# UNSUPERVISED LEARNING I: OVERVIEW OF TECHNIQUES

Joseph Simonian, PhD
*Senior Affiliate Researcher*
*CFA Institute*

Unsupervised learning is a branch of machine learning that encompasses algorithms used to discover hidden patterns and structures in data without labeled examples from which to learn. Unlike supervised learning, there is no "ground truth" to guide the learning process, which means that the algorithm must discover hidden patterns and relationships in data without any explicit guidance from real-word observations regarding what constitutes the correct answer. Without ground truths, unsupervised learning algorithms must rely on mathematical principles, such as maximizing likelihood or minimizing error, to capture the essence of the data. This makes unsupervised learning both an art and a science, requiring careful consideration of what constitute meaningful patterns versus mere noise.

In financial contexts, unsupervised learning can be particularly useful because financial markets are often opaque, labeled data are often scarce or expensive to obtain, or such data quickly become obsolete. In other words, the "correct" answer is often elusive to varying degrees. Financial markets are also dynamic, and as market regimes change, new patterns emerge and traditional relationships often break down. In such cases, unsupervised learning methods can be invaluable in helping practitioners discover structures in financial data that may prove valuable in their portfolio and risk management efforts.

## Clustering

Perhaps the most well-known framework for unsupervised learning is clustering. Simpler clustering algorithms, such as *k-means clustering* (Lloyd 1982), operate according to a criterion of compactness, with observations grouped into different clusters based on their distance from designated *centroids*. These centroids are the average (mean) positions of all the data points that belong to a particular cluster. The algorithm for *k*-means clustering is shown in **Figure 1**.

A *k*-means clustering approach makes a good choice when data are numeric, clusters are roughly spherical and similar in size, and a fast, scalable clustering for large datasets is needed. It is mathematically simple, efficient, and easy to interpret. However, *k*-means also assumes that clusters are spherical and equal sized, which is not always the case. Further, it is sensitive to initialization and outliers and requires a specification of the number of clusters, *k*. Finally, *k*-means clustering can detect only clusters that are linearly separable, limiting its usefulness in applications in which nonlinear or otherwise nuanced relationships are present.

With the foregoing in mind, however, note that *k*-means has nevertheless been applied to portfolio construction. For example, Wu, Wang, and Wu (2022) used *k*-means to cluster stocks according to their continuous trend characteristics and then used inverse volatility weighting, risk parity, and mean–variance-type considerations to arrive at final portfolio weights.

## Figure 1. Algorithm: *k*-Means Clustering

Input: Dataset $X$ with $n$ points, number of clusters $k$, maximum iterations

Output: Cluster assignments and centroids

Begin:

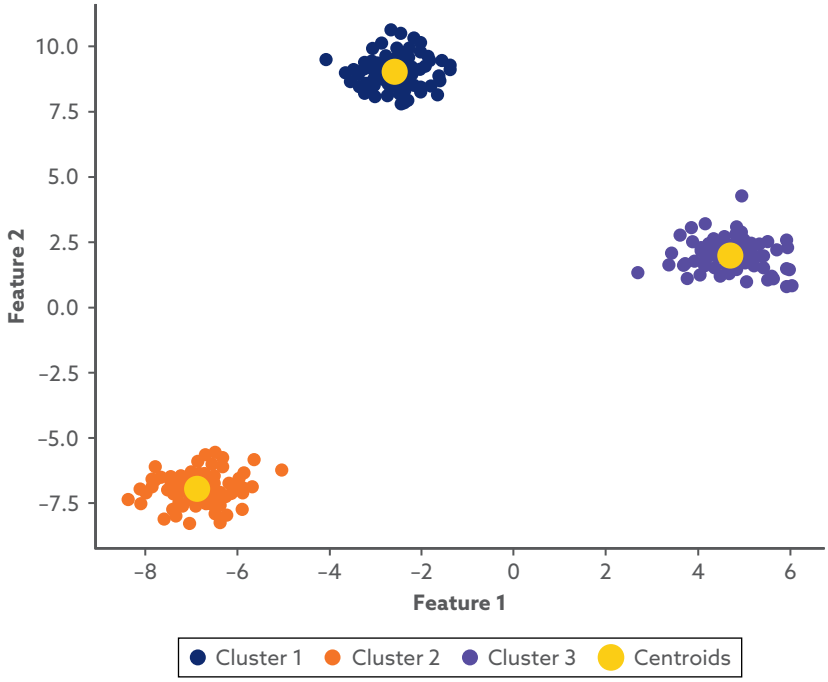
1. **Initialize *k* centroids randomly:** $\mu_1, \mu_2, \ldots, \mu_k$
2. **For iteration = 1 to max iterations:**
   - **a.** For each data point $x_i$ in $X$:
     - Calculate distance to each centroid: $d(x_i, \mu_j)$ for $j = 1$ to $k$
     - Assign $x_i$ to closest centroid: $c_i = \text{argmin}_j\, d(x_i, \mu_j)$
   - **b.** For each cluster $j = 1$ to $k$:
     - Update centroid: $\mu_j$ = mean of all points assigned to cluster $j$
   - **c.** If centroids have not changed significantly:
     - Break (convergence achieved)
3. **Output:** Cluster assignments and final centroids (see **Exhibit 1**)

An alternative clustering algorithm that provides a remedy to the limitations of *k*-means clustering is *spectral clustering*, which involves using matrix representations of finite graphs in order to determine the similarity between observations in a dataset (see **Figure 2**). Indeed, any set of observations or set of vectors of observations may be represented in graphical form. Spectral clustering can thus be viewed as a graph partition problem, in which clusters correspond to connected graph components.

The basic ideas behind spectral clustering were introduced in important papers by Hall (1970), Donath and Hoffman (1973), and Fiedler (1973). For a historical overview of spectral clustering, see Spielman and Teng (2007). Spectral clustering has also been applied in finance. In portfolio management and financial network analysis, spectral clustering excels at identifying groups of correlated assets, detecting market sectors based on complex interdependencies, and analyzing systemic risk by revealing the underlying network structure of financial markets where assets may be connected through indirect relationships that are not apparent in the original feature space. For example, Simonian and Wu (2019) used spectral clustering to build a regime-based trading model. They showed that their framework both produces predictively effective macro signals and classifies regimes in an economically intuitive way. In their framework, graph components are composed of vectors, with each vector consisting of respective values for growth, inflation, and leverage factors.

*Hierarchical clustering* creates a tree-like structure of clusters by merging smaller clusters into larger ones (agglomerative) or by splitting larger clusters into smaller ones (divisive).

## Exhibit 1. Clustering Output (*k*-means)




*Source:* All synthetic data created by author.

## Figure 2. Algorithm: Spectral Clustering

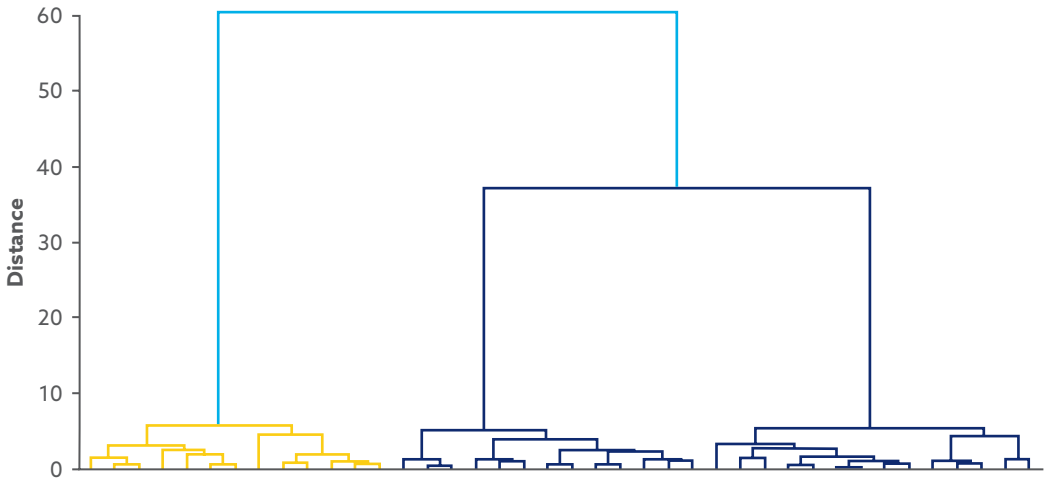
1. **Construct similarity matrix W:**
   - For each pair $(i, j)$: $\mathbf{W}[i, j] = \exp[-||x_i - x_j||^2/(2\sigma^2)]$
2. **Compute degree matrix D:**
   - $\mathbf{D}[i, i] = \Sigma_j \mathbf{W}[i, j]$, $\mathbf{D}[i, j] = 0$ for $i \neq j$
3. **Compute normalized Laplacian:**
   - $L_{norm} = \mathbf{D}^{(-1/2)} \times (\mathbf{D} - \mathbf{W}) \times \mathbf{D}^{(-1/2)}$
4. **Find *k* smallest eigenvectors of *L*:** $v_1, v_2, \ldots, v_k$
5. **Form matrix V** = $[v_1, v_2, \ldots, v_k]$ ($n \times k$ matrix)
6. **Normalize rows of V to unit length**
7. **Apply *k*-means clustering to rows of V**
8. **Output:** Cluster assignments

Agglomerative clustering is more common and works "bottom up" by initially treating each data point as a separate cluster and then iteratively merging the closest pair of clusters until all points belong to a single cluster or a desired number of clusters is reached (see **Figure 3**).

## Figure 3. Agglomerative Clustering Algorithm

1. **Initialize:** Each point as its own cluster $C = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$
2. **Compute distance matrix D between all pairs of points**
3. **While $|C| > 1$:**
   - **a.** Find closest pair of clusters $(C_i, C_j)$ using linkage method:
     - Single: $\min\{d(x, y) : x \in C_i, y \in C_j\}$
     - Complete: $\max\{d(x, y) : x \in C_i, y \in C_j\}$
     - Average: $\text{mean}\{d(x, y) : x \in C_i, y \in C_j\}$
   - **b.** Merge $C_i$ and $C_j$ into new cluster $C_k = C_i \cup C_j$
   - **c.** Update $C$ by removing $(C_i, C_j)$ and adding $C_k$
   - **d.** Update distance matrix **D** for new cluster $C_k$
   - **e.** Record merge in dendrogram
4. **Output:** Dendrogram structure (**Exhibit 2**)

## Exhibit 2. Hierarchical Clustering Dendrogram




*Note:* Different colors represent different clusters at different distance cutoffs.

The algorithm requires a distance metric between data points as well as a linkage criterion to measure distances between clusters—for example, single linkage (minimum distance), complete linkage (maximum distance), average linkage (average distance), or Ward linkage (minimizes within-cluster variance).

A well-known investment application of hierarchical clustering is Hierarchical Risk Parity (HRP), introduced by López de Prado (2016). HRP uses hierarchical clustering to infer relationships between assets, which are then used directly for portfolio diversification, addressing three major concerns of quadratic optimizers: instability, concentration, and underperformance. The approach departs from classical mean–variance optimization by using a three-step process that organizes assets into hierarchical clusters based on their correlation structure, reorganizes the correlation matrix according to this tree structure, and then allocates capital recursively through the hierarchy using inverse variance weighting within each cluster.

Another prominent technique is DBSCAN (density-based spatial clustering of applications with noise), a density-based clustering algorithm that groups together points in high-density areas while marking points in low-density regions as noise or outliers, making it particularly effective for discovering clusters of arbitrary shapes and sizes (Ester, Kriegel, Sander, and Xu 1996). The algorithm requires two parameters: epsilon ($\varepsilon$), which defines the neighborhood radius, and minimum points (MinPts), required to form a dense region. Core points have at least MinPts neighbors within $\varepsilon$ distance, border points within $\varepsilon$ distance of core points, and noise points not meeting either criterion. In contrast to *k*-means, DBSCAN does not require advance specification of the number of clusters and can identify clusters with irregular shapes, making it robust against outliers and noise. In financial applications, DBSCAN excels at fraud detection by identifying unusual transaction patterns, market anomaly detection, and customer behavior analysis where normal clustering algorithms might fail because of the presence of outliers or nonspherical cluster shapes that are common in financial data distributions.

An extension of DBSCAN is OPTICS (ordering points to identify the clustering structure), which creates an ordering of data points that represents the density-based clustering structure, providing more detailed insights into cluster hierarchies and varying density regions within the dataset (Ankerst, Breunig, Kriegel, and Sander 1999). The algorithm computes core distances and reachability distances for each point, creating a reachability plot that visualizes the clustering structure across different density thresholds, allowing analysts to extract clusters at multiple scales without specifying parameters in advance. This hierarchical approach is particularly valuable when dealing with clusters of varying densities or nested clusters or when the optimal clustering parameters are unknown because it provides a comprehensive view of the data's density structure. In high-dimensional financial data analysis, OPTICS proves invaluable for identifying complex market structures, detecting multiscale patterns in trading data, and analyzing portfolio correlations where traditional clustering methods might miss important structural relationships because of varying density patterns for different market conditions or time periods.

Affinity propagation (AP), introduced by Frey and Dueck (2007), belongs to the family of graph-theoretic clustering techniques and is based on the concept of "message passing" (Mézard 2007) between the candidate members of a cluster that continues until each candidate is sufficiently informed to join the appropriate cluster. AP begins by measuring the similarity, $s(\mathbf{i}, \mathbf{k})$, between vectors, which represents the similarity of vector **k** to vector **i**. Similarity is measured by a metric chosen by the model builder (e.g., Euclidean distance).

The basic input to AP is a real-valued number $s(\mathbf{k}, \mathbf{k})$, called a *preference*, for each observation. Observations with larger preference values are more likely to be selected as cluster centers, also known as *exemplars*. However, cluster selection is a function of not only preference size but also the two message-passing operations that are the essence of the AP algorithm. The first—the *responsibility*, $r(\mathbf{i}, \mathbf{k})$—is a message transmitted from observation **i** to a candidate exemplar **k** that expresses the suitability of observation **k** as an exemplar for observation **i** given the suitability of other candidate exemplars. The second—the *availability*, $a(\mathbf{i}, \mathbf{k})$—works in the opposite direction and is sent from a candidate exemplar **k** to an observation **i**. Availability expresses how appropriate it would be for observation **i** to select observation **k** as its exemplar, given the existing support that observation **k** has from other observations to serve as an exemplar. The AP process begins by initializing the availabilities to zero, $a(\mathbf{i}, \mathbf{k}) = 0$, and then proceeds to compute the responsibilities using the following rule:

$$r(\mathbf{i},\mathbf{k}) = s(\mathbf{i},\mathbf{k}) - \max_{\mathbf{k}' \text{ s.t. } \mathbf{k}' \neq \mathbf{k}} \{a(\mathbf{i},\mathbf{k}') + s(\mathbf{i},\mathbf{k}')\}. \tag{1}$$

The following formula then determines whether an observation is a good exemplar:

$$a(\mathbf{i},\mathbf{k}) = \min\left\{0, r(\mathbf{k},\mathbf{k}) + \sum_{\mathbf{i}' \text{ s.t. } \mathbf{i}' \neq \{\mathbf{i},\mathbf{k}\}} \max\{\{0, r(\mathbf{i}',\mathbf{k})\}\}\right\} \tag{2}$$

The "self-availability" of an observation is expressed as follows:

$$a(\mathbf{i},\mathbf{k}) = \sum_{\mathbf{i}' \text{ s.t. } \mathbf{i}' \neq \mathbf{k}} \max\{\{0, r(\mathbf{i}',\mathbf{k})\}\} \tag{3}$$

An investment application of AP is presented by Simonian (2020), who used it to determine the level of diversity within a set of investment signals. To classify signals according to their statistical predictive properties, the author posited a vector consisting of information coefficient (IC) and IC variance values in various regime-specific subsamples as inputs into the clustering algorithm. Using AP in this manner allows us to gain a multidimensional view of investment signal diversity, with each measure providing information on a different aspect of predictive effectiveness.

# Cluster Evaluation Techniques

Although many clustering algorithms require positing the number of clusters, techniques have been developed that allow the user to determine the most suitable clustering scheme. Two techniques in particular have become popular. The first is the *silhouette score*, an internal clustering evaluation metric that measures how similar each point is to points in its own cluster compared with points in other clusters, providing both individual point scores and an overall clustering quality measure. For each data point, the silhouette coefficient is calculated as $(b - a)/\max(a, b)$, where $a$ is the mean distance to other points in the same cluster (intracluster distance) and $b$ is the mean distance to points in the nearest neighboring cluster (intercluster distance). The silhouette coefficient ranges from −1 to 1. Values close to 1 indicate that the point is well matched to its cluster and poorly matched to neighboring clusters, values around 0 suggest that the point is on or very close to the decision boundary between clusters, and negative values indicate that the point might have been assigned to the wrong cluster.

The mathematical foundation of the silhouette score relies on distance-based cohesion and separation measures. For a point $i$ in cluster $C$, the intracluster distance, $a(i)$, represents the

average distance between point $i$ and all other points in the same cluster, measuring cluster cohesion. The intercluster distance, $b(i)$, is the minimum average distance from point $i$ to points in any other cluster, measuring cluster separation. The silhouette coefficient, $s(i) = [b(i) - a(i)]/\max[a(i), b(i)]$, provides a normalized measure that balances cohesion and separation, with higher values indicating better clustering quality.

The second popular cluster evaluation technique, the *Adjusted Rand Index* (ARI), introduced by Hubert and Arabie (1985), builds on the measure introduced by Rand (1971) and is a more explicitly probabilistic measure of cluster uniqueness. Two important characteristics distinguish an ARI value from a silhouette score. The first is that a Rand Index value is relational. Whereas a silhouette score tells us how tight a particular clustering scheme is, an ARI value ranges from 1 to −1 and tells us how similar two clustering schemes are. A value of 0 represents two independent clusters, and a value of 1 represents identical clusters. Negative values indicate worse-than-random clustering. Accordingly, the second distinguishing characteristic of an ARI value is that lower values indicate more unique pairs of clustering schemes. The metric is particularly valuable because it adjusts for the expected similarity that would occur by chance alone, making it more reliable than the basic Rand Index when comparing clusterings with different numbers of clusters or when dealing with imbalanced cluster sizes.

The mathematical foundation of the original Rand Index begins with the contingency table, which cross-tabulates the cluster assignments from two different clusterings. Given two clusterings $U = \{U_1, U_2, \ldots, U_r\}$ and $V = \{V_1, V_2, \ldots, V_s\}$, the contingency table entry $n_{ij}$ represents the number of objects that are in both cluster $U_i$ and cluster $V_j$. The Rand Index is calculated by counting the number of pairs of objects that are either in the same cluster in both clusterings or in different clusters in both clusterings and dividing by the total number of pairs. This raw measure does not account for the expected agreement that would occur by random chance, however, which is where the adjustment becomes crucial. The mathematical formula for ARI can be expressed as $\text{ARI} = (\text{RI} - \text{Expected}_{RI})/[\max(\text{RI}) - \text{Expected}_{RI}]$, where RI is the Rand Index, $\text{Expected}_{RI}$ is the expected value of the Rand Index under the null hypothesis of random clustering, and max(RI) is the maximum possible value of the Rand Index. This adjustment ensures that the expected value of ARI is zero when clusterings are independent, making it a more interpretable measure than the raw Rand Index.

# Dimension Reduction Techniques

Finance is a data-driven enterprise. Indeed, the sheer size of data processed and the number of variables considered in financial applications may at times test the limits of mathematical models and information technology infrastructure. Given this fact, reducing the dimensions of a problem when possible is a critical aspect of any investment process.

*Principal component analysis* (PCA) is a dimension reduction technique that has been used in finance for many years (Pearson 1901; Hotelling 1933). PCA transforms high-dimensional data into a lower-dimensional space while preserving maximum variance. PCA works by finding the principal components, which are orthogonal directions in the feature space that capture the most variance in the data. The algorithm computes the covariance matrix of the data, performs eigendecomposition to find eigenvectors (principal components) and eigenvalues (variance explained), and then projects the original data onto the space spanned by the top $k$ eigenvectors (see **Figure 4**). This linear transformation creates uncorrelated features ordered by the

## Figure 4. PCA Algorithm

1. **Center the data:** $X_{centered} = X - \text{mean}(X)$
2. **Compute covariance matrix:** $\mathbf{C} = [1/(n-1)] \times X_{centered}^{T} \times X_{centered}$
3. **Perform eigendecomposition:** $\mathbf{C} = \mathbf{V} \times \Lambda \times \mathbf{V}^{T}$
   - $\mathbf{V}$: eigenvectors (principal components)
   - $\Lambda$: eigenvalues (variance explained)
4. **Sort eigenvectors by decreasing eigenvalues**
5. **Select top *k* eigenvectors:** $W = \mathbf{V}[:, 0{:}k]$
6. **Transform data:** $Y = X_{centered} \times W$
7. **Output:** $Y$, $W$, explained variance ratio

amount of variance they explain, making PCA particularly useful for data visualization, noise reduction, and feature extraction in preprocessing pipelines.

Perhaps the most well-known example of a financial application of PCA appears in the decomposition of the yield curve by Litterman and Scheinkman (1991). Their application of PCA involves constructing a matrix where each row represents a specific date and each column represents yields at different maturities (such as 3-month, 6-month, 1-year, 2-year, 5-year, 10-year, and 30-year rates). PCA then decomposes the yield curve data into orthogonal components ranked by their explanatory power. Their study (and others that followed) revealed that three principal components explain approximately 95%–99% of yield curve movements:

- *First principal component (level):* This component typically accounts for 80%–90% of the variance and represents parallel shifts in the yield curve. When this factor moves, all yields tend to move up or down together by similar amounts. This behavior reflects broad monetary policy changes, inflation expectations, or general economic conditions.
- *Second principal component (slope):* Explaining roughly 5%–15% of variance, this component captures the steepening or flattening of the yield curve. It represents the spread between long-term and short-term rates, often reflecting expectations about future monetary policy or economic growth.
- *Third principal component (curvature):* Accounting for 1%–5% of variance, this component captures changes in the curve's convexity or "bow" shape. It reflects relative movements in medium-term rates compared with short- and long-term rates, often related to market expectations about intermediate-term economic conditions.

Another popular dimension reduction technique is *t-distributed stochastic neighbor embedding* (*t*-SNE), a powerful dimension reduction technique that can be used to visualize high-dimensional data in a lower-dimensional space, typically 2D or 3D (van der Maaten and Hinton 2008). It is particularly effective for exploring complex datasets and identifying clusters

## Figure 5. *t*-SNE Algorithm

1. **Compute high-dimensional similarities:**
   - For each point $x_i$, calculate conditional probabilities: $p_{\{j|i\}} = \exp(-||x_i - x_j||^2/2\sigma_i^2)/\Sigma k \neq i \exp(-||x_i - x_k||^2/2\sigma_i^2)$
   - Symmetrize: $p_{ij} = (p_{\{j|i\}} + p_{\{i|j\}})/2n$
2. **Initialize low-dimensional embedding:**
   - Randomly place points $y_i$ in target dimensional space
3. **Iterative optimization: For each iteration, compute low-dimensional similarities:**
   - Use *t*-distribution: $q_{ij} = (1 + ||y_i - y_j||^2)^{-1}/\Sigma k \neq l\ (1 + ||y_k - y_l||^2)^{-1}$

   Calculate gradient:
   - Minimize Kullback–Leibler (KL) divergence: $KL(P||Q) = \Sigma_{ij}\ p_{ij} \log(p_{ij}/q_{ij})$
   - Gradient: $\partial C/\partial y_i = 4\Sigma_j\ (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$

   Update positions:
   - Apply gradient descent with momentum to move points $y_i$
4. **Output:** Final low-dimensional embedding $Y$

or patterns that might not be apparent in raw data. In finance, *t*-SNE can be applied to analyze and visualize market segmentation, such as grouping stocks or assets based on their historical performance, risk profiles, or other features. The algorithm for *t*-SNE appears in **Figure 5**.

To provide a supervised counterpoint, we mention *linear discriminant analysis* (LDA), a technique that finds linear combinations of features that best separate different classes, making it particularly useful for classification preprocessing (Fisher 1936). LDA can serve as a powerful classification tool in financial applications, particularly for identifying trading signals. In credit risk modeling, LDA can help separate borrowers into distinct risk categories using financial ratios and other predictive variables, optimizing the linear combination of features that best discriminates between default and nondefault cases. For algorithmic trading, LDA can be used to classify market conditions into bullish, bearish, or neutral regimes based on technical indicators and market microstructure variables.

Another technique, *independent component analysis* (ICA), assumes that data are generated by mixing independent source signals and attempts to recover these original independent components, making it valuable for blind source separation problems such as audio signal processing (Comon 1994; Bell and Sejnowski 1995). ICA is therefore useful in financial applications that require blind source separation, particularly in identifying independent market factors from mixed signals. For example, in multiasset portfolio analysis, ICA separates returns into independent components that may represent different economic factors (inflation, growth, sentiment) that are not directly observable but drive asset performance. For high-frequency trading,

ICA helps isolate genuine price signals from market noise by identifying independent sources of price movement. The technique is particularly valuable in emerging markets, where traditional factor models may not apply, because ICA can discover country-specific or sector-specific independent factors that influence asset returns without requiring prior assumptions about factor structure.

## Deep Learning Approaches

*Autoencoders* are neural networks designed to learn efficient data representations in an unsupervised manner by training the network to reconstruct its input data (Hinton and Salakhutdinov 2006). The architecture consists of an encoder that compresses the input into a lower-dimensional latent representation and a decoder that reconstructs the original input from this compressed representation. The network is trained to minimize reconstruction error, which forces it to learn meaningful features that capture the most important aspects of the data. Applications of encoders include dimensionality reduction, denoising, feature learning, and data compression. The algorithm for autoencoders is shown in **Figure 6**.

Variations of standard encoders include denoising autoencoders that learn to reconstruct clean data from corrupted inputs, variational autoencoders (VAEs) that learn probabilistic latent

### Figure 6. Autoencoder Algorithm

1. **Initialize networks:**
   Encoder: $f_{enc}(x; \theta_{enc}) \rightarrow z$ with parameters $\theta_{enc}$
   Decoder: $f_{dec}(z; \theta_{dec}) \rightarrow \hat{x}$ with parameters $\theta_{dec}$
2. **Training loop (for each epoch):**
   For each mini-batch:
   Forward pass:
   Encode: $z = f_{enc}(x; \theta_{enc})$—compress input to latent code
   Decode: $\hat{x} = f_{dec}(z; \theta_{dec})$—reconstruct from latent code
3. **Loss computation:**
   Reconstruction loss: $L = ||x - \hat{x}||^2$ (mean squared error)
4. **Backward pass:**
   Compute gradients with respect to both networks: $\nabla\theta_{enc}L$, $\nabla\theta_{dec}L$
   Update encoder: $\theta_{enc} \leftarrow \theta_{enc} - \alpha\nabla\theta_{enc}L$
   Update decoder: $\theta_{dec} \leftarrow \theta_{dec} - \alpha\nabla\theta_{dec}L$
5. **Output:** Trained encoder and decoder networks

representations, and sparse autoencoders that enforce sparsity constraints on the hidden layer activations to learn more interpretable features (Kingma and Welling 2014). Unlike traditional autoencoders, which compress data into a deterministic latent space, VAEs model the latent space as a probability distribution (typically Gaussian). This probabilistic approach allows VAEs to generate realistic synthetic data by sampling from the learned latent distribution. VAEs consist of two main components:

- *Encoder:* Maps input data to a latent space by learning the parameters (mean and variance) of a probability distribution
- *Decoder:* Reconstructs the original data from samples drawn from the latent distribution

The key innovation of VAEs is the use of variational inference, where the model learns a latent distribution that captures the underlying structure of the data. The training process involves minimizing a loss function that combines reconstruction error (how well the decoder reconstructs the input) and a regularization term (how close the learned latent distribution is to a prior, typically a standard normal distribution). Examples of how VAEs are used in finance include synthetic data generation for testing trading strategies or stress-testing models, uncovering latent factors driving asset prices or market behavior, and identifying unusual patterns in financial data by comparing reconstructed data with the original input.

Another algorithm used to generate synthetic data is known as a *generative adversarial network* (GAN; Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, and Bengio 2014). GANs are a class of generative models that use a game-theoretic framework to learn and generate new data that mimic the distribution of a given dataset. GANs consist of two neural networks:

- *Generator:* Creates synthetic data from random noise, attempting to mimic the real data distribution
- *Discriminator:* Distinguishes between real data (from the dataset) and fake data (produced by the generator)

The generator and discriminator are trained simultaneously in a zero-sum game: The generator attempts to "fool" the discriminator by producing realistic data, and the discriminator tries to correctly identify real versus fake data. Over time, the generator improves its ability to create realistic data, and the discriminator becomes better at distinguishing real from fake. When the GAN converges, the generator produces data that are indistinguishable from the real data. GANs are widely used in finance for tasks that involve generating realistic synthetic data, modeling complex distributions, and simulating market scenarios. Simonian (2024) describes how the synthetic data generated by GANs can be used in the model validation process.

## Anomaly Detection

Anomaly detection is an important part of finance because extreme outliers, such as stock market crashes, can have outsized financial and economic implications. However, anomaly detection is important not only for investors but also in such areas as credit card fraud detection, identifying unusual trading patterns, detecting market manipulation, and monitoring portfolio performance for abnormal behavior patterns that could indicate operational risks or model failures. Although other types of machine learning models, such as DBSCAN and support vector

## Figure 7. LOF Algorithm

1. **Find *k*-nearest neighbors for each point**
2. **Calculate reachability distance:**
   $r_{dist}(A, B) = \max[k\text{-distance}(B), d(A, B)]$
3. **Compute local reachability density:**
   $Local_{rd}(A) = 1/(\Sigma Reach_{dist}(A, B)/|N_k(A)|)$
4. **Output:** $\text{LOF}(A) = [\Sigma I_{rd}(B)/I_{rd}(A)]/|N_k(A)|$ for $B$ in $N_k(A)$

machines, can perform anomaly detection, dedicated algorithms have also been developed for outlier detection. *Isolation Forest* uses decision trees to isolate anomalies by randomly selecting features and splitting the data according to threshold values, operating under the principle that anomalies are few and different—hence, easier to isolate (Liu, Ting, and Zhou 2008). The algorithm constructs an ensemble of isolation trees by randomly selecting features and split values, with anomalies requiring fewer splits to be isolated and thus having shorter average path lengths in the trees, whereas normal points require more splits and have longer paths. This approach is particularly effective because it directly targets anomalies rather than profiling normal instances, making it computationally efficient with linear time complexity and effective in high-dimensional spaces.

Another popular anomaly detection technique is the *local outlier factor* (LOF), which detects anomalies by comparing how densely packed each point is relative to its local neighborhood (Breunig, Kriegel, Ng, and Sander 2000). The key insight is that outliers exist in sparser regions compared with their neighbors. As a rough analogy, think of data points as houses in a city: In dense neighborhoods, houses are close together, and in less populated areas, houses are spread out. An outlier is like a house that is unusually isolated compared with the density of its surrounding neighborhood. The LOF algorithm is shown in **Figure 7**.

# Conclusion

In this chapter, I have provided an overview of some major unsupervised learning algorithms along with examples of how they are applied in various areas of finance. I have shown that unsupervised learning plays a major role in classification, anomaly detection, and synthetic data generation—all major application areas in finance. The next chapter will delve deeper into an area of unsupervised learning—network theory—that has become popular in recent years because of its flexibility and power to illuminate various types of connections that exist between financial and economic entities and actors.

# References

Ankerst, Mihael, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. "OPTICS: Ordering Points to Identify the Clustering Structure." *SIGMOD Record* 28 (2): 49–60. doi:10.1145/304181.304187.

Bell, Anthony J., and Terrence J. Sejnowski. 1995. "An Information-Maximization Approach to Blind Separation and Blind Deconvolution." *Neural Computation* 7 (6): 1129–59. doi:10.1162/neco.1995.7.6.1129.

Breunig, Markus M., Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. "LOF: Identifying Density-Based Local Outliers." *Proceedings of the 2000 ACM SIGMOD Record*: 93–104. doi:10.1145/342009.335388.

Comon, Pierre. 1994. "Independent Component Analysis, a New Concept?" *Signal Processing* 36 (3): 287–314. doi:10.1016/0165-1684(94)90029-9.

Donath, W. E., and A. J. Hoffman. 1973. "Lower Bounds for the Partitioning of Graphs." *IBM Journal of Research and Development* 17 (5): 420–25.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." *KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*: 226–31.

Fiedler, M. 1973. "Algebraic Connectivity of Graphs." *Czechoslovak Mathematical Journal* 23 (98): 298–305.

Fisher, Ronald A. 1936. "The Use of Multiple Measurements in Taxonomic Problems." *Annals of Eugenics* 7 (2): 179–88. doi:10.1111/j.1469-1809.1936.tb02137.x.

Frey, Brendan J., and Delbert Dueck. 2007. "Clustering by Passing Messages Between Data Points." *Science* 315 (5814): 972–76. doi:10.1126/science.1136800.

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative Adversarial Nets." *Proceedings of the 27th International Conference on Neural Information Processing Systems* 2: 2672–80.

Hall, K. M. 1970. "An *r*-Dimensional Quadratic Placement Algorithm." *Management Science* 17 (3): 219–229.

Hinton, Geoffrey E., and Ruslan Salakhutdinov. 2006. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313 (5786): 504–07. doi:10.1126/science.1127647.

Hotelling, Harold. 1933. "Analysis of a Complex of Statistical Variables into Principal Components." *Journal of Educational Psychology* 24 (6): 417–41. doi:10.1037/h0071325.

Hubert, Lawrence, and Phipps Arabie. 1985. "Comparing Partitions." *Journal of Classification* 2 (1): 193–218. doi:10.1007/BF01908075.

Kingma, D. P., and M. Welling. 2014. "Auto-Encoding Variational Bayes." arXiv: 1312.6114.

Litterman, Robert B., and Josè Scheinkman. 1991. "Common Factors Affecting Bond Returns." *Journal of Fixed Income* 1 (1): 54–61. doi:10.3905/jfi.1991.692347.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. 2008. "Isolation Forest." *Proceedings of the 8th IEEE International Conference on Data Mining*: 413–22. doi:10.1109/ICDM.2008.17.

Lloyd, S. P. 1982. "Least Squares Quantization in PCM." *IEEE Transactions on Information Theory* 28 (2): 129–137.

López de Prado, Marcos. 2016. "Building Diversified Portfolios That Outperform Out of Sample." *Journal of Portfolio Management* 42 (4): 59–69. doi:10.3905/jpm.2016.42.4.059.

Mézard, Marc. 2007. "Where Are the Exemplars?" *Science* 315 (5814): 949–51. doi:10.1126/science.1139678.

Pearson, Karl. 1901. "On Lines and Planes of Closest Fit to Systems of Points in Space." *London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2 (11): 559–72. doi:10.1080/14786440109462720.

Rand, William M. 1971. "Objective Criteria for the Evaluation of Clustering Methods." *Journal of the American Statistical Association* 66 (336): 846–50. doi:10.1080/01621459.1971.10482356.

Simonian, Joseph. 2020. "Modular Machine Learning for Model Validation: An Application to the Fundamental Law of Active Management." *Journal of Financial Data Science* 2 (2): 41–50. doi:10.3905/jfds.2020.1.027.

Simonian, Joseph. 2024. *Investment Model Validation: A Guide for Practitioners*. Charlottesville, VA: CFA Institute Research Foundation. doi:10.56227/24.1.15.

Simonian, Joseph, and Chenwei Wu. 2019. "Minsky vs. Machine: New Foundations for Quant-Macro Investing." *Journal of Financial Data Science* 1 (2): 94–110. doi:10.3905/jfds.2019.1.004.

Spielman, D.A. and S.-H. Teng. 2007. "Spectral Partitioning Works: Planar Graphs and Finite Element Meshes." *Linear Algebra and its Applications* 421 (2): 284–305.

van der Maaten, Laurens, and Geoffrey Hinton. 2008. "Visualizing Data Using t-SNE." *Journal of Machine Learning Research* 9 (86): 2579–605.

Wu, Dingming, Xialong Wang, and Shaocong Wu. 2022. "Construction of Stock Portfolios Based on *k*-Means Clustering of Continuous Trend Features." *Knowledge-Based Systems* 252 (27 September). doi:10.1016/j.knosys.2022.109358.