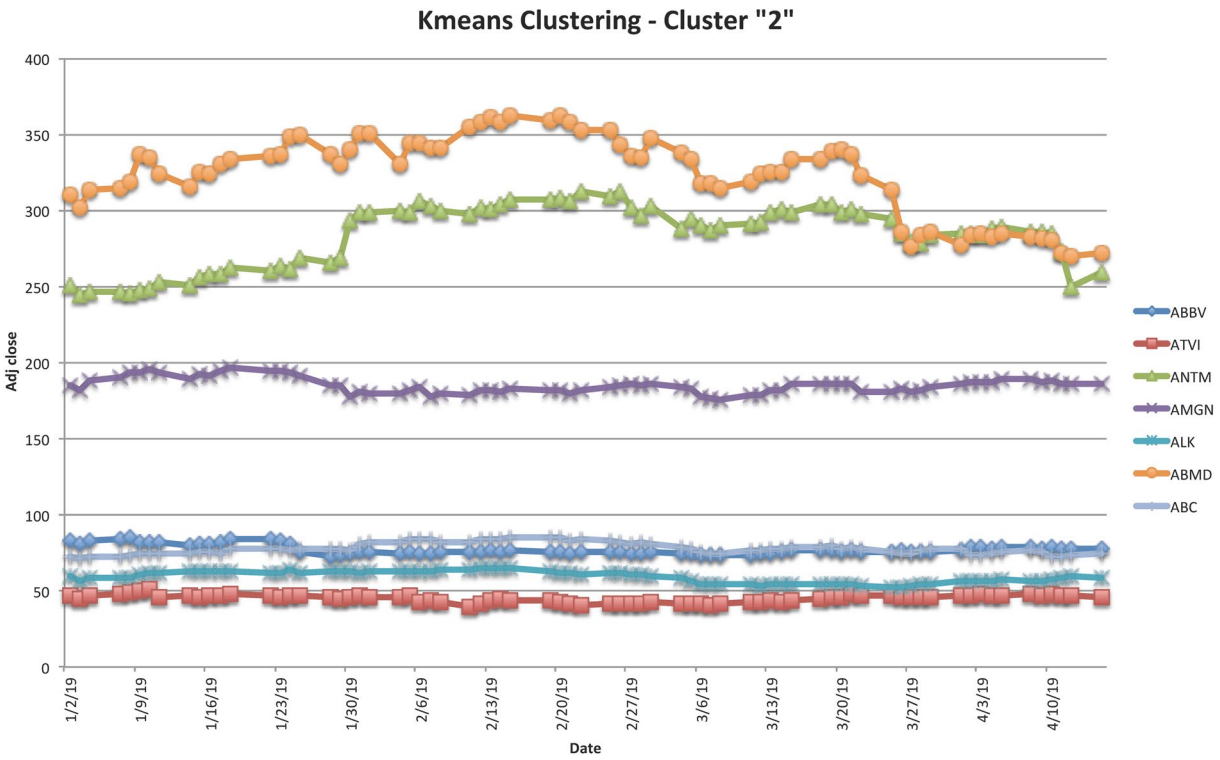

Fig. 7 KMeans clustering: Cluster "2"

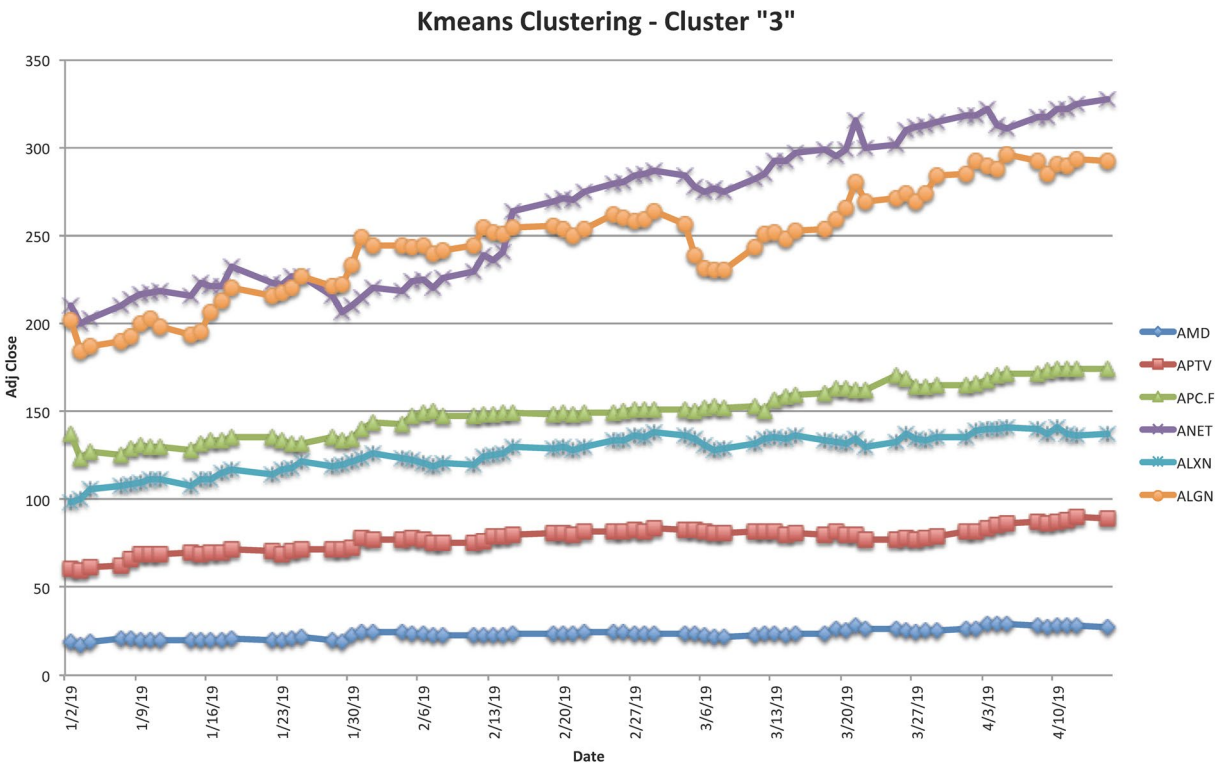

Fig. 8 KMeans clustering: Cluster "3"

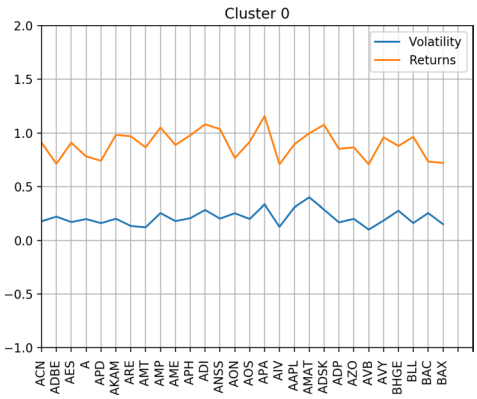

Fig. 9 The range of volatility and returns for Cluster "0"

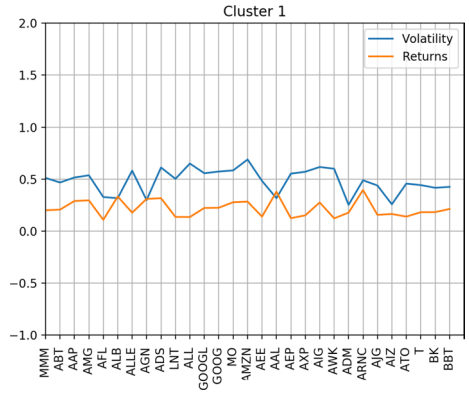

Fig. 10 The range of volatility and returns for Cluster "1"

The total number of trained parameters is 12, 805, which implies creating a fully connected network.

We trained the model with different number of repetitions (i.e., epochs) in order to investigate the performance and influence of estimating the parameter values in further details. We trained the model for 1000 epochs. Figure 13 illustrates the relationship between the number of epochs and the error rate (i.e., loss). As the figure indicates, the loss value is approximately zero when the number of epochs is greater than 316.

We kept the number of epochs as 1000, even though 316 epochs were sufficient. Once the neural network model is trained using the training dataset, it is given the test dataset to predict the time series labels. The prediction is in the form of a numerical floating value that needs to be rounded to its closest integer value. The closest integer value for each time series in fact represents the cluster

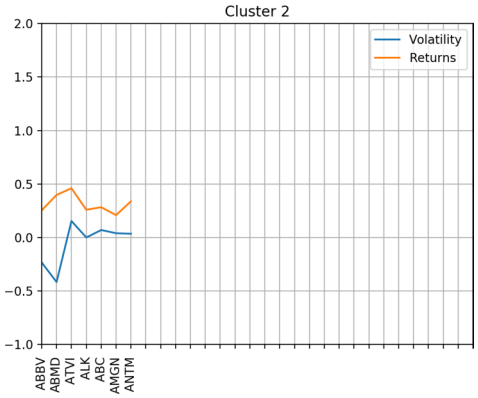

Fig. 11 The range of volatility and returns for Cluster "2"

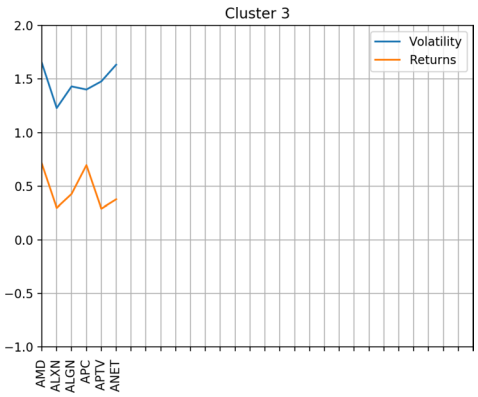

Fig. 12 The range of volatility and returns for Cluster "3"

label for the given time series. The computed numerical output of the program and the rounded with absolute *d* values along with the actual label for the test data set are reported in Table 3.

7.7 Predicting clustering labels through the encoder–decoder

The total number of data set was 70 (i.e., the time series data for 70 stock indices were captured), of which 46 time series data were considered for training the network, and the remaining data set (i.e., 24) was used for testing the model. As Table 3 lists, the neural network was able to predict the cluster label of 21 out of 24 test data correctly achieving an accuracy of 87.5 in prediction (i.e., the cases with dark color). Note that the accuracy in here means matching between the results obtained by the conventional KMeasn and the results achieved by the proposed two-stage deep learning methodology. The differently classified stock indices by the deep learning-based methodology are ALXN, APA, and AMZN, which are emphasized in bold in the table. The results also may indicate that the deep learning-based approach may take into account additional *latent* features while clustering the given data sets.

To help realize the results of the autoencoder-based deep learning time series classification model, we visualize the results. Figures 14, 15, 16 and 17 show the results of the prediction of cluster's label for the test set in which a time series with dark thicker color (i.e., except Cluster "2") show the miss-classifications performed by the prediction performed by the encoder–decoder module. The prediction results in three instances of mislabeling colored in black in the figures.

7.8 KMenas versus deep learning-based clustering

The investigation of why ALXN, APA, and AMLN are classified differently by the encoder–decoder module reveals interesting findings. The clustering performed by conventional KMeans clustering and the encoder–decoder module are illustrated in Figs. 18 and 19. By referring to Table 1,

Table 2 The input and output shapes along with the number of parameters trained

	Type	Layer (type)	Output shape	Parameter#
1	–	Input 1 (Input Layer)	(None, 2)	0
2	Encoder(Layer 1)	dense_1 (Dense)	(None, 100)	300
3	Encoder(Layer 2)	dense_2 (Dense)	(None, 50)	5050
4	Encoder(Layer 3)	dense_3 (Dense)	(None, 20)	1020
5	Encoder(Layer 4)	dense_4 (Dense)	(None, 4)	84
6	Decoder(Layer 1)	dense_5 (Dense)	(None, 20)	100
7	Decoder(Layer 2)	dense_6 (Dense)	(None, 50)	1050
8	Decoder(Layer 3)	dense_7 (Dense)	(None, 100)	5100
9	Decoder(Layer 4)	dense_8 (Dense)	(None, 1)	101
Total trainable parameters				12,805

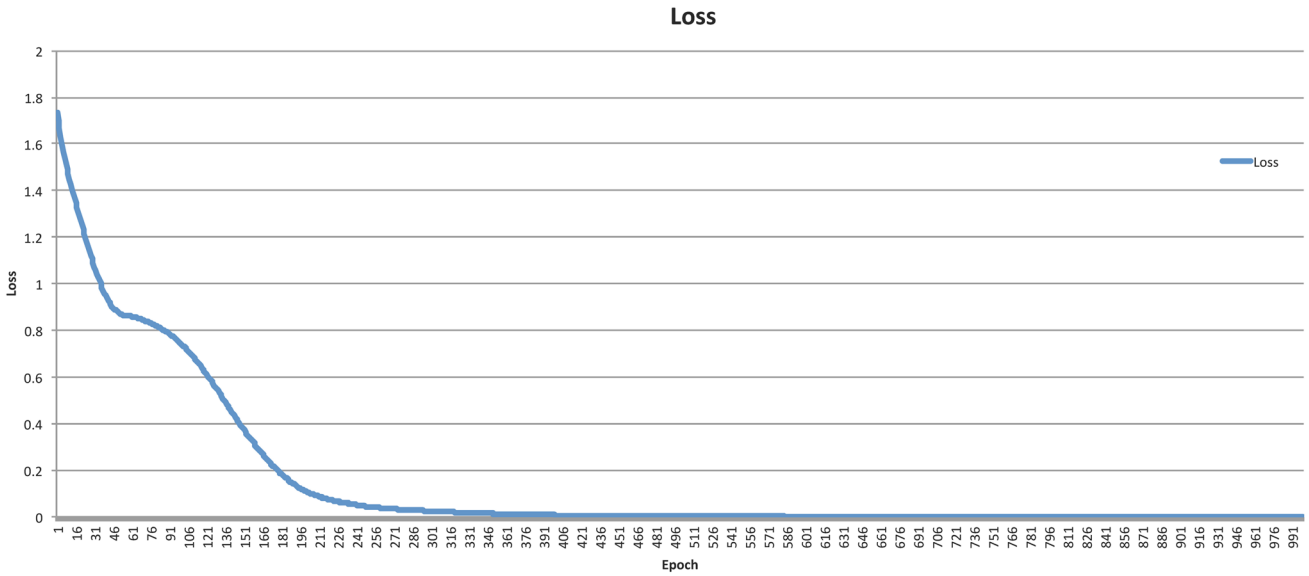

Fig. 13 Loss versus epochs

Table 3 Numerical prediction of time series' cluster labels

	Index	Volatility	Returns	Exact output Label prediction (*r*)	Absolute rounded Prediction *d* (Predicted label)	Actual KMeans Cluster Label	Missed Labeled
1	ADS	0.317	0.612	7.2130698e−01	1.	1	
2	MMM	0.201	0.512	9.9868220e−01	1.	1	
3	AAPL	0.308	0.894	−1.8404983e−04	0.	0	
4	ACN	0.179	0.909	−1.4865603e−03	0.	0	
5	ANET	0.377	1.634	3.0150795e+00	3.	3	
6	**ALXN**	**0.298**	**1.229**	**2.4604988e+00**	**2.**	**3**	**X**
7	AMG	0.296	0.537	9.9893832e−01	1.	1	
8	AIG	0.276	0.617	8.0927080e−01	1.	1	
9	AON	0.254	0.766	4.7755931e−03	0.	0	
10	A	0.200	0.781	2.7296934e−03	0.	0	
11	AEP	0.125	0.553	9.9907714e−01	1.	1	
12	AES	0.172	0.909	−1.5225317e−03	0.	0	
13	BK	0.183	0.417	9.9732614e−01	1.	1	
14	ATVI	0.460	0.154	1.9892873e+00	2.	2	
15	AVB	0.102	0.708	7.7624805e−02	0.	0	
16	AAL	0.379	0.317	1.0931786e+00	1.	1	
17	T	0.182	0.443	9.9788338e−01	1.	1	
18	AWK	0.123	0.600	9.9946052e−01	1.	1	
19	ATO	0.140	0.457	9.9807245e−01	1.	1	
20	**APA**	**0.336**	**1.157**	**2.0670881e+00**	**2.**	**0**	**X**
21	ALB	0.333	0.317	9.9551427e−01	1.	1	
22	AMT	0.123	0.866	−1.1737701e−03	0.	0	
23	ADI	0.285	1.085	1.2981926e−01	0.	0	
24	**AMZN**	**0.284**	**0.689**	**3.1280313e-03**	**0.**	**1**	**X**

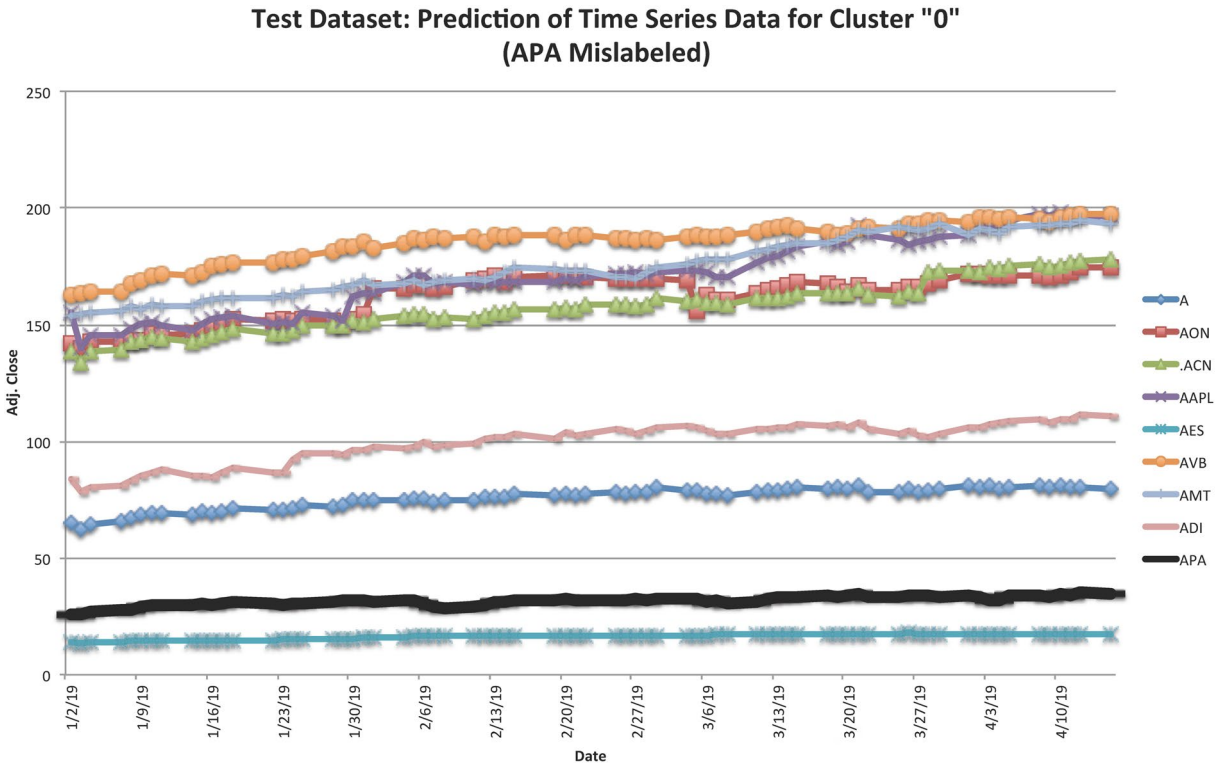

Fig. 14 Prediction of time series data for Cluster "0"

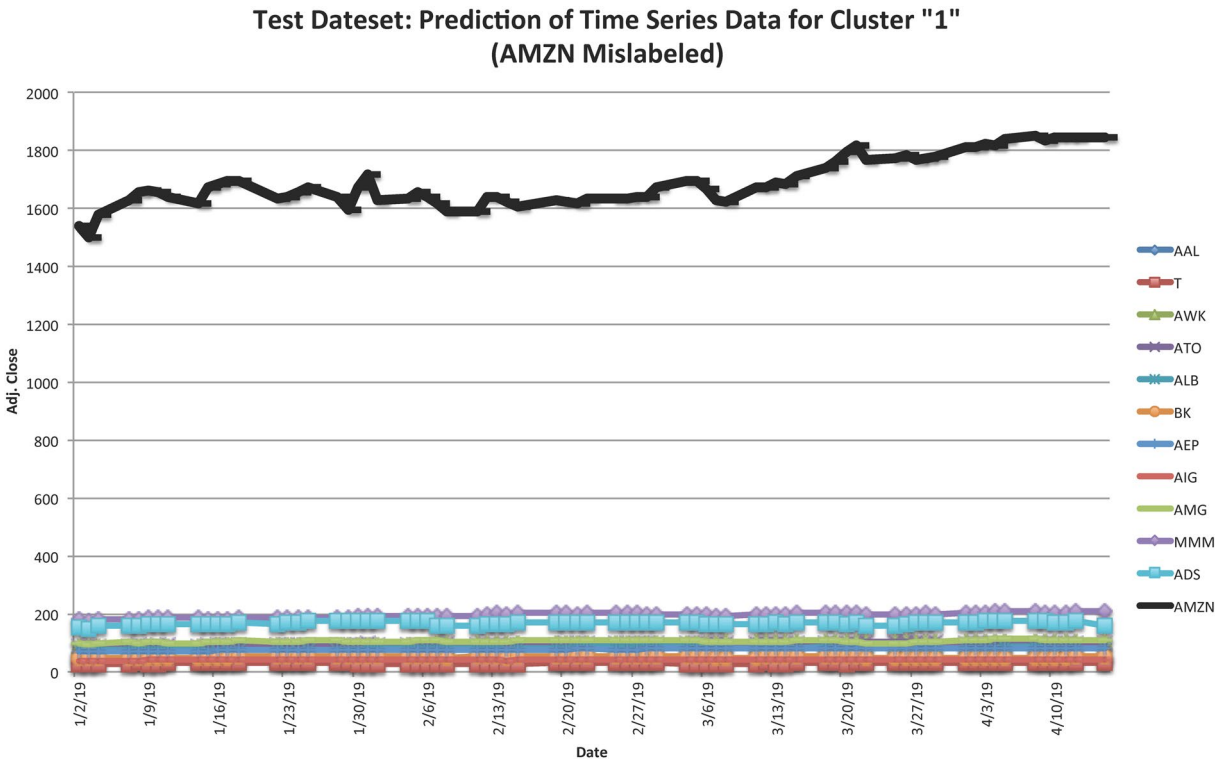

Fig. 15 Prediction of time series data for Cluster "1"

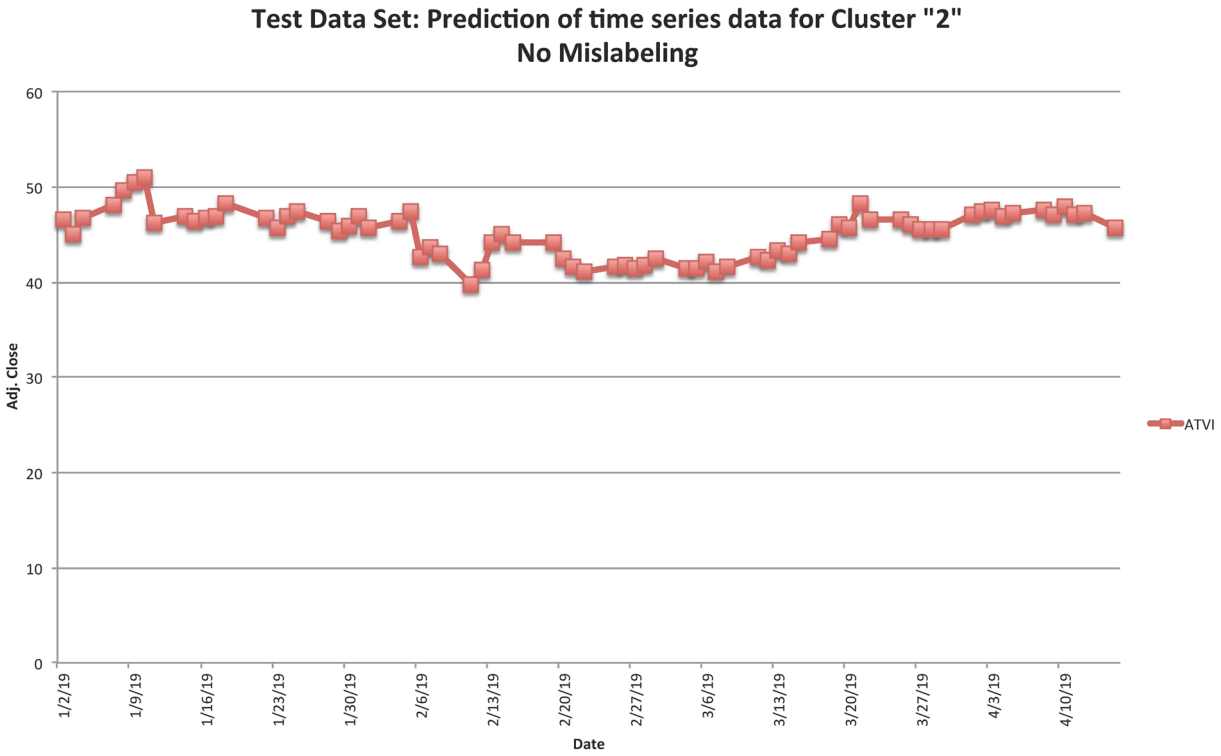

Fig. 16 Prediction of time series data for Cluster "2"

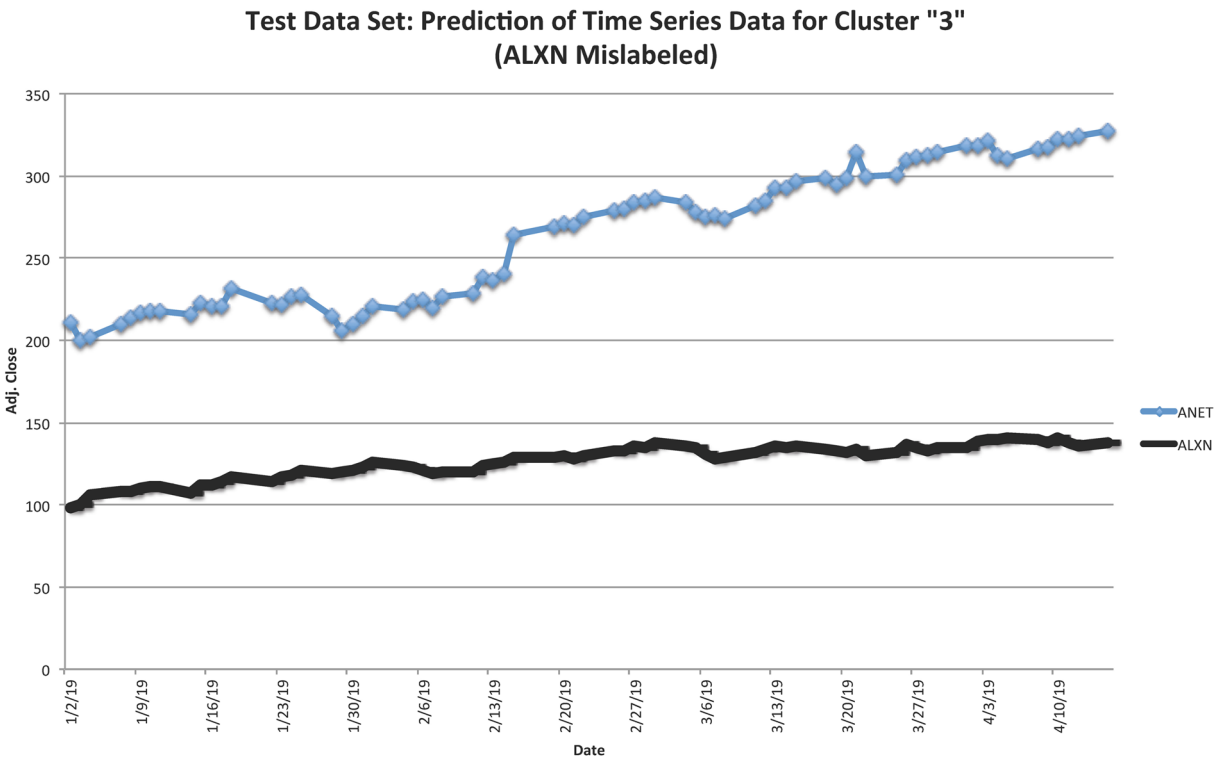

Fig. 17 Prediction of time series data for Cluster "3"

in which the results of KMeans clustering are reported, we observe that:

- The feature vector of AMZN is $< 0.284, 0.689 >$. A comparison of the feature vector for AMZN with those clustered together in Cluster "1" by conventional KMeans shows that the return value for AMZN is on the upper bound of the return values clustered in Cluster "1" (i.e., it is the max value for the return).
- A similar finding is observable for APA which is clustered by conventional KMeans in Cluster "0" with $< 0.336, 1.157 >$ (Table 1). Similarly, both volatility and return values calculated for APA are on the upper bounds of the volatility and return values calculated for stock indices cluster together in Cluster "0".
- Similarly, the feature vector calculated for ALXN is $< 0.298, 1.229 >$ which both are on the lower bounds of the feature vectors for volatility and returns clustered together in Cluster "2".

The findings indicate that, these three stock indices (i.e., ALXN, APA, and AMZN) are on the border line of clusters (aee Figs. 18 and 19). Even though the figures may imply that the clustering produced by KMeans has been performed reasonably well, it may also indicate that clustering performed by the autoencoder might have taken into account some other hidden factors. Hence, since deep learning-based approach is discovering and taking into account more hidden features among these two values (i.e., volatility and return), the clustering performed by the autoencoder is actually providing more insights about these stock indices and their relationships. More precisely, it might indicate that there might be some other hidden features discovered by the autoencoder that are missed and not formulated by the conventional KMeans clustering algorithms.

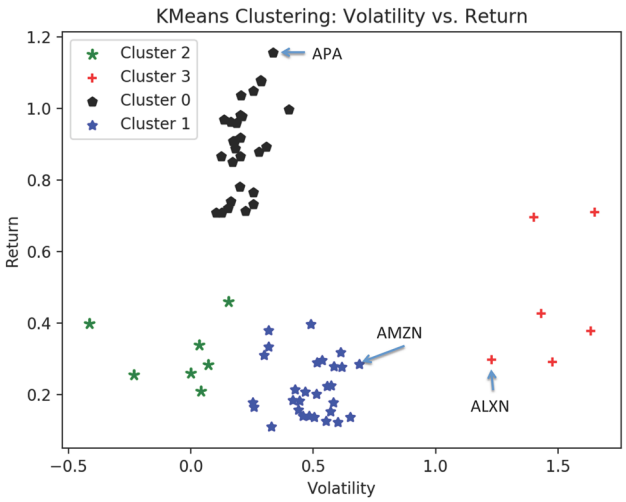

Fig. 18 KMeans Clustering

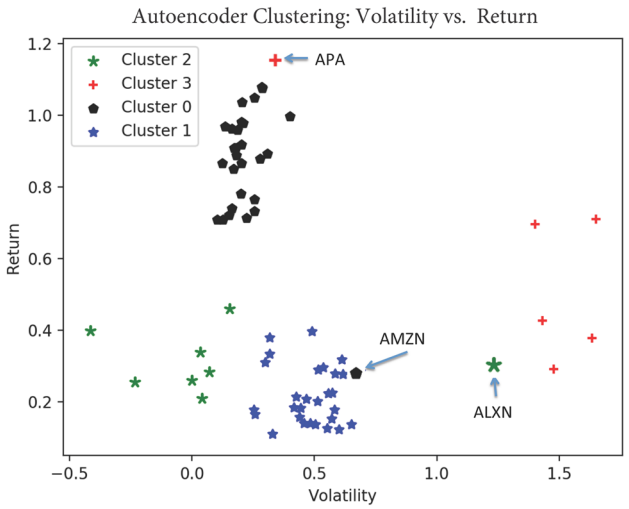

Fig. 19 Autoencoder Clustering

8 Conclusions and future work

This paper introduced a two-stage deep learning-based approach to address the time series clustering problem. The time series clustering methodology presented in this paper first generates labels for time series data using KMeans clustering. The clustering performed through the vanilla form of a KMeans algorithm is unsupervised, in which the labels of the data are unknown. Using the results produced by the KMeans algorithm, it is possible to label each cluster and thus enable treating the problem as a supervised learning problem. Once the cluster labels are produced, then they are given to an encoder-decoder-based deep learning neural network in order to build a classifier and thus a clustering model. The most important advantage of building such a neural network is that it models hidden features and takes such features into account when building the prediction model.

It is important to note that it is possible to apply cluster analysis directly to the raw time series data and produce some clusters for the time series data. However, there are several problems with this approach: (1) it is computationally more expensive to cluster a set of time series directly, depending on the length of the time series, mainly due to the curse of dimensionality and cost of computation required for clustering, (2) Since the entire time series data are taken into account for clustering, it is possible to have some noises in the data that may lead us to an

improper clustering, (3) since the raw time series data are not labeled, this approach is basically a clustering without knowing the exact number of optimal clusters (unsupervised clustering) and thus the clustering may not be accurate.

In our work, we have introduced an approach to create some labels for the clusters in order to address the aforementioned problems listed above. The introduced approach (1) captures a summary of the underlying time series that explains the variability, volatility, and the trend of the time series thus ignoring possible noises in the computation. The summary is called feature vector; (2) then the feature vector is given to an encoder–decoder learning module to identify latent and also most important features from the feature vector, and then further eliminate the features in the feature vector whose contributions to the computation is less significant; and (3) use the feature vector as a label for the time series and thus transform the problem into supervised clustering where the optimal number of clusters is determined.

The case study conducted in the context of the financial time series data shows an accuracy of 87.5% in clustering such data. More importantly, we observed that the deep learning-based model performs comparatively similar to the conventional KMeans clustering. However, we obtained an interesting result. It was observed that some of the data points were classified differently by these two approaches (i.e., conventional KMeans and the proposed encoder–decoder deep learning-based methodology). This indicates that there might be some hidden features that the conventional algorithms such as KMeans cannot capture when clustering data; whereas, the deep learning-based approaches are able to capture these hidden features and thus perform clustering on an augmented set of features. This observation poses an interesting research question where the performance of these two approaches need to be studied.

The application of deep learning approaches to time series analysis and in particular financial time series data is in its early stages. Several other classical problems in time series analysis can be formulated using deep learning techniques such as shock and anomaly detection, seasonal effects as well as clustering and prediction at different levels of abstractions. Neural network-based techniques such as Long Short Term Memory (LSTM) [20–22] and their autoencoder-based variations, Generative Adversarial Networks (GANS), attention networks need to be further explored for formulating classical problems in time series clustering and data analysis. There are also some other machine learning and deep learning-based techniques that can be applied and their performance can be investigated such as Random Forest [24]. It is of utmost importance to conduct several experimental studies with the objective of comparing the performance of clustering produced by conventional algorithms and deep learning-based clustering models. Our results show that conventional clustering techniques take into account explicit features; whereas, the advanced deep learning-based models explore not only explicit but also implicit features when clustering data points. As a result, it is necessary to investigate which clustering makes more sense with respect to the underlying application domains.

Funding This work is partially funded by the support from National Science Foundation under the Grant Numbers 1564293, 1723765, and 1821560.

Data Availability Statement The data produced by this research are publicly available.

Compliance with ethical standards

Conflict of Interest The authors do not have any conflict of interests to disclose.

Appendix: Codex listing

A: Transformation of unsupervised learning to supervised learning through using metadata as labels

Listing 1. Enabling supervised learning through utilization of metadata of time series as labels.

```
Algorithm 1 (Transforming Unsupervised Learning to Supervised Learning):
Description: Transforming Unsupervised Data to Labeled Data and Clustering
Stock Labeled Data Using Optimal KMean Algorithm.
Inputs: 1) URL to scrap, 2) Number of Clusters,
    3) Number of Stock Tickers, 4) The Start Date of Collecting Stock Prices
Outputs: The Cluster Label of the Scrapped Stock Tickers

# Setting
1.  numpy.random.seed(7)           # For reproducibility purpose
2.  sp500_URL = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'   # Web page to scrape
3.  no_clusters = 4                # Number of desired clusters obtained through experiments
4.  no_tickers  = 70               # Number of tickers to scrap and analyze
5.  start_date  = "01/01/2019"     # The start date to scrap tickers data

# Declaring Some Data Frames to Hold Scrapped Data
6.  tickers  = []  # A data frame to hold the tickers: <ticker>
7.  TP_DF    = []  # A data frame to hold the scrapped data: <ticker, prices[]>
8.  TVR_DF   = []  # A data frame to hold: <ticker, volatility, returns>
9.  VR_DF    = []  # A data frame to hold: <volatility, returns> (used by clustering)

# Scrapping the Web page and tickers
10.  sp500_scrapped = read_html(sp500_URL)  # using the Panda read_html function
11. tickers = read(sp500_scrapped)

# Retrieve "Adj_Close" from yahoo regarding each ticker
12. for each ticker in tickers[<= no_tickers] do
13.    prices = read(ticker, "yahoo", start_date)["Adj_Close"]
14.    TP_DF.append(<ticker, prices[]>);
15. end for
# Sort the records to re-construct based on "ticker" or index
16. TP_DF.sort

# Compute volatility and returns regarding each ticker
17. for each ticker in TP_DF do
18.    index      = ticker
19.    returns    = mean(prices) * 252
20.    volatility = std(prices) * sqrt(252)
21.    TVR_DF.append(<index, volatility, returns>)
22. end for

# Build VR_DF[] using TVR_DF[]
23. VR_DF = <TVR_DF.volatility, TVR_DF.returns>

# Cluster <returns, volatility> data without ticker using KMean clustering
# Build the clustering model using KMean
24. clusters = KMean(n_clusters = no_clusters)
# Fit the model/predict the cluster labels regarding each data item <ret, vol>
25. predicts = clusters.fit_predict(VR_DF)
# Report the silhouette value using Euclidean distance and identify centroids
26. centers = clusters.cluster_center_
27. score = silhouette_score(VR_DF, predicts, metric = "euclidean")

# Assign the cluster tag regarding each ticker (<index, voly, ret, cluster>)
28. for each ticker in TP_DF do
29.    TVR_DF.cluster[index == ticker] = pd.DataFrame(predicts[index == ticker])
30. end for

# Save the data to a file to be used by Algorithm 2: (<ticker, vol, ret, cluster>)
31. TVR_DF.to_csv("/.../k-means-StockData.csv")
```

B: Deep leaning-based autoencoder supervised learning for clustering

Listing 2. Deep learning-based (Encoder-Decoder) supervised learning for predicting cluster labels of stock indices.

```
Algorithm 2 (Supervised Learning through Autoencoder-based Deep Learning):
Description: Building An Autoencoder to Predict Cluster Label of Stock Indices
Inputs: 1) Training labeled set, 2) Testing unlabeled set.
Outputs: An Autoencoder-based Deep Learning Model to Predict Cluster Labels

# Setting
1.  seed = 7
2.  numpy.random.seed(seed)    # For reproducibility purpose
3.  no_clusters = 4    # Number of desired clusters (i.e., # of Neurons or Nodes)
4.  BatchSize   = 1024 # data batch size retrieved by the learner in each iteration
5.  InCol = 2  # The shape of the input data used regarding training <vol, ret>
6.  OuCol = 1  # The shape of the output data, A floating value
7.  TestSize = 0.33  # the percentage of the "test" data of the splitting data
8.  noEpochs = 1000 # Number of epochs (learning round) regarding training the model

# Loading labeled data (train and test): (<ticker, volatility, returns, cluster>)
9.  TVR_DF = pd.read_csv("/.../k-means-StockData.csv")  # Created by Algorithm 1

# Splitting the data set into training and test set
10.  x = TVR_DF[<volatility, returns>]
11.  y = TVR_DF[<cluster>]
12.  X_train, X_test, y_train, y_test =
                train_test_split(x, y, test_size=TestSize, random_state = seed)

# Alternatively the InCol = TVR_DF.shape[1] command can be used to capture the
# input shape, instead of using the hard coding style regarding InCol.
# InCol = TVR_DF.shape[1]

# Build a tensor shape
13. input_dim = Input(shape = (InCol, ))

# Build the autoencoder
# Build the encoder part that represents the input
14. encoded = Dense(100, activation = "relu")(input_dim)
15. encoded = Dense(50, activation = "relu")(encoded)
16. encoded = Dense(20, activation = "relu")(encoded)
17. encoded = Dense(no_cluster, activation = "sigmoid")(encoded)

# Build the decode part that losey reconstruct the input
18. decoded = Dense(20, activation = "relu")(encoded)
19. decoded = Dense(50, activation = "relu")(decoded)
20. decoded = Dense(100, activation = "relu")(decoded)
21. decoded = Dense(OuCol)(decoded)

# Map input to its reconstruction
22. autoencoder = Model(input_dim, decoded)

# Compile the autoencoer with proper optimizer and loss function
23. autoencoder.compile(optimizer="adam", loss="mse")

# Train the autoencoder model using training data set
24. train_history = autoencoder.fit(X_train, y_train,
                epochs = noEpochs, batch_size = BatchSize)

# Predict the cluster tag of the test data set using the autonecoder model
25. predicts = autoencoder.predict(X_test)

# Report the labels of each stock index in the test data
26. return np.absolute(np.rint(predicts))
```

References

1. Liao TW (2005) Clustering of time series data—A survey. Pattern Recognit 38(11):1857–1874
2. Chen Y, Nascimento MA, Ooi BC, Tung AK (2007) Spade: on shape-based pattern detection in streaming time series, In: IEEE 23rd international conference on data engineering, pp 786–795
3. Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. Proc VLDB Endow 1(2):1542–1552
4. Stefan A, Athitsos V, Das G (2013) The move-split-merge metric for time series. IEEE Trans Knowl Data Eng 25(6):1425–1438
5. Wang X, Mueen A, Ding H, Trajcevski G, Scheuermann P, Keogh E (2013) Experimental comparison of representation methods and distance measures for time series data. Data Min Knowl Discov 26(2):275–309
6. MacQueen J (1967) Some methods for classification and analysis of multivariate observations, In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1, no. 14, Oakland, CA, USA, pp 281–297
7. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: proceedings of the second international conference on knowledge discovery and data mining (KDD'96). pp 226–231
8. Wang W, Yang J, Muntz RR (1997) Sting: a statistical information grid approach to spatial data mining. VLDB 97:186–195
9. Sheikholeslami G, Chatterjee S, Zhang A (1998) Wavecluster: a multi-resolution clustering approach for very large spatial databases. VLDB 98:428–439
10. Fu T, Chung F, Ng V, Luk R (2001) Pattern discovery from stock time series using self-organizing maps. In: Workshop notes of KDD2001 workshop on temporal data mining, pp 26–29
11. Aghabozorgi S, Ying Wah T, Herawan T, Jalab HA, Shaygan MA, Jalali A (2014) A hybrid algorithm for clustering of time series data based on affinity search technique. Hindawi's Sci World J
12. Lai C-P, Chung P-C, Tseng VS (2010) A novel two-level clustering method for time series data analysis. Expert Syst Appl 37(9):6319–6326
13. Sewell M (2011) Characterization of financial time series. UCL Department of Computer Science, Research Note: RN/11/01. Tech, Rep
14. Mamtha D, Srinivasan KS (2016) Stock market volatility: conceptual perspective through literature survey. Mediterr J Soc Sci 7(1):208–212
15. Bakaert G, Wu G (2000) Asymmetric volatility and risk in equity markets. Rev Financ Stud 13(1):1–42
16. Whitelaw R (2000) Stock market risk and return: an empirical equilibrium approach. Rev Financ Stud 13(3):521–547
17. Shawky HA, Marathe A (1995) Expected stock returns and volatility in a two regime market. J Econ Bus 47(5):409–422
18. Chung K, Chuwongananat C (2018) Market volatility and stock returns: the role of liquidity providers. J Financ Mark 37:17–34
19. Bandhopadhyay S (2012) On the return and volatility spillover between us and Indian stock market. Int J Financ Manag 2(3)
20. Siami-Namini S, Tavakoli N, Namin AS (2018) A comparison of ARIMA and LSTM in forecasting time series, In: 17th IEEE international conference on machine learning and applications, ICMLA 2018, Orlando, FL, USA, December 17–20, pp 1394–1401
21. Tavakoli N (2019) Modeling genome data using bidirectional LSTM. In: The 1st IEEE international workshop on deep analysis of data-driven applications (DADA) in conjunction with COMPSAC
22. Siami-Namini S, Tavakoli N, Namin AS (2019) The performance of LSTM and BiLSTM in forecasting time series. In: IEEE Big Data, Los Angeles, California, USA
23. Hubens N (2019) Deep inside: Autoencoders. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f, Tech. Rep
24. Abri F, Siami-Namini S, Khanghah MA, Soltani FM, Namin AS (2019) Can machine/deep learning classifiers detect zero-day malware with high accuracy? In: IEEE Big Data, Los Angeles, California, USA