Hierarchical Clustering of Times Series using Gaussian Mixture Models and Variational Autoencoders

Per Wilhelmsson

Master's thesis
2019:E27

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Abstract

This thesis proposes a hierarchical clustering algorithm for time series, comprised of a variational autoencoder to compress the series and a Gaussian mixture model to merge them into an appropriate cluster hierarchy. This approach is motivated by the autoencoders good results in dimensionality reduction tasks and by the likelihood framework given by the Gaussian mixture model. In contrast to similar clustering algorithms, this algorithm tries to answer the question of the true number of clusters in the data set and gives superior visualisation possibilities. Furthermore, the thesis shows how cluster analysis in general can be applied to several interesting problems in finance, and specifically how this algorithm is engineered to outperform standard clustering algorithms on these problems.

Acknowledgements

I would like to express my gratitude to Kidbrooke Advisory AB for providing me with a general thesis idea, data to work with, a work place, good company and lots of support regarding technical issues and abstract ideas. My gratitude is specifically directed to my supervisor at Kidbrooke Advisory AB, Björn Bergstrand, for his time and diligent and thorough input on this thesis. I would also like to express my gratitude towards Filip Persson Mörk, for giving me many of the ideas implemented in this thesis and his enthusiastic involvement in the process, as well as Edvard Sjögren for his time and knowledge on important issues. At last, I would like to thank my university supervisor Magnus Wiktorsson for valuable input on theoretical aspects and problems related to the thesis structure in general.

Contents

Chapter 1

Introduction

1.1 Background

Consider two stocks that move together at nearly every point in time. When the first one goes down, the other goes down. When the second stock moves up, the first one follows. Despite having different names, is it reasonable to differentiate between them from a mathematical point of view?

The question whether two objects, abstract or concrete, belong to the same category is an old and prevalent question in many fields of human life. Two similar, but not identical objects, may differ in subtle but crucially important aspects. Or they may differ only due to their different trajectories through time, like two oaks growing their branches in their particular way to accommodate themselves in a complex environment, but unambiguously belong to the same family of oaks.

This thesis will touch on the statistical method of *clustering*, which in short is to find patterns in data, categorising similar objects in groups and distinguish dissimilar objects by putting them into different groups. Specifically, it will do so in the setting of financial assets using a combination of agglomerative hierarchical clustering, Gaussian mixture models (GMM) and a novel statistical method called Variational Autoencoders (VAE).

1.2 Motivation

If two stocks are highly dependent, it makes a lot of sense to put them together when analysing the market. The probability that they consistently move together by chance is very small. Instead, it is reasonable to assume that they vary together due to some common underlying factor such as belonging to the same market segment or even being owned by the same corporation.

For example, imagine that you are constructing your own portfolio of financial assets. Then, dependence structures may be utilised to form advantageous trading and investment strategies. This is done by basically every mathematically founded portfolio optimisation strategy, most famously by Markowitz's Critical Line Algorithm (CLA) from 1954 [1]. The basic goal of portfolio optimisation is to find the maximal expected return on the investment while keeping the risk low. If two assets are strongly correlated, a basic algorithm like CLA will skew the portfolio weights very harshly in favour of the better performing asset [1]. For example, consider two stocks A and B that move exactly the same every day, but suddenly diverges at one day, stock A moves up and stock B moves down. In this setting, CLA would say that a portfolio with both A and B should contain almost only A. From a human investors perspective, this may in fact seem very risky. Is A *really* a safer and better performing asset to hold over B, or was this outcome an instance of randomness?

A sound risk management strategy is to strive for *robustness*, which means that small changes in the underlying data should not result in large changes in the outcome. In the example of the two stocks A and B, the outcome changed dramatically due to an incident on the single day when A and B deviated. Before that day, CLA would put equal weights on both stocks. Thus, CLA can't be considered to be a robust algorithm.

One intuitive way of dealing with this lack of robustness is to categorise assets that are similar into clusters and manually assign conservative weights on every member of the cluster. In a setting where there are thousands of additional stocks, A and B would by every measure be categorised into the same cluster and treated in a more robust way than CLA is treating them.

Applications of cluster analysis

There are plenty of application areas for asset clustering in the financial industry. The following list asks some relevant questions that motivate this thesis and that will be followed up through out the text.

- Is it possible to create more robust, intuitive and better performing portfolios with less computation time by grouping assets into clusters and optimise over the clusters?
- If assets are sorted into a hierarchical clustering scheme, is it possible to extract useful and valuable information? For example, are there any unintuitive relationships? Is it possible to get a graspable and intuitive understanding of the market by putting it into a hierarchy? Or are there any dominance relationships in the market? It is reasonable to expect that a clustering hierarchy helps to paint a picture of the marketplace and helps with any type of qualitative analysis.
- How many "true" clusters are there in the market? Trivially, there are as many clusters as unique funds in the market place. But it is also reasonable to think that many assets can be described by a more fundamental process without great loss in detail. How many such processes are there in the market place?
- If clusters exists, are they stable over time? Is it possible to find some underlying process that can help predict if two assets are likely to continue to move together also in the future?
- If clusters exists, is it also possible to say which members are most likely to be good representatives of the cluster as a whole or to say which members are more likely to be outliers? Is it possible to quantitatively say if there are assets that should belong to two clusters because they resemble not only members in the assigned cluster, but also resemble other assets?
- Is it possible to gain information about future movements of an asset by looking at similar assets? If a stock goes haywire over a period of time, it might be reasonable to look at similar stocks with normal trajectories to gain insight about a typical movement rather than just considering the spectacular movements the stock showed during this period of time.
- Once a cluster has been found, is it possible to visualise it in relation to other clusters in a way that is easy to grasp and understand? Many clustering problems are by nature high dimensional and thus hard to visualise and communicate. Is there a reliable and accurate way to do this without too much efforts?

1.3 Thesis contribution

Clustering in general is a widely used and recognised method of finding patterns in any kind of data sets. However, there are less explored clustering problems where common cluster algorithms might not work optimally. By tailoring a new algorithm to a specific problem type, large improvements in performance can be made.

This thesis will contribute to the field of clustering by exploring the intersection of the following subset of problems.

- **Clustering of time series**. Time series differ from other data sets in a number of ways that force clustering algorithms to operate in slightly different ways than otherwise. There are several algorithms to specifically cluster time series. Some will be presented in the next chapter.
- **Visualisation of time series**. Time series are notoriously high dimensional and relationships between different time series can be hard to visualise and communicate. By applying state of the art dimensionality reductions techniques, visualisation and clustering will become more intuitive and understandable.

- **How many clusters are there**? All standard clustering algorithms avoid to answer this question, either by requiring a prespecified number of clusters or by sorting the data into a hierarchy, effectively leaving this question unanswered. By including a likelihood framework, this thesis will give a suggestion of how this question can be answered.
- **Including a soft clustering framework in hierarchical clustering**. The traditional hierarchical clustering algorithms, presented in section 2.4, use a hard clustering framework, i.e. assign every point to one and only one cluster. Soft clustering algorithms can assign a data point to multiple clusters by giving it a likelihood assignment for every cluster.
- **Utilising a variational autoencoder in cluster analysis**. Autoencoders have received a lot of attention with the growing interest in artificial neural networks. Autoencoders can be used for non-linear dimensionality reduction and is thus suitable for preprocessing data before clustering. If a variational autoencoder is used, it is possible to extract further information about the affinity between data.

1.4 Objectives

There is actually nothing that prevents a straightforward use of an agglomerative hierarchical algorithm on a set of time series, but that approach would leave many of the questions above unanswered. The purpose of this thesis is to develop a tailored algorithm that takes a perspective on the questions posed above and compare it with a correlation based agglomerative hierarchical clustering algorithm. The two approaches will be compared in a few tests specified in chapter 4 and evaluated in chapter 5 and 6. The thesis will be limited to time series in general and discuss how different types of time series needs to be approached. The main focus will be on financial time series. There are no obvious obstacles impeding the framework to be applied on other types of data sets, but it will not be investigated in this thesis.

1.5 Thesis outline

In addition to this introductory chapter, the following chapters are included in the thesis.

Chapter 2: Theory

An overview of relevant theory and earlier work related to the topic is presented. The chapter starts with presenting what clustering is and some common approaches. It then introduces the theory behind artificial neural networks and concludes with theory necessary for testing the algorithm.

Chapter 3: Model building

This chapter gives a detailed map over the algorithm architecture. An autoencoder is build and crucial details are discussed. Thereafter an agglomerative hierarchical Gaussian mixture model is integrated into the autoencoder output.

Chapter 4: Tests and data

When the algorithm is implemented, it needs to be tested to ensure that the thesis objectives are fulfilled. This chapter presents data sets and specific tests that are conducted to investigate the functionality of the algorithm.

Chapter 5: Results

The results from the tests specified in chapter 4 are presented with plots of clustered time series, tables with measurements and results from hypothesis tests.

Chapter 6: Discussion

The results from chapter 5 are discussed and evaluated. Good results are be compared with bad outcomes and a discussion of when to use and when to not use the algorithm is presented. Lastly, conclusions are made and some ideas for future work are presented.

Chapter 2

Theory

2.1 Clustering

Clustering is an unsupervised statistical learning method for detecting patterns in data [2] and is encountered in all branches of science, e.g. psychology, anthropology, finance and engineering. [3][4][5]. The overall goal of clustering is to group objects according to two properties.

- **Property 1**: Two similar objects should be grouped into the same cluster.
- **Property 2**: Two dissimilar objects should be grouped into different clusters.

Whether an object is similar or dissimilar is usually measured by a distance measure that captures the notion of similarity. The term *unsupervised* means that the user is only in control of the data set X and does not posses the response variable Y, i.e. the true labels or categories. The objective is thus to learn patterns in X without the use of Y. The true categories Y are what the user is searching for. The term unsupervised is in opposition to the term *supervised*, which means that the data is labelled and methods can thus be trained and evaluated using the given labels.

Cluster analysis consists of two major parts: 1) A distance function 2) a clustering algorithm. A distance function resembles the notion of how close two data points are to each other. The set of distances is the raw material that the algorithms are fed with and thus plays a crucial role in cluster analysis. Some [6] even argue that a proper distance function is more important than the algorithm. If an effective distance function is used, data points within a true cluster will be close to each other and no false clusters will appear, which makes almost any algorithm able to find a global minimum to the objective function, equation 2.2. Depending on the problem at hand, different distance functions can be used.

A proper distance measure is defined [7] as a function $d(x, y)$ such that.

- $d(x, x) = 0 \ \forall \ x$
- $d(x, y) > 0$ for $x \neq y$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(z, y)$

The clustering algorithm takes the set of pairwise distances and determines a proper clustering of the data set. Many algorithms separates the problem of finding the best number of clusters K from finding the optimal clustering given that K is pre-specified and focuses on the latter, e.g. the famous clustering algorithm *K-means*. It is possible to iterate over several K and compare the clustering. For example, let K be specified on an interval $K \in \{1, \cdots, \hat{K}\}$, where $\hat{K}$ is the maximum amount of clusters tried in the iteration. Then, the number of distinct assignments is [8]

$$S(N, \hat{K}) = \frac{1}{\hat{K}!} \sum_{k=1}^{\hat{K}} (-1)^{\hat{K}-k} \binom{\hat{K}}{k} k^N \tag{2.1}$$

where N is the number of data points. The number of distinct assignments grows very fast with both N and $\hat{K}$. A modest clustering problem with $N = 200$ and $\hat{K} = 10$ results in $S(200, 10) \approx 3 \cdot 10^{193}$, which stretches beyond any imaginable processing power. A clustering algorithm visits only a small fraction of possible assignments and usually does so in a greedy descent fashion. This frequently leads to suboptimal solutions and different algorithms may give radically different results. The good news is that algorithms may be engineered for specific problems with good result. The bad news is that it often is rather easy to come up with an adversarial example which the algorithm won't be able to solve, which makes it very hard to generalise and automate clustering problems.

2.1.1 Objective function and evaluation

Let K be the number of clusters that are to be found, indexed $k \in \{1, \ldots, K\}$ and let $C(i)$ denote the cluster index assigned to the ith data point. To form an optimal clustering with K clusters, the *within cluster distance* $W_K(C)$ is minimised [8], where $d(x_i, x_{i'})$ is the distance between the two data points i and j.

$$W_K(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(j)=k} d(x_i, x_j) \tag{2.2}$$

Another approach is to maximise the *between cluster distance* $B(C)$.

$$B_K(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(j)\neq k} d(x_i, x_j) \tag{2.3}$$

These two approaches are equivalent. Let To be the total distance between all points. Since the data X is fixed, the total distance cannot change and is thus constant. Thus, minimising $W(C)$ is equivalent to maximising $B(C)$.

$$To = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} d(x_i, x_j) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \Big[\sum_{C(j)\neq k} d(x_i, x_j) + \sum_{C(j)=k} d(x_i, x_j) \Big] = W_K(C) + B_K(C) \tag{2.4}$$

If K is set to the number of data points N, the objective function $W_N(C)$ is trivially equal to zero since every single data point is seen as a cluster. However, this does not provide any information about underlying structures and patterns in the data, which are the ultimate objectives to reveal. The challenge is to set $K << N$ and force the clusters to symbolise meaningful and relevant features within the data. Thus, the objective function is a tool to reach an end rather than the end goal itself.

2.1.2 The problem of K, choosing the number of clusters

As seen above, the number of clusters K is crucial to the task of clustering and most clustering algorithms works only if given a pre-specified K. Thus, finding the optimal number of clusters is a separate problem which can be investigated at an earlier stage or be integrated in the clustering algorithms using some iterative approach. For example, the problem at hand may suggest a problem specific number of clusters or different number of clusters can be tried and the objective function $W_K(C)$ can be evaluated for all K. Since $W_N(C) = 0$, a more sophisticated method than minimising $W_K(C)$ with respect to K has to be applied.

Gap Statistic

The *gap statistic* [8] compares the curve $\log(W_K(C))$ to the curve obtained when using uniformly sampled data over a hypercube covering the data. If there actually are clusters within the data, $\log(W_K(C))$ will be lower than $\log(W_{K,Uniform}(C))$. When the distance between the two curves is maximised, the clustering has managed to find a suitable number of clusters.

Define the gap curve $G(K) = \log(W_{K,Uniform}(C)) - \log(W_K(C))$. Also define s_K to be the sample standard deviation of $\log(W_{K,Uniform}(C))$ over n simulations and $s_K^{'} = s_K\sqrt{1+\frac{1}{n}}$. The optimal K is then defined as

$$K^* = \text{argmin}_K\{K \mid G(K) \geq G(K+1) - s_{K+1}^{'}\} \tag{2.5}$$

Due to the sampled curve $\log(W_{K,Uniform}(C))$, the gap statistic is a simulation based method and will suffer from the curse of dimensionality. When clustering high dimensional data, the covering hypercube will be very, very sparse even when a huge sample is drawn. For this method to work properly, a dimensionality reduction should be applied before analysing.

Bayesian hierarchical clustering

Bayesian hierarchical clustering (BHC) is a clustering algorithm introduced by Katherine Heller and Zoubin Ghahramani in 2005 that uses a bayesian approach to determine the number of clusters in a data set [9]. The algorithm uses marginal likelihoods to determine what clusters to merge and when to stop merging clusters. Along with the marginal likelihood, a Dirichlet process mixture model is used as prior and efficient hypothesis testing is made at every merging step. Thereafter, Bayes theorem is invoked and the best performing cluster merge is chosen. All details are found in the article. Although easy to implement and understand, it may suffer from numerical errors for large data sets. The algorithm includes multiplication and addition of extremely large numbers with extremely small numbers. This may effect the decision making process and result in bad performance.

Likelihood based methods

Likelihood based methods for choosing model orders are very common in many statistical fields, e.g. regression and auto-regressive models [10][11], but are seldom used in clustering. The apparent reason is that clustering is usually not seen from a probabilistic perspective. Instead, it is seen as a optimisation problem, as outlined above. However, a probabilistic perspective can be taken by replacing distances with likelihood, and the optimisation problem is reframed as a maximum likelihood problem. For example, this can be done by using a Gaussian Mixture Model (GMM), outlined in section 2.3, where a distance to the cluster centre is replaced by the likelihood of the point given the cluster (the Gaussian distribution). The framework for likelihood models can then be utilised, including Bayesian Information Index (BIC), Akaike Information Criteria (AIC) and Likelihood Ratio Test [10].

A Likelihood Ratio Test is asymptotically $\chi^2(f)$-distributed, where f is the difference in model parameter between the larger and the smaller model. Because it is *asymptotically* $\chi^2(f)$-distributed, the likelihood ratio test may perform poorly when the number of data points in the merged cluster is low. Thus, a likelihood ratio test is not likely to perform well when true clusters are small, which makes it impractical for distinguishing between some clustering models. This is also true for BIC and AIC, since they are both derived from approximations that require the number of data points to be much larger than the number of parameters estimated.

2.1.3 Dimensionality reduction in clustering

Although clustering can be used as dimensionality reduction in it self, other methods are usually used to reduce the dimensionality of the data before using clustering algorithms. A big challenge in cluster analysis is the "curse of dimensionality". To illustrate why, imagine a two dimensional grid with a center point and an enclosing square. The $4 = 2^2$ corner points of the square are all of equal distance from the center point. In three dimensions, the square turns into a cube with $8 = 2^3$ equidistant corners. Now, imagine a d-dimensional space. The corresponding hypercube would now have 2^d equidistant corner. For $d = 100$, which is a small dimensionality for e.g. time series, there would be about $1.26 \cdot 10^{30}$ corners. This means that there are plenty of ways for objects to be on equal distance to each other and many ways to form clusters. A remedy to this curse is to apply dimensionality reduction techniques before clustering, which speeds up computation as well as reduces the complexity of the space. Dimensionality reduction works very well if the high dimensional data lies in the vicinity of a low dimensional manifold, which arguably is the case in many data sets. The most used dimensionality reduction techinque is Principal Component Analysis (PCA). PCA projects the high dimensional data onto a latent space of lower dimension so that the maximal amount of variance is perserved in the latent space.

During recent years, autoencoders have been used for dimensionality reduction with good results [12]. Autoencoders are presented in section 2.7.

2.2 Measuring cluster quality

If there is a reference cluster, either a true clustering or a clustering from another algorithm, metrics such as Purity Index, Rand Index, and Mutual Information can be utilised to evaluate the similarity of two clusterings. For this section, define two arbitrary clusterings U and V, and let N be the total number of data points used for both clusterings. All metrics are from [13].

Purity

Define $p_{U,V}(i,j) = \frac{|u_i \cap v_j|}{N}$, which is the probability that a random data point belongs to cluster i in clustering U and to cluster j in clustering V, and $p_U(i) = \frac{|u_i|}{N}$ to be the fraction of data points in cluster i in U. Now, look at cluster i in U and find the most prevalent cluster from V within i. Define p_i to be the ratio of the prevalent cluster j in i, i.e. the *purity* of the cluster i. Mathematically, p_i is defined as $p_i = \max_j \frac{p_{U,V}(i,j)}{p_U(i)}$.

Now, the weighted mean purity $P(U,V)$ of a clustering is

$$P(U,V) = \sum_{i=1}^{K} \frac{p_U(i)}{N} p_i \tag{2.6}$$

Purity measures how pure U is given that V is the true classification. If neither V nor U are true classifications, they are interchangeable. However, P is not symmetrical with respect to U and V. For example consider the following two clusterings with two clusters each.

$$U = \{1,1,1,1,2,2\}$$

$$V = \{2,2,2,1,2,2\}$$

Here, $P(U,V) = \frac{4}{6}\frac{3}{4} + \frac{2}{6}\frac{2}{2} = \frac{5}{6}$. But $P(V,U) = \frac{1}{6}\frac{1}{1} + \frac{5}{6}\frac{3}{5} = \frac{2}{3}$. When the objective is to measure e.g. clustering stability over time, a mean value of the two can be used.

$$P_{mean} = \frac{P(U,V) + P(V,U)}{2}$$

.

Rand Index

In the begin of this chapter, clustering was described as the process of putting similar points in the same cluster and dissimilar points in different clusters. When comparing two clusterings U and V, imagine giving one point to every pair of points that stay in the same cluster in both U and V. Denote the total of such pairs as TP (*true positive*). Then also give one point for every pair that stay in different clusters in U and in V and denote the total number of such pairs as TN (*true negative*) . No points are given to pairs that are in the same cluster in U but not in V, or are in the same cluster in V but not in U. If there are N points in the data set, there are $\frac{N(N-1)}{2}$ pairs of points in the data set. The *Rand Index* is given by

$$RandIndex = \frac{TP + TN}{N(N-1)/2} \tag{2.7}$$

The Rand Index measures the ratio of number of pairs that fulfill the clustering objective and captures the intuitive task of clustering more precisely than what purity does. However, when the number of clusters is large, it is very easy to fulfill the true negative condition, which may skew the index if the objective is to find similarities rather than dissimilarities. To avoid such situations, a relative importance α can be used to skew the index in a more appropriate way. If we also define FP (*false positive*) and FN (*false negative*), we can rewrite equation 2.7 as $\frac{TP+TN}{N(N-1)/2} = \frac{TP+TN}{TP+FP+FN+TN}$. The parameter α is introduced to lower the importance of elements being part of different clusters and makes the index emphasise when elements stay in the same cluster in both settings. The formula is

$$RandIndex_{\alpha} = \frac{TP + \alpha TN}{TP + FP + \alpha(FN + TN)} \tag{2.8}$$

With $\alpha = 0$, the index measures the number of pairs that were in the same clusters in both clusterings over the total number of pairs that were in the same cluster in the first clustering. This may be a more appropriate index for some occasions when similarity is more important than dissimilarities.

Mutual Information

In probability theory, *Entropy* $\mathbb{H}[X]$ is a measure of uncertainty of a random variable X. When X is a discrete variable defined on the interval $\{1, \cdots, K\}$, the entropy is defined as

$$\mathbb{H}[X] = -\sum_{k=1}^{K} p(X = k)\log_2\big(p(X = k)\big) \tag{2.9}$$

The concept of entropy can be applied to measure cluster purity by viewing a clustering as a discrete probability distribution. Define two arbitrary clusterings U and V. An interesting question is to ask how much we can know about V given that we have seen U? If U and V are independent, U doesn't give any information about V and if they are totally dependent, knowing U implies that V totally known. Such analysis is usually done with the correlation coefficient. A more general approach is to use *Mutual Information* $\mathbb{I}[U, V]$. Define $p_{U,V}(i, j) = \frac{|u_i \cap v_j|}{N}$, which is the probability that a random data point belongs to cluster i in clustering U and to cluster j in clustering V, $p_U(i) = \frac{|u_i|}{N}$ and $p_V(j) = \frac{|v_j|}{N}$. The mutual information is given by

$$\mathbb{I}[U, V] = \sum_{i=1}^{|C|}\sum_{j=1}^{|D|} p_{U,V}(i, j)\log\left(\frac{p_{U,V}(i, j)}{p_U(i)p_V(j)}\right) \tag{2.10}$$

Mutual information measures the reduction of uncertainty about one clustering given that the other clustering is known. If $\mathbb{I}[U, V] = 0$, no reduction of uncertainty is obtained, which only happens if $p_{U,V}(i, j) = p_U(i)p_V(j)$, i.e. U and V are independent. A positive value indicates that it is possible to say something about one clustering given the other. Negative values of $\mathbb{I}[U, V]$ is not possible [13]. Negative values would correspond to an increase of uncertainty when getting information about one clustering, which would be a rather strange event. The mutual information index lies between $0 \leq \mathbb{I}[U, V] \leq \min\Big(\mathbb{H}[U], \mathbb{H}[V]\Big)$. Thus, the normalised mutual information $NMI[U, V]$ can be used to measure cluster quality, where $NMI[U, V] = 1$ indicates a pure/stable clustering and $NMI[U, V] = 0$ indicates a total loss of similarity.

$$NMI[U, V] = \frac{\mathbb{I}[U, V]}{(\mathbb{H}[U] + \mathbb{H}[V])/2} \tag{2.11}$$

2.3 Gaussian Mixture Models in clustering

2.3.1 K-means and the EM-algorithm

The easiest and most common clustering algorithm is the *K-means* algorithm. The algorithm is initialised by randomly or intelligently placing K center points into the data space and is then iterating between two steps: 1) Assign every data point in the data set to one of the K center points, i.e. assign a cluster to every data point by choosing the closest center point 2) Relocate the K center points to be in the middle of the clusters corresponding to each center point. For many data sets, K-means is a fast and easy algorithm that performs well, but there are a lot of adversarial examples that occur in real life and that are easily manageable for the slightly more complex algorithm *EM-clustering* with Gaussian distributions.

The *Expectation Maximation algorithm* (EM) is a general framework that applies the iterative structure of K-means to a general distribution. The EM-algorithm is often used together with Gaussian distributions due to accurate results and explicit updating schemes [13]. One intuitive difference between the EM-algorithm and K-means is that K-means defines distance to a center point by drawing a circle around it while the EM-algorithm utilises general ellipses. Figure 2.1 shows a data set where K-means perform badly and the EM-algorithm captures what seem to be a very natural clustering of the data. The notion of distance is skewed by the EM-algorithm, weighting distances in different dimensions unequally. This often captures the underlying notion of what a cluster is. The K-means algorithm is actually a special case of the EM-algorithm in the Gaussian setting, where the covariance matrices are set to be equal to the identity matrix instead of being estimated and fitted to the data.

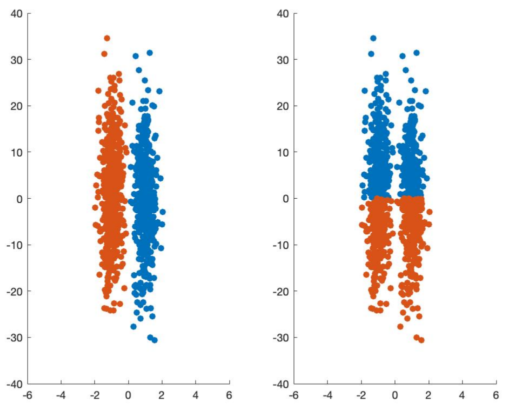

Figure 2.1: *An example where K-means (to the right) fails to capture the true clustering but where a Gaussian mixture model manages to assign the data to their true clusters.*

2.3.2 Gaussian Mixture Models

When the EM-algorithm is applied to a Gaussian framework, it is utilising what is called a *Gaussian Mixture Model* (GMM). A GMM consists of an arbitrary number of superpositioned Gaussians. In the setting of probability theory, the probability distribution of a GMM is captured by the following formula.

$$p(\mathbf{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\mathbf{x} \mid \mu_i, \Sigma_i) \tag{2.12}$$

$$\sum_{i=1}^{K} \phi_i = 1$$

$$1 > \phi_i > 0 \ \ \forall\, i$$

Here, K is the number of Gaussians that are used to fit the data set and and ϕ_i is the relative weight assigned to that Gaussian. To draw a random sample from this distribution is equivalent of first drawing a number from a distribution over $\{1, \ldots, K\}$ with probability distribution ϕ_i and then use this number to decide which of the K Gaussians defined by (μ_i, Σ_i) to draw a sample from. Instead of using a single Gaussian to fit an arbitrary data set, a GMM can be fitted with at least the same precision. However, finding the maximum likelihood fit of a GMM is not as straight forward as with a single Gaussian distribution. The EM-algorithm is a greedy way to find such an optima.

Advantages and disadvantages

Many applications demand that a data point is an exclusive member of a cluster, which is referred to as *hard clustering*. A GMM structure can perform so called *soft clustering* on the data set by assigning cluster likelihoods to each data point, i.e. the likelihood that data point i belongs to cluster k. This provides the user with information about the stability of every data point and can be used to e.g. predict changes in future cluster assignments. The likelihood of the data point can also be used to detect outliers. A point in the middle of the Gaussian distribution will have a much higher likelihood than a data point that exists far out from all cluster centers. It is easy to move from a soft cluster assignment to a hard cluster assignment by picking the cluster with highest probability (highest relative likelihood) for every data point. Moving from a hard cluster assignment to a soft assignment is not as trivial, which most other clustering algorithms (such as K-means) fail to do.

Besides being more flexible with cluster shapes, GMM can also provide the user with useful information about the number of clusters in the dataset through the model likelihood. Consider figure 2.2, where two GMMs with $K = 5$ and $K = 4$ are displayed. Eyeballing figure 2.2 suggests that a clustering with 5 clusters is appropriate.

The most prevalent disadvantage to using GMM comes from the same source as its advantages, namely the Gaussian ellipse. The covariance matrix Σ is estimated from samples and one data point is no longer enough to establish a cluster. In figure 2.2, each cluster contains 100 data points, which makes the covariance matrix behave well. When the number of data points in a cluster is less than the number of dimensions, the estimated Σ will not be invertible and no ellipse can be formed [13]. And even when there are a few more data points, outliers in the cluster may affect the sample covariance matrix to a large degree and very skewed ellipses might be the result. Clusters with few data points has to be assigned an independent covariance matrix or have a prior on the covariance matrix, which will affect the likelihood of the model and/or add an hyperparameter and additional assumptions to the clustering algorithm. The choice is thus between bad initialisation or additional assumptions, which are both disadvantages for the GMM.

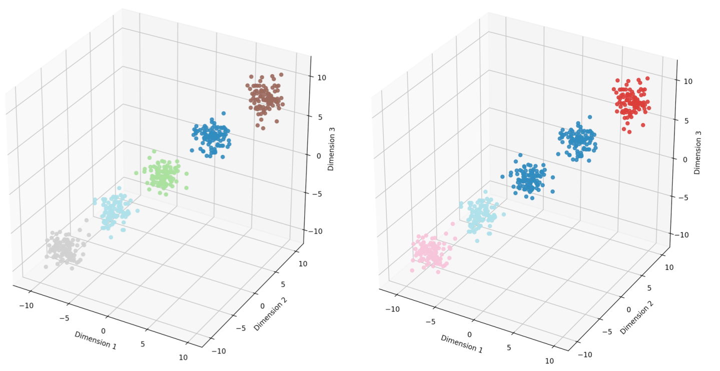

Figure 2.2: *Two clusterings on the same data set using $K = 5$ and $K = 4$ clusters. The likelihood of the first clustering with $K = 5$ is much larger than the likelihood of the second clustering with $K = 4$. For details, see section 5.1*

2.4 Hierarchical Clustering

Hierarchical clustering aims to order clusters into a natural hierarchy, where every cluster is a sub-cluster of a larger cluster (expect from the top cluster containing all data points). Hierarchical clustering has a natural way of dealing with the problem of K. Due to the hierarchical structure, the dataset X can easily be re-clustered for any K by dividing or merging clusters according to the hierarchy. The algorithm does not need a prespecified K, but it will not give any information about a suitable K.

Hierarchical clusters can be visualised using dendrograms, which is seen in Fig 2.3. The x-axis enumerates the data indices and the y-axis displays the distance between two clusters. The T-intersections symbolise a merge/partitioning of two clusters. To measure how well the dendrogram preserves the pairwise distances between the unclustered data points, the *cophenetic correlation coefficient* can be used [14]. If the cophenetic correlation coefficient is large, the y-axis of the dendrogram will be an accurate measure of distances between clusters and can thus be used to decide how many clusters there are in the data set. The cophenetic correlation coefficient c is defined as

$$c = \frac{\sum_{i<j} \big(d(x_i, x_j) - D\big)\big(t(x_i, x_j) - T\big)}{\sqrt{\big[\sum_{i<j} \big(d(x_i, x_j) - D\big)^2\big]\big[\sum_{i<j} \big(t(x_i, x_j) - T\big)^2\big]}}. \tag{2.13}$$

where

- $d(x_i, x_j)$ is the metric used
- D is the mean distance within the dataset
- $t(x_i, x_j)$ is the dendrogrammatic distance between x_i and x_j
- T is the average of $t(x_i, x_j)$.

A hierarchy provides two natural starting points for an algorithm, the bottom and the top. Algorithms that start at the bottom, i.e. $K = N$, are called *Agglomerative*, and algorithms that start at the top, i.e. $K = 1$ are called *Divisive*.

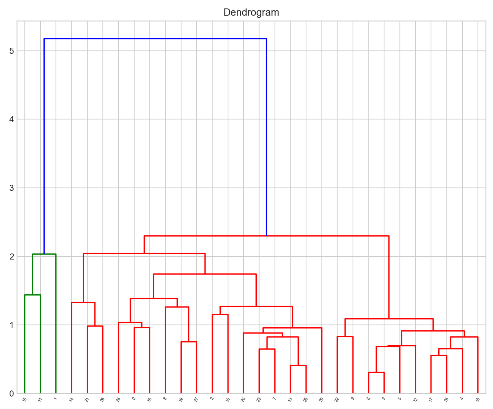

Figure 2.3: *An example of a dendrogram representing a hierarchical clustering. The red and green graphs constitute the first partitioning on this artificial dataset.*

2.4.1 Agglomerative Clustering

An agglomerative clustering algorithm sees all data points in X as clusters and aims to iteratively merge the two clusters that are closest to each other according to the distance metric d. Agglomerative clustering is the opposite of divisive clustering. Divisive hierarchical clustering considers the data set X as one big cluster and iteratively partitions the cluster where the split results in the greatest reduce of within cluster distance $W(C)$. This approach is repeated until every cluster is partitioned in a full hierarchy with the big cluster at the top and the N clusters at the bottom.

There are three main types of agglomerative clustering [13], which differ in how the dissimilarity between groups of points are defined.

Single link

Single link clusterings attempts to merge the two clusters that have the shortest distance between the closest members of each cluster. The distance between two clusters i and j is defined as.

$$d_{SL}(i,j) = \min_{[a \in i, b \in j]} d_{a,b}$$

Single link emphasise that close points must be in the same cluster. However, it may create large and unintuitive clusters stretching over the whole hyperspace and points that are seemingly unrelated become related due to a long chain of relations.

Complete link

Complete linkage takes the opposite approach and demands that the most dissimilar points in both clusters are closer to each other than any other pair of clusters to be merged. The distance between two clusters i and j is defined as

$$d_{CL}(i,j) = \max_{a \in i, b \in j} d_{a,b}$$

Complete linkage tends to create conservative clusters that don't venture the hyperspace.

Average link

Average linkage can be seen as an intermediate distance measure, measuring the distance between cluster centers.

$$d_{avg}(i,j) = \frac{1}{n_i n_j} \sum_{a \in i} \sum_{b \in j} d_{a,b}$$

2.4.2 GMM in hierarchical clustering

Traditional agglomerative hierarchical clustering algorithms are all based on a distance measure, iteratively merging the two nearest clusters according to the linkage criteria discussed above. Another approach is to use the GMM structure and merge the two clusters that maximises the total log likelihood of the data at every iteration. This gives an alternative merging criteria which utilises the likelihood framework and gives additional insight of when to stop the algorithm, i.e. gives insight on the problem of choosing K.

In the hierarchical setting, the disadvantage of needing several data points to estimate the covariance matrix is neatly solved by initiating the algorithm with a traditional agglomerative approach and switching to a GMM framework when the clusters starts to reach a considerable size. This approach solves the problem of assuming a covariance matrix and speeds up the algorithm due to the relatively fast computation of distances over estimating distributions and calculating likelihoods. This is described thoroughly in section 3.3.1.

2.4.3 Linkage matrix

An efficient way to store and express an hierarchical clustering is with the *linkage matrix*, for example used by MatLab, R and SciPy. If the data set contains n data points, the linkage matrix is an $(n-1) \times 3$ matrix. The first two columns indicate what clusters are merged and the third column contains the distances between the two clusters according to the chosen linkage. If the cluster number in column one or two is equal to or less than n, it indicates the merge of the data point with that index. If the cluster number is larger than n, it indicates that a cluster constituted by two or more data points is merged with another cluster. If the cluster number is $i > n$, this cluster is defined as the merge found at row $i - n$ in the linkage matrix. An example is seen in table 2.1

1	2	0.5	$\{1,2\},\{3\},\{4\}$
5	3	0.6	$\{1,2,3\},\{4\}$
6	4	1.2	$\{1,2,3,4\}$

Table 2.1: A hierarchical cluster structure with n = 4 data points represented by the linkage matrix. The two first columns show what cluster to merge. The third column shows the distance between the clusters and the last column shows the clustering. The linkage matrix consists of the first 3 columns.

2.5 Temporal Clustering

Clustering works on any numeric and/or categorical dataset X. Temporal clustering refers to the specific situation where X consists of time series, i.e. when $x \in X$ is an ordered sequence of data. On a basic level, temporal clustering does not differ from ordinary clustering. However, there are a few details that need to be regarded that are not present in most other types of data sets.

- The relation between adjacent time points in x may hold the feature of interest and the sequential structure of a time series needs to be exploited in a proper way. For example, a relatively fast *change* in the mean temperature of the atmosphere is revealed by the difference in temperature values rather than by the data values themselves. A financial crash on wall street is seen not on the values of the stocks but on the differences and movements on stock prices. Many clustering applications on time series wish to capture such changes rather than actual temporal values in the data.
- Time series within a data set may not be properly aligned or scaled, resulting in bad clustering results if traditional algorithms are used without keeping such relations in mind.

Such problems are solved by transforming the time series before feeding them to the clustering algorithm or by choosing an appropriate distance measure that should be invariant to a number of transformations. A few such transformations are listed here [6].

Scaling and translation invariance

A distance measure should not be confused by a linear transformation of the data, i.e. if the time series $x_{\{t\}}$ or $y_{\{t\}} = ax_{\{t\}} + b$ is used should not matter for the resulting distance to another point. For example, a proper and useful distance measure should regard a time series of temperature measurements in degrees Celsius as very close to a times series in Farhenheit over the same area.

Shift invariance

A set of time series often contain shifted series and dependencies, meaning that one time series $x_{\{t\}}$ is a lagged version of another time series $y_{\{t\}}$, i.e. $x_{\{t\}} \approx y_{\{t-\tau\}}$. One example is that two time series measuring todays temperature should not be considered dissimilar if the only difference is that one time series start at 8 am and the other time series starts at 11 am. A proper distance measure should imitate a phase shift of three hours to align the two series.

Uniform scaling invariance

Sequences of different length require stretching or shrinking in order to be properly aligned. That is, a temperature measurement series sampled every minute should be close to a measurement series sampled every hour.

Complexity invariance

Time series are often prone to noise and it is important for many applications that a distance measure disregards noise or other complex superpositions. For example, an audio signal recorded outdoor may be considered to be similar to the same audio recorded indoors, despite the extra noise added from recording outdoors.

2.5.1 Time series distance measures

In general, problems regarding transformations should be solved before measuring distances. However, that may not be so easy to do in advance, which is why a it is favourable if a distance measure possesses such qualities. It may be the case that a different transformation is needed for every pair of time series compared, disabling the use of a general transformation. Here are a few popular distance measures for time series, satisfying the set of invariances to different degrees. [6].

Euclidian Distance

Simple euclidian distance between to time series may be used with good results if combined with standard transformations to set the series on the same amplitude, translation, phase and length.

$$ED(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \tag{2.14}$$

Correlation

Correlation may be used to measure the distance between two time series. Correlation can be related and understood as the angle between the vector $x_{\{t\}}$ and $y_{\{t\}}$. If two time series are perfectly correlated, i.e. $\rho(x_{\{t\}}, y_{\{t\}}) = 1$, the angle between their vectors are equal to 0 degrees and they should be regarded close to each other. Two kinds of distance measures are possible to form using correlation.

$$D_{\rho,1} = 2(1 - \rho(x_{\{t\}}, y_{\{t\}})) \tag{2.15}$$

$$D_{\rho,2} = 2(1 - |\rho(x_{\{t\}}, y_{\{t\}})|) \tag{2.16}$$

The second distance measure is useful for example in a financial setting, where perfect anti-correlation indicate that a portfolio manager should go short in the other asset.

Dynamic Time Warping

Dynamic Time Warping (DTW) is a distance measure which allows for local phase shifting [6]. Let the length of the time series be m and let M be an $m \times m$ matrix, where $M(i, j) = |x_i - y_j|$. Also, let the Warping path $W = \{w_1, w_2, \dots, w_k\}$ with $k \geq m$ be a contiguous set of matrix elements of M. The DTW distance is then defined as

$$DTW(\vec{x}, \vec{y}) = \min \sqrt{\sum_{i=1}^{k} w_i} \tag{2.17}$$

DTW can be computed using dynamic programming minimising subpaths through the matrix. The referred article [6] has an intuitive and understandable illustration of what DTW does. DTW can easily capture phase shifts in the data and may also capture important properties such as climaxes of different lengths. DTW would probably capture the similarity of one time series measuring the sound level in dB at an arena during a concert day and one time series measuring the number of people at the arena. There is a climax of two hours of high sound levels and a climax of five or six hours of number of peoples in the arena. DTW would link those two climaxes very well to each other, despite their different durations.

2.5.2 Deep Temporal Clustering

Deep Temporal Clustering is a novel method for classification of time series using autoencoders and KL-divergence to establish clusters, which in turn are used to classify a time series [12]. The autoencoder is used to reduce the dimensionality and to extract characteristic and defining features from the time series. Other methods of feature extraction may weigh uninteresting or noisy features relatively high and thus classify the time series on suboptimal information. This is avoided by clever implementation of the autoencoder, described thoroughly in the article [12]. Through the autoencoder, a time series of length m is reduced to a point in a d-dimensional space. Parallell with this dimensionality reduction, the algorithm fits the d-dimensional points to cluster centroids using KL-divergence. This encourages the formation of clusters which then are used to classify previously unseen entries by calculating the similarity or distance to the clusters. The authors presents competitive results on open time series data sets.

2.6 Artificial Neural Networks

Artificial Neural Networks (ANN) constitutes a paradigm within the field of machine learning. Despite being defined as early as 1943 [2], ANNs were not widely applied until recent years when G. Hinton and R. Salakhutdinov wrote their famous article *Reducing the Dimensionality of Data with Neural Networks* [15], where they used an ANN to push the out of sample predictive boundary on the ubiquitous MNIST hand written digit data set beyond its former limit. Artificial neural networks is a huge topic and will only be covered briefly in this thesis. The next sections will describe the layer architecture, the activation function and back propagation.

2.6.1 Layers

Layers are the building blocks of an artificial neural network. There are several types of layers that can be utilised to make an artificial neural network perform better at a certain task.

Dense layers

A dense layer is the most basic of network layers. Every node in the first layer is connected to every node in the second layer. Thus, the value in one node can affect every other value in the next layer, laying the groundwork for complex structures to be learnable. Although being very powerful and versatile, they are often too big, general and slow to train for many applications. Consider a layer with 100 nodes and a second layer with 50 nodes. The total number of connection is then $100 \cdot 50 = 5000$.

Convolutional layers

Convolutional layers are filters that detect certain features in the data. They are usually used in image processing and time series analysis to detect e.g. facial features or specific events. For example, there can be one convolutional layer that is specialised on finding puppy ears in a picture. This filter is applied to every group of pixels to measure the degree of puppy ears prevalent in the pixel group. Convolutional layers uses way less connections and are often better than dense layer at common tasks such as feature extraction, partly due to being trained to find specific features and partly due to the lack of noise from irrelevant nodes (the existence of puppy ears in the upper left corners of a picture is in most cases independent of what happens in the lower right corner).

Locally connected layers

A locally connected layer is similar to convolutional layers, but they differ in that locally connected layers apply new filters at every group of nodes whereas convolutional layers apply the same filter to all groups of nodes. Instead of measuring the degree of puppy ears in a specific square of a picture, locally connected layers are measuring different things at every new square of the picture. This can be useful when analysing e.g. stocks, where a locally connected layer can measure "how much was this stock affected by the crash of Lehman Brothers in 2008" when hovering the historical data of year 2008. A convolutional layer would ask that same question when looking at data of year 2007, which of course would not yield any useful information. A locally connected layer can also be understood as a dense layer constrained to only be connected to the nearest nodes in the next layer. Figure 2.4 portrays a locally connected network.

2.6.2 Activation Functions

The value of a node $z_{i,j}$ in layer i and position j depends on 1) the value of the nodes $z_{i-1,\ell}$ along the connections to the previous layer 2) the weight value $w_{i-1,\ell}$ on the connection 3) the *activation function* ϕ_i. This is described by

$$z_{i,j} = \phi_i\Big(\sum_{\ell=1}^{L} z_{i-1,\ell} w_{i-1,\ell}\Big) \tag{2.18}$$

Activation functions have been observed in real neural networks (e.g. in brains) [2] and are useful to capture non-linear structures in the data. A few common activation functions are listed here.

Identity function

$$\phi(x) = x$$

ReLu

$$\phi(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

ELU

$$\phi(\alpha, x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{else} \end{cases}$$

Sigmoid

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

2.6.3 Back propagation

Given an input x, the ANN produces an output $f(x)$ by feeding x forward through the hidden layers of the network. This process goes under the name *forward propagation.* In order to learn interesting and useful features, an ANN has to change its weights such that the input instance x is correctly mapped onto y, the true labels. The weights are randomly initialised and the probability that x is anywhere close to being correctly mapped to y is infinitesimal. In order to make a better prediction next time, the cost function $J\big(y, f(x)\big)$ would have to be fed backwards through the hidden layers and nudge the weights in the direction which would minimise the prediction error if the same sample was fed through the network once again. This nudging process is called *back propagation.*

In theory, back propagation is not much more than the derivative chain rule of the cost function $J\big(y, f(x)\big)$ applied to every single weight within the network. This means that at every weight, the magnitude of contribution to the error is calculated and the weight is nudged accordingly. If this particular weight is highly responsible for the error, it gets changed more than a weight that is not as responsible for the error. The notion of "responsibleness" is measured by the chain rule derivative. The influence of the ℓ :th node in the previous layer on the j :th node in the current layer is specified by the chain rule. Let $Y_\ell = w_{i-1,\ell} z_{i-1,\ell}$ and $z_{i,j} = \phi_i(\sum Y_\ell)$. Then,

$$\frac{\partial z_{i,j}}{\partial z_{i-1,\ell}} = \frac{\partial \phi_i(Y_\ell)}{\partial Y_\ell} \frac{\partial Y_\ell}{\partial z_{i-1,\ell}} \tag{2.19}$$

2.7 Autoencoders

An *autoencoder* is a neural network that tries to map the input instance x onto x instead of onto y. This means that the autoencoder strives to produce an output that is as similar as possible to the input. Furthermore, the autoencoder has a hidden layer, which frequently is referred to as the *latent space*. The first part of the autoencoder is referred to as the *encoder*, because it is taking the input and encodes it onto a latent code z. The output of the the encoder is referred to as the latent representation of the input and is denoted z_x. The second half is called the *decoder* since it takes the latent representation z_x and decodes it into x. The name autoencoder arises from the fact that the network encodes and decodes its own inputs. The structure can be be understood as two consecutive functions $f(x)$ and $g(z_x)$. The function $f(x)$ takes the input and maps it onto the latent representation z_x, and the function $g(z_x)$ and reconstructs x from the latent representation z_x. In this setting, $f(x)$ corresponds to the encoder and $g(z_x)$ corresponds to the decoder. The autoencoder can then be expressed as $g\big(f(x)\big)$. If the dimensionality of the hidden layer d_{hidden} is lower than the dimensionality of the input layer d_{input}, the autoencoder is said to be an *undercomplete autoencoder*. The whole structure of an autoencoder is seen in Fig 2.4, where $d_{hidden} = 3$, and $d_{input} = 5$.

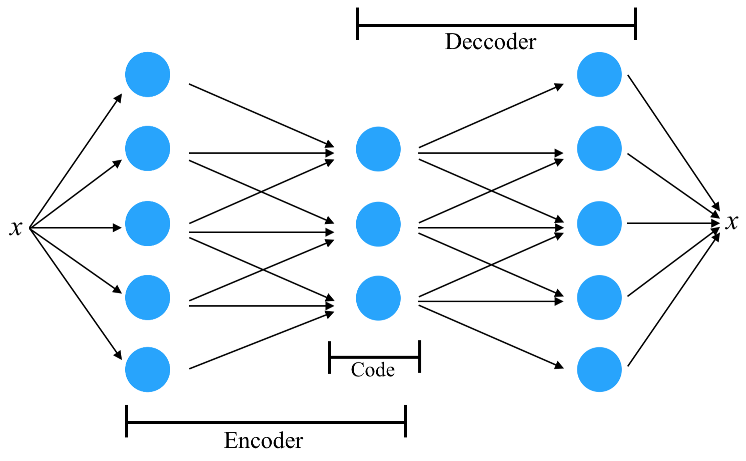

Figure 2.4: *A locally connected, undercomplete Autoencoder with a 5 dimensional input space and a 3 dimensional hidden layer. The arrows are called weights*

A normal feed forward neural network is motivated by its predictive ability, i.e. a well trained network will produce meaningful outputs. This is not the case for autoencoders. In most cases, the interesting part is what exists in the hidden layer, namely the latent representation z_x. If a 5-dimensional input x is fed through the network in Fig 2.4, it will first be projected down on a 3-dimensional space and then reconstructed in its 5-dimensional form. If the reconstruction is good and the network was properly trained, the network was able to extract a 3-dimensional representation of x (i.e. z_x) and reconstruct x from z_x without a significant loss. This means that the network has learned to identify key features within the data. An intuitive way of understanding this process is to imagine a 3d printer being able to reconstruct the body of a bird by only looking at its shadow. The autoencoder has learned to do this by changing its weights and the information dense latent representation z_x exists in the hidden layer, the *latent space*.

2.7.1 Latent Space

The latent space can be seen as the output of a dimensionality reduction technique. The big advantage of autoencoders over linear techniques like PCA is that non-linear activation functions can be applied and a non-linear dimensionality reduction is obtained. Thus, many patterns and features that PCA cannot capture may easily be learned by an autoencoder with non-linear activation functions.

Artificial Neural Networks usually have many more parameters to tune than the dimensionality of the input data. Thus, they have large possibilities to give very different outputs as a result of a minor change in the input and are thus prone severe overfitting. This is especially true for the latent space. If the autoencoder is set to only learn to reconstruct the input data, it can do so perfectly by mapping every input data to a specific point in the latent space and learn to decode it only by using the latent space representation as a memory marker. This corresponds to someone learning to pronounce words one by one rather than learning phonetic rules. If you have a very powerful memory, that may be an easier approach than learning complicated and generalisable rules.

Ideally, the latent space should be restricted so that it contains useful information in the *general* case rather than the *specific* case. That means that a dimension in the latent space could correspond to a specific feature or that small changes in latent space representation z corresponds to small changes in the output. Such restrictions can be incorporated into the loss function to ensure that the latent space behave non-chaotic. This is discussed in detail section 3.1.3.

2.7.2 Variational Autoencoder

Variational Autoencoders (VAE) have gained a lot of attention since they were introduced in 2014 [2]. A VAE differs from a regular autoencoder in two ways: 1) It has an extra term added to its loss function, namely the *KL-divergence*. 2) it introduces noise to the latent space, making the latent representation slightly different every time the input is fed through the network. This has many consequences and opens the autoencoder framework to interesting tasks.

- When noise is introduced into the latent space, the autoencoder is forced to make the latent space quasi-continuous. If the autoencoder does not know where the latent representation z will be, it has to make every point within a few standard deviations of $\mathbb{E}[z]$ produce somewhat accurate reconstruction results. This can be used to ensure that points that are located close to each other in the latent space actually look quite similar in the original data.

- If all points within a few standard deviations from $\mathbb{E}[z]$ give a somewhat accurate result when fed through the decoder, these points can be used to create new instances of the original data. For example, a video game designer may create a tree for a video game and then train an VAE to create similar but not identical trees. This feature makes the network a *generative* network, since it can generate new samples.

- A VAE makes it easy to generate arbitrary points in latent space. These points can in a second step be forced to look like a specific object when decoded. This opens the autoencoder to be pre trained in a very specific manner, enabling very specific tasks to be preformed.

The KL-divergence, described in more detail below, forces the latent representation to follow a certain distribution, usually a Gaussian distribution. When generating samples from a normal distribution and feeding them through the decoder, the KL-divergence has ensured that the samples aren't way off from any earlier points. The KL-divergence is not necessary to include but is usually included to make the network a well functioning generative network. However, the name variational autoencoder stems from the variational middle layer producing the noise.

2.8 Error Measurement and Loss Functions

Many applications of statistical theory boils down to an optimisation problem, which have been seen in the general clustering problem, equation 2.2, and in the training of an ANN. A generic problem is to fit the best possible model to a given situation or given data set under some constrains. This is done by 1) formulating a quantitative measurement of how good (or bad) the model fit is 2) find the maximum (or minimum) of this measurement. Such functions are usually formulated as *loss functions*, which measures the residuals of the model or how bad a model fits the data. There are plenty of different loss function with different meaning and area of use. They often try to capture some intuitive notion of goodness of fit. The choice of loss function has a huge impact on the final model, which motivates careful customisation in order to make the model portray the data as well and as general as possible. Several loss functions can be weighted together using time dependent weights in order to obtain and capture complex behaviour.

2.8.1 Mean Square Error

The by far most common and popular loss function is the *mean square error*, MSE. The MSE of a model $f(x)$ is defined as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 \tag{2.20}$$

where, y is the set of true values. Its popularity stems from its convenient mathematical properties, such as being convex, differentiable and yielding analytical solutions. It occurs in the normal distribution and has close relation to the Pythagorean theorem and the second moment of a probability distribution. It is often used as loss function either by itself, e.g in linear regression, or together with extensions, as in ridge regression or Lasso regression. However, it is sensitive to outliers due to its quadratic structure.

2.8.2 Mean Absolute Error

The *mean absolute error*, (MAE) is sometimes used as a complement to the mean square error.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)| \tag{2.21}$$

The MAE can be used to deal with problems of outlier influence, since it is less sensitive to outliers. However, it does not have a continuous derivative, which may cause problems in some optimisation tasks.

2.8.3 Hubert loss

It is possible to get the best from both MSE and MAE by using Hubert loss.

$$\text{Hubert loss}_\delta = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| < delta \\ \delta(|x| - \frac{1}{2}\delta), & \text{otherwise} \end{cases} \tag{2.22}$$

Hubert loss is equal to MSE for $|x| < \delta$ and to MAE for $|x| > \delta$. The penalty for this combination is that the threshold value δ has to be chosen manually.

2.8.4 Kullback-Leibler Divergence

A more sophisticated and less intuitive loss function is the *Kullback-Leibler Divergence*, (KL-divergence) which measures how much one probability distribution differs from another one. It is used in information theory to measure how many more bits that are expected to be used to send a message using a suboptimal encoding over a more optimal encoding.

Let P and Q be continuous distributions. The KL-divergence $D_{KL}(P||Q)$ is defined as

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)\log\Big(\frac{p(x)}{q(x)}\Big)dx \tag{2.23}$$

If P and Q are equal for every x, the KL-divergence will be 0. This is easily seen since $\log\Big(\frac{p(x)}{q(x)}\Big) = 0$ for all x. High values of the KL-divergence indicates dissimilarity between P and Q. KL-divergence can be used to make a model behave as a target distribution, which is the way variational autoencoders uses KL-divergence described above.

2.9 Test specific theory

2.9.1 Measure distribution divergence

In section 4.2.2, distributions will be estimated using Gaussian kernels. Let z be the return series of a fund over a time span τ. Also, let z be a part of a cluster $C_{z,\tau}$. Now, define $p_{z,\tau}(x)$ to be the continues estimation of the distribution of z obtained with a Gaussian kernel [8], defined by equation 2.24. All new parameters are defined below.

$$p_{z,\tau}(x) = \frac{\sum_{t\in\tau}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-z_t)^2}{2\sigma^2}}}{\int_{-\infty}^{\infty}\Big(\sum_{t\in\tau}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y-z_t)^2}{2\sigma^2}}dy\Big)} \tag{2.24}$$

Now, define $p_{C_{z,\tau}}(x,\beta)$ to be a continuous estimation of the distribution of z using a Gaussian kernel and data from other cluster member in $C_{z,\tau}$, defined by function 2.25.

$$p_{C_{z,\tau}}(x,\beta) = \frac{f(x,\beta)}{\int_{-\infty}^{\infty}f(a,\beta)da} \tag{2.25}$$

$$f(x,\beta) = \frac{\beta}{N}\sum_{[y\in C_{z,\tau},y\neq z]}\sum_{t\in\tau}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-y_t)^2}{2\sigma^2}} + (1-\beta)\sum_{t\in\tau}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-z_t)^2}{2\sigma^2}}$$

- y is a fund in cluster $C_{z,\tau}$ not equal to z.
- $\sigma > 0$ is called the bandwidth in the literature and corresponds to the smoothness of the estimation. A large σ will make $p(x)$ a smooth bell curve and a small σ will make $p(x)$ sprawling. See figure 4.6 for an illustration.
- $\beta \in [0,1]$ weights how much the other cluster member should influence $p_{C_{z,\tau}}(x,\beta)$. Intuitively, $p_{C_{z,\tau}}(x,\beta)$ can be understood as a smooth estimation of the distribution of z, where β is the amount of influence from samples from the other cluster members and $(1-\beta)$ is the influence from samples from z. If $\beta = 1$, only other cluster members are used to estimate $p_{C_{z,\tau}}(x,1)$. If $\beta = 0$, only samples from z is used to estimate its distribution.

The estimated distributions are used to predict the future distribution of z, i.e. $p_{z,\hat{\tau}}(x)$, where $\hat{\tau}$ is the consecutive time period. This is done with the *mean integrated square error* (MISE) and the *KL-divergence.*

$$\text{MISE}_{[f(x),g(x)]} = \int_{-\infty}^{\infty}\big(f_(x) - g(x)\big)^2dx \tag{2.26}$$

$$\text{KL-divergence}_{[f(x),g(x)]} = \int_{-\infty}^{\infty}f(x)\log\big(\frac{f(x)}{g(x)}\big)dx \tag{2.27}$$

These integrals are estimated by the following expressions, $f[x]$ and $g[x]$ are equally spaced, vectorised versions of $f(x)$ and $g(x)$ defined on a finite interval of choice.

$$MISE_{f[x],g[x]} = \frac{1}{n}\sum_{i=1}^{n}\big(f[x_i] - g[x_i]\big)^2$$

$$KL_{f[x],g[x]} = \frac{1}{n}\sum_{i=1}^{n} f[x_i] \cdot \log\Big(\frac{f[x_i]}{g[x_i]}\Big)$$

2.9.2 Utility Theory

Utility theory is a framework that quantitatively captures concepts such as *risk averseness*, described thoroughly here [16]. For example, an individual who is not the least risk averse would never buy insurance. The reason is that insurance is almost by definition a loss affair in terms of expected value. However, most people buy insurance and according to utility theory the reason is that an insurance has higher expected utility than the money spent on the insurance has.

A risk averse person has a concave utility function, effectively saying that a large loss is much worse than a large gain is good. Instead of optimising a portfolio solely on the value, a portfolio can by the analogy above be optimised over the utility function of the value. For the tests conducted in chapter 4, the following utility function is used.

$$u(x, r) = \frac{x^{1-r}}{1-r} \tag{2.28}$$

where x is the value and $r > 1$ is the level of risk aversion.

Chapter 3

Model Building

With the theory covered in chapter 2 as basis, this chapters gives a thorough walkthrough of how to implement the ideas from chapter 1 and to derive the results in chapter 4.

As stated in chapter 1, the goal of this thesis is to engineer an algorithm that is fed with a set of time series and outputs either a cluster assignment for every time series or gives an hierarchical clustering structure together with information and statistics of likely number of clusters in the data set as well as additional information for further analysis. The algorithm is constituted by 2 major parts.

- **A variational autoencoder** (VAE). The VAE is used to reduce the number of dimensions in the data set and to replace the need for a complex distance measure. It also provides information about the affinity of different time series.
- **An agglomerative hierarchical Gaussian mixture model**. This algorithm takes the coordinates given by the VAE and fits the points to Gaussian distributions in a hierarchical fashion, merging the two Gaussians that results in the most likely merge.

Both parts and how they communicate are described in the next two sections. Throughout the text, this algorithm will be referred to as **the thesis algorithm**, or simply **the algorithm** if the context is unambiguous.

3.1 Implementing a variational autoencoder

The overall goal of the variational autoencoder is to extract information dense representations of the input data and provide this information to the agglomerative hierarchical clustering algorithm. Consider figure 3.1, the input x is fed to the input layer and transported through the network. The VAE weights (i.e. the arrows) are trained so that the output resembles the input as closely as possible. If total resemblance is obtained, the latent representation z_x in the middle (i.e. the three green circles in the middle) contain all information necessary to reconstruct x. This latent representation z_x is what is used in the clustering algorithm. In order to ensure reliable results, a few constrains and properties has to be put on the autoencoder, which will be discussed in the following pages.

3.1.1 The latent space - a chaotic environment

The term *latent space* refers to the space in which the encoded input z_x exists. In the case of figure 3.1, the latent space is a 3-dimensional space. Every green circle corresponds to a dimension and when an input x is fed through the network, the value of the green circles gives the coordinates to each dimension in the latent space respectively. Each input x will thus be represented by a 3-dimensional point and can be fed to the cluster algorithm. See figure 2.2 for an illustration. Applying a clustering algorithm to points in the latent space is straight forward. To ensure that the clusters in the latent space capture important features in the input space

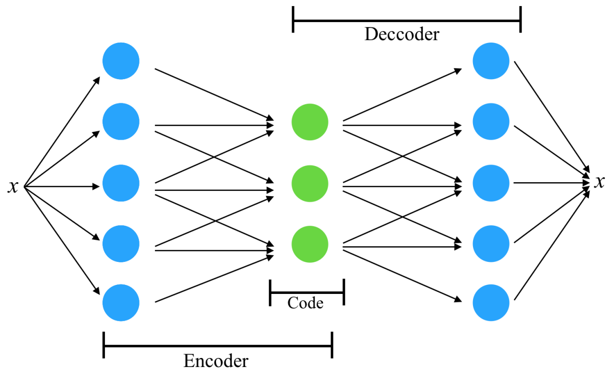

Figure 3.1: *Input x is fed to the left column of blue nodes, transformed into a latent representation z_x in the latent space (green nodes), and transformed back to the output layer (the right column of blue nodes). The arrows are called weights*

(i.e does a cluster $\{z_x\}$ represent a meaningful cluster $\{x\}$?), the two clustering properties mentioned in section 2.1 need to be ensured.

- **Property 1**: Two points that are close to each other in the latent space should be close to each other in the input space. This corresponds to the notion that two dissimilar objects x_i and x_j should not end up in the same cluster in latent space.
- **Property 2**: Two points that are close to each other in the input space should be close to each other in the latent space. This corresponds to the notion that two similar objects x_i and x_j should end up in the same cluster in latent space.

Due to the overfitting tendencies of autoencoders, these two properties are not trivially ensured. There is nothing that prevents two neighbouring points in the latent space to be mapped to completely different outputs, since a regular autoencoder is only trained to replicate the input as well as possible and not to care about the latent representation z_x. Two similar but not identical objects x_i and x_j are not guaranteed to be close to each other in latent space. To ensure meaningful clusters, something more has to be done.

Enforcing a non-chaotic latent space

Ironically, the chaotic nature of the latent space can be controlled with another chaotic phenomenon, *noise*. By introducing noise to the latent representation z_x, the autoencoder is forced to not put dissimilar inputs next to each other in the latent space and overfitting is avoided. Intuitively, when noise is introduced, an input x is mapped not only to a single point z_x, but rather to a space covering z_x. The autoencoder does not know the exact location of z_x and must therefore reserve a space around the most likely location $\mathbb{E}[z_x]$ from which all points map to something similar to x. This ensures that all points within the noise range are interchangeable. Thus, the first property is ensured.

The noise does not necessarily ensure the second property: that two similar points in input space end up close to each other in the latent space. The latent space is only forced to cover *some* types of similarities, but not *all* types of similarities. There is nothing that prevents one type of similarity to not be modelled at all, opening the possibility to the latent space representations to be very far from each other despite being similar in some sense.

This problem is solved neatly by the training process. At the initial steps of training, all weights are randomly initialised, which tends to map all inputs down to almost the same latent space representation around the origin. If the initialised noise standard deviation is set large enough, the first property has not yet been ensured and dissimilar inputs may end up very close to each other. For every optimisation iteration the network does, the network pushes inputs that are dissimilar away from each other in pace with the establishment of the first property and to ensure accurate reconstruction. Especially, it separates highly dissimilar objects with a higher probability than it separates more similar objects, i.e. the autoencoder is motivated to separate dissimilar object rather than separating similar objects. Although not being as strong an argument as the argument for the first property, it seems to be sufficiently strong in practice. This is especially true if the number of potential latent space representations is constrained so that there is just not enough space for two similar points to occupy two different subspaces in the latent space. This space constraint can be controlled by hyperparameters on the loss function, described below. Thus, the second property will be ensured.

3.1.2 Layers

The structure of the autoencoder has a huge impact on the performance. The autoencoder depicted in figure 3.1 has very few weights (18 arrows inside the system) and can only take a 5-dimensional input. If the data only has 5 dimensions, this is of course fine, but the 18 tuneable weights can capture only a very limited amount of complexity in the data. To ensure that the autoencoder performs properly, it needs a sufficient number of nodes and weights.

Section 2.6.1 describes a few common types of layers. When constructing an autoencoder for time series, convolutional or locally connected layers seem preferable. In many applications, especially for stock price analysis, there is not really any need for analysing dependencies between time points far away from each other. A more efficient use of computing resources is to spend most of it to find structures in adjacent time points, which are more likely to contain useful characteristics. This is exactly what locally connected layers are designed to do, which is a strong argument to use them. This is essentially the same argument to why convolutional layers are used in image recognition [17].

Although being favourable in theory, locally connected layers were slower in practice. When implemented for this thesis, a locally connected layer iterated slower than a dense layer. Speed is the main motivation for using locally connected layers over dense layers. Due to the lack of improvement in speed, dense layers are used when the thesis algorithm is implemented for testing in chapter 4, despite theoretical arguments.

A larger network, i.e. more nodes and weights, will perform better and be more agile, but be slower to train. An optimal size is hard to find for a general task, but can easily be changed and tailored for specific data sets.

Variational middle layer

To introduce noise into the latent space, a variational middle layer is needed. This layer is constituted by two parts 1) the expected latent representation z_x, $\mathbb{E}[z_x]$, 2) the noise distribution of z_x. Multivariate, uncorrelated Gaussian noise is used and the variational middle layer is used to determine standard deviation along each dimension. This is done by letting the optimisation algorithm decide the standard deviation for each dimension, optimising the parameters to ensure an end result compatible with the loss function. Without the variational middle layer, the input would be mapped to a specific point z. With the variational middle layer, z is replaced with $\mathbb{E}[z]$ and a noise of appropriate size is added.

3.1.3 Loss function

The loss function is the welding force of the autoencoder. Desired results can be obtained by customising the loss function. In the case of a variational autoencoder, the suggested loss function [2] is the mean square error over the output plus the KL-divergence of the latent space. Especially, when the latent representation is fitted to a Gaussian distribution, this loss function takes the following form [2].

$$\text{Loss function} = \sum_{i=1}^{n} \big(x_i - g\big(f(x_i)\big)\big)^2 + \sum_{i=1}^{n}\sum_{j=1}^{d} \frac{z_{i,j}^2 + e^{\gamma_{i,j}} - 1 - \gamma_{i,j}}{2} \tag{3.1}$$

where

- x_i is the input.

- $g\big(f(x_i)\big)$ is the reconstruction of x_i.
- n is the number of data points.
- d is the dimensionality of the latent space.
- z_{x_i} is the latent representation of x_i.
- $\gamma_{i,j}$ is a transformation of the standard deviation from the variational middle layer. $\sigma_{i,j} = e^{\frac{\gamma_{i,j}}{2}} \implies \gamma_{i,j} = \log(\sigma_{i,j}^2)$.

For the purposes of obtaining good clustering qualities, this thesis proposes the following alteration of the loss function.

$$\text{Loss function} = \sum_{i=1}^{n} \big(x_i - g\big(f(x_i)\big)\big)^2 + \beta \sum_{i=1}^{n}\sum_{j=1}^{d} \frac{\alpha z_{i,j}^2 + e^{\gamma_{i,j}} - 1 - \nu\gamma_{i,j}}{2} \tag{3.2}$$

The motivation behind this loss function is that the added hyperparameters β, α and ν are attached to important features of the loss function.

- β: The relation between the mean square error and the KL-divergence is important to determine how much the autoencoder should learn to reconstruct the input data versus how much attention the variational middle layer should be given influence. This parameter is introduced here [18]
- α: The hyperparameter α is connected to the term z_i^2. Minimising z_i^2 is equivalent of putting a penalty on z_i far away from the origin. This means that the latent space representations z_x are encouraged to stay within a confined space with soft borders. A high α puts extra effort on constraining this confined space and a low α allows z_x to spread over a larger space.
- ν: This parameter is connected to γ, which is the logarithm of the variance used by the variational layer. When the variance is small, i.e. the point move very little in the latent space, γ will be take a large negative value. Thus, a large ν will penalise small variances and z_x is forced to cover a larger area.

If the balance between α, β and ν is right, i.e. if the balance between total available space, individual space requirements and the reconstruction precision is right, **clusters are forced to form**. Similar objects have to end up close to each other, because all other options are too costly. Dissimilar objects have to be separate, because the reconstruction loss make it too costly to put them close to each other. Thus, minimising the loss function 3.2 leads to a meaningful clustering of the data.

3.1.4 Output

As indicated earlier, the reconstruction of the input x is the formal output of the autoencoder but is not the only thing of interest. Instead, four different outputs are generated and passed on to the agglomerative hierarchical GMM.

1. **The reconstructed input**. Although not needed in order to cluster the data, it is important to doublecheck that the reconstruction looks fairly similar to the input. Property 1 and 2 discussed above are only valid if the reconstructions are good, since similar and dissimilar objects are measured on how similar or dissimilar their reconstructions are, not the actual input. The clustering will not be better than the reconstructions allow it to be.
2. **The expected latent representation**: $\mathbb{E}[z]$. The point $\mathbb{E}[z]$ is the latent representation which most accurately will reconstruct the input. Thus, these are the latent representations that will be used for clustering. Ideally, these points should form easily separable clusters.

3. **The covariance matrix of z**: $\text{Cov}[z]$. The covariance matrix of z, defined by $\gamma_{i,j}$, can be used to cap the size of clusters. All points within a few standard deviations of $\mathbb{E}[z]$ are arguably similar to z and should be clustered together. Points that are many standard deviations away from $\mathbb{E}[z]$ are less likely to be similar to z, and should thus not be clustered together. $\text{Cov}[z]$ can also be used as a prior for the GMM.

4. **A sample of the latent space**: $\mathbb{E}[z] + \text{Cov}[z]^{\frac{1}{2}}\epsilon$. Here, ϵ is standard Gaussian noise. This output is used to check how large portion of the space that is used. Ideally, this output should scatter the points all over the space, indicating that the space is somewhat continuous.

The output is used to verify that the end results are reliable and to feed the clustering algorithm with data.

3.2 Implementing an agglomerative hierarchical Gaussian mixture model

When the input data has been reduced in dimension and is represented by the latent representation z, the clustering algorithm can start to work. Instead of merging based on a distance between two clusters, the standard deviations given by the VAE are used to fit Gaussian distributions over the clusters and merge clusters that result in the largest log likelihood. This provide the user with information about the likely number of clusters in the data as well as allowing more complex cluster geometries to form.

3.2.1 Fitting a distribution to the data

Merging based on likelihood of Gaussian distributions differs from distance based agglomerative clustering in one important aspect, namely the need to estimate a distribution to fit the data. A good and reliable estimation requires more than a few points. Starting an agglomerative algorithm implies that every cluster is a single point, which makes a good estimation infeasible. There are several ways to approach this problem.

- A first approach is to use the standard deviations $\sigma_{i,j}$ that the VAE sets for every point, as described in section 3.1.4. Such initialisation is motivated by the fact that a point is unlikely to be a part of a cluster with a covariance matrix exceeding this initialisation by far. Thus, this initialisation should be seen as an upper bound on the size of the covariance matrix.

- Another approach is to use the standard deviations from the VAE as a prior and update the covariance matrix as more data i gathered. This opens up for a Bayesian approach to this problem, which probably is advantageous but beyond the scope of this thesis. A simple updating scheme would be

$$\Sigma = \frac{C\bar{ov}[z] + \Sigma_{est}[z_i](n_z - 1)}{n_z}$$

  where $C\bar{ov}[z]$ is the mean of prior standard deviations in the cluster, $\Sigma_{est}[z_i]$ is the estimated covariance matrix of the cluster and n_z is the number of data points in the cluster.

- A third approach is to use two extra hyperparameters. The first hyperparameter determines the values along the diagonal of the covariance matrix and the second hyperparameter decides how many points a cluster needs to have before estimating a covariance matrix. This number has a lower bound dependent on the dimensionality of the data space. If there are too few points (fewer than then the dimension of the latent space), the covariance matrix will be singular [13]. This is the approach that will be used in chapter 4.

The mean of the distribution can be set to the sample mean. It is also possible to use a centroid approach, demanding that the mean of a distribution is set to an actual data point. This helps the thesis algorithm to find more realistic clusters, but makes the total likelihood curve unstable (i.e. not a monotonic function of the number of clusters).

3.2.2 Merging criteria

This algorithm uses a likelihood approach instead of a distance approach. Instead of merging clusters that have the shortest distance to each other, merging is based on how likely the merge is. There are a few different ways to choose merging criteria. Two are presented here.

- The most straightforward way is to choose the merge that results in the *smallest reduction of the total likelihood*. Let ℓ_k denote the sum of log likelihood for the clustering at the k:th iteration and let $\ell_{k+1}(i,j)$ denote the sum of log likelihood of the clustering at the $k+1$:th iteration if cluster i and j are merged. The merging criteria can then be formulated as a distance minimisation problem with the following distances.

  $$d_k(i,j) = \ell_k - \ell_{k+1}(i,j)$$

  It is important to stress that a more complex model should always have a higher likelihood value due to its higher capability of fitting the data. When constraining the model, i.e. merging two clusters, there is an inevitable decrease in total likelihood value. Thus, $d_k(i,j) > 0$. By choosing the merge which results in the minimum decrease of total likelihood, the maximum likelihood for the constrained model is obtained. This leads to an hierarchy of models where every model is the maximum likelihood model under the constraints. In terms of distances, the merging criteria is

  $$(i_{[k,merge]}, j_{[k,merge]}) = \text{argmin}_{i,j} d_k(i,j)$$
  $$\ell_k = \ell_k\big((i_{[k,merge]}, j_{[k,merge]})\big)$$

- Although being consistent with the notion of maximum likelihood, the former merging criteria is skewed to favour the concatenation of small clusters over the concatenation of large clusters. There may be outliers in the data that truly are their own independent clusters and should not be merged with any other points or clusters. Such a merge would be unlikely, but due to the low number of points, the total impact may be very small. Compare this with a merge of two clusters containing several hundreds of points. A merge of that magnitude may result in small changes in likelihood for each point, but the sheer number of points make the decrease of total likelihood noticeable, even though such a merge may seem natural. In that situation, the former merging criteria may merge small unlikely clusters prior to large, likely clusters.

  A solution to this is to use the mean decrease of sum of log likelihood as merging criteria.

  $$d_k(i,j) = \frac{\ell_k - \ell_{k+1}(i,j)}{n_i + n_j}$$

  where n_i and n_j are the number of objects in cluster i and j respectively. This merging criteria does not guarantee a maximum likelihood model. Another way to avoid merging outliers is to assign a small covariance matrix to them. Since that has to be done a priori, the algorithm needs extra hyperparameters and is thus less attractive to use. From this perspective, this second merging criteria may be favourable over adding an extra hyperparameter, despite leading to potentially cumbersome clusterings.

3.2.3 Stopping criteria

As stated above, the log likelihood of the data is ideally a monotone function of the number of data points. Simply choosing the number of clusters that maximises the log likelihood will always lead to putting each point in its own cluster, which is overfitting by definition and makes cluster analysis superfluous in the first place. An agglomerative algorithm creates a hierarchy and merges clusters until all data points are in the same clusters. This situation yields a model with very low log likelihood and is underfitting the data, also making cluster analysis superfluous. The intuitive criteria is to stop merging when a merge is unnatural, e.g. having a mean far away from the points, having a large covariance matrix or strange form on the resulting ellipsoid. Such events are partly measured by the log likelihood graph and signified by a large drop.

As seen in section 2.1.2, the standardised tests for deciding the number of parameters in a model are hard to use due to the low number of data points merged at most steps in the hierarchy. These obstacles are hard to overcome with a likelihood based model. Nevertheless, an unlikely cluster will have a lower likelihood and the likelihood plotted against the number of cluster can be studied manually. Together with the information about the number data points in the merged cluster, it should be possible to detect an unnatural cluster. For example, if two distant points are merged, their individual contribution to the decrease of the likelihood should be fairly large and detectable, either manually or by a decision algorithm. The disadvantage with this approach is that it only detects when small and unlikely clusters are merged. When large unlikely clusters are merged, the log likelihood is a better detector. On the other hand, when large clusters are merged, a likelihood ratio test, BIC or AIC may work properly.

Another approach would be to sample from the cluster distribution and measure how extreme the actual outcome is compared to the simulated outcomes. However, this is very time consuming since thousands of samples has to be drawn for each step in the hierarchy.

Due to the many obstacles that come with a robust method to evaluate whether a cluster is likely or not, the thesis algorithm will only present the total log likelihood and a visual representation to aid a manual decision for where to cut the hierarchy. For most practical applications, this is enough.

3.3 Model extensions

So far, the main idea of the model is presented. There are a few extensions that can improve performance, speed and switch focus of the algorithm.

3.3.1 Optimising the clustering algorithm

Estimating Gaussian distributions and calculating likelihood values consumes more computing power than calculating distances. This difference can make the GMM approach unfeasible for even relatively small data sets. One way to significantly improve the speed, without much of a loss in performance, is to integrate a normal agglomerative algorithm with average linkage into the algorithm. At an early stage when clusters are small and plentiful, if the initialisations are reasonable the GMM approach should not give different results from what the distance approach give. To see why, imagine merging two points into one cluster using the likelihood approach. Points that are closer to each other will have a higher likelihood, and the two points with the highest likelihood will be the points nearest to each other. If distance is used as merging criteria, the result will be the same. The advantages for using the likelihood merging criteria is more prevalent higher up in the hierarchy, when clusters are bigger and initialisations matter less. Usually, the fair, true or optimal number of clusters are much smaller than the number of data points. For example, all data sets in the open classification data base for time series UCR Time series Classification Archive [19] have far less classes than data points. If the number of clusters happen to not be much smaller than the number of data points for some reason, the likelihood approach (or any other clustering algorithm for that matter) is not likely to yield good results in the first place, which gives an additional motivation as to why it is safe and sound to use this mixture approach in optimisation purposes.

3.3.2 Reclustering

When the hierarchy is established and an appropriate cutting point has been chosen, it may be wise to recluster the data. Due to the hierarchical structure and problems with non-optimal initialisation, unnatural clusterings may occur in the data, exemplified by figure 3.2. This can be solved by reclustering, which is done by estimating mean and covariance for every cluster and calculate the likelihood for each point and each cluster. A new cluster assignment is given to each point based on its maximum likelihood for every cluster, yielding the result in figure 2.2.

This procedure is equivalent of one iteration in the EM-algorithm. If the hierarchical structure is not important for the application, the full EM-algorithm can be applied with the hierarchy as a suitable and stable initialisation.

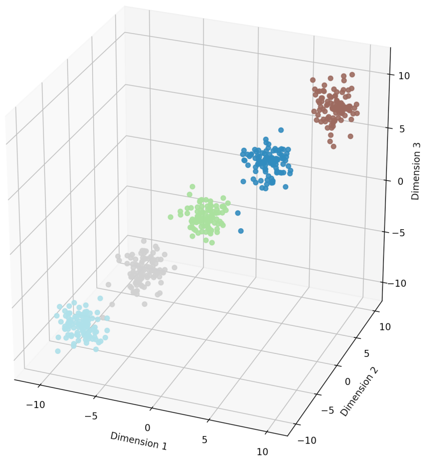

Figure 3.2: *The data portrays 5 easily separable clusters. With a suboptimal initialisation of the covariance matrix, the likelihood approach correctly suggests that the data should form 5 clusters, but fails to group them in what seem to be the most natural clustering. There should probably not be any grey points in the light blue cluster and no blue points in the green cluster. This is resolved by reclustering.*

3.3.3 Classification and outlier detection

This thesis is mainly concerned with clustering using autoencoders. However, small changes in the thesis algorithm can turn the algorithm into a classification algorithm instead of a clustering algorithm.

If the data set is partitioned into training and test sets, the training set can be used to create the latent space. When the training is done, the test data can be fed through the network and a nearest neighbour algorithm can be used to classify the test data. The clustering part can provide some extra information, but may be skipped. If the training of the network is done efficiently, this may be much faster than applying nearest neighbour at the crude data in high dimensional cases. This is because the autoencoder works as a dimensionality reduction on the data, making the thesis algorithm run faster.

Essentially, this alteration makes the autoencoder operate as a normal artificial neural network. The advantage is that it has a better framework for providing good visualisations and detection of outliers. However, an autoencoder will likely never be better than a classification ANN at classification tasks.

3.4 Hyperparameters

A generalised algorithm to solve any problem without any tuning or data preparation is still a utopian dream in the field of statistical learning. Although not achievable, it is well worth to strive for reducing the number of parameters that require tuning (i.e. hyperparameters) when implementing or creating an algorithm. This section lists and recapitulates all hyperparameters that are needed and states whether they are of great or little importance for the end result in the general case.

It should be stressed that the architecture of the autoencoder is strictly speaking not a hyperparameter, but it still requires a deliberate design to obtain good results and is treated as a hyperparameter in this list.

- As touched upon in section 3.1.2, it is important to have a sufficiently large network to encode and decode the data properly. It is generally better to have a large network. The time needed to train the network is quite small due to the constraints from the variational middle layer. Thus, a network larger than necessary does not increase time as much as a too small network will decrease the quality of the output.

An artificial neural network comes with a set of hyperparameters that all need to be tuned, which are the first to follow in this list.

- **Learning rate**. Learning rate corresponds to the length of the steps taken in the direction of the gradient. This is usually set to 0.01 or 0.001. For this application, learning rate is of less importance due to the variational middle layer, which in practice will put a random element on the gradient making meaningful learning impossible after a number of epochs, i.e. there is no need for the delicate fine tuning and precision associated with a low learning rate.

- **Epochs**. Epochs refers to the number of times the data set is fed through the network. In classification tasks, a high number of epochs is associated with overfitting. In the realm of autoencoders, overfitting is what enables clustering. A high number of epochs will not damage the result. However, every iteration consumes computing power, so the number should not be set to high. $500 - 3000$ is a good guideline.

- **Batch size**. Batch size corresponds to the number of data points that are fed through the network before applying the optimisation algorithm and gradient descent. For most usages, the batch size is ideally set to the same size as the data set. In the era of big data, this might be troublesome for many computers and the batch size hyperparameter helps cope with this by chopping the data set into chewable pieces.

- **Dimensions** d. This refers to the dimensionality (the number of nodes) in the middle layer of the autoencoder (i.e. the latent space) and has a huge influence over the functionality of the thesis algorithm. Fundamentally, it controls the amount of space available for the data in the latent space and does so exponentially. For visualisation purposes and small data sets, a 2 or 3 dimensional latent space is preferable. When working with large data sets with many clusters, more dimensions may be needed.

- β. This parameter controls the balance between the mean square error and the altered KL-divergence in the loss function 3.2. It is the least important loss parameter and can trivially be set to 1. When $\beta = 1$, initial testings show that the data tend to occupy a simplex over the interval $[-1, 5]$ in every dimension. If more space is needed, β can be set to a lower value but with the consequence that the autoencoder focuses more on reconstruction than on clustering.

- α. Directly controls the space available for in the latent space by penalising z^2. If more space is needed, α can be set to a lower value. Default value from KL-divergence is $\alpha = 1$.

- ν. Controls the standard deviation of the sampling function. Can trivially be set to $\nu = 1$, but may lead to poor clustering qualities. Ultimately, ν forces the latent representations to move, and if small or no movements are deployed, there are no reasons for the data points to form meaningful clusters. By choosing a larger ν, small standard deviations are penalised exponentially. Too large values of ν makes the autoencoder unstable because it is encouraged to increase the standard deviation to infinity and not caring about reconstruction precision at all. Together with the dimension of the latent space d, ν has the largest impact on the end result.

- **Pair**. To speed up the clustering algorithm, a normal agglomerative clustering algorithm is used. The hyperparameter **Pair** controls the number of clusters that are merged using this algorithm before switching to the likelihood merger. This parameter should be set so that the number of clusters merged by the likelihood merger is larger than the number of clusters in the data (which generally is unknown). A good guideline is 3/4 of data set size. For large data set with large clusters, such as in figure 3.2, this number can be set way higher.

- **Proportional**. This parameter gives two options on what clusters to merge using the likelihood merger 1) *True* merges clusters who's mean decrease in joint likelihood is the lowest 2) *False* merges clusters who's total decrease in joint likelihood is the lowest. This is thoroughly covered in section 3.2.

- **Cutoff**. This parameter sets boundaries on the size of the sample covariance matrix. This is done either with the use of the standard deviations given by the VAE or manually by specifying the maximum size of the covariance matrix. If Cutoff is set to false, no maximum limit is put upon the covariance matrix. The purpose of this hyperparameter is to discourage large clusters.

Chapter 4

Tests and Data

4.1 Data

The thesis algorithm is tested in several ways and compared with baseline methods. To do these tests, described in section 4.2, a few different data sets are used, which are presented here.

4.1.1 Generated clusters

The data in figure 3.2 will be used as a first sanity check on the second part of the thesis algorithm, i.e. the agglomerative hierarchical GMM. This data set contains 500 data points in a three dimensional space where the points are grouped in 5 groups with 100 points in each. The groups are centred along the natural diagonal with mean $[-8, -4, 0, 4, 8]$ and the identity matrix as covariance matrix.

4.1.2 Financial Time series

As the first chapter of this thesis indicates, cluster analysis can be a useful tool in a financial setting. To test and demonstrate its usefulness, a data set containing the daily value of 438 liquid funds is used. The original data set was larger, but non-liquid funds were removed to obtain the 438 funds. The reason is that the VAE did not manage to reconstruct the non-liquid funds during experimental runs. This will be discussed more thoroughly in chapter 6.

The time series are 2601 long and covers fund values from 2008 to 2018. Since this thesis covers unsupervised learning methods, the data set is not parsed into training data and test data and performance will not be measured using validation data. However, the fund names will be used for qualitative analysis of cluster quality.

The data set contains pure monetary values and is transformed into standardised log returns. If Z is the original data, X is the transformed data according to the following transformation.

$$X_{i,t} = \frac{\log\left(\frac{Z_{i,t+1}}{Z_{i,t}}\right) - \mathbb{E}\left[\log\left(\frac{Z_{i,t+1}}{Z_{i,t}}\right)\right]}{\sigma_i} \tag{4.1}$$

where σ_i is the sample standard deviation of the log returns $\{\log(\frac{Z_{i,t+1}}{Z_{i,t}})\}$ and $t \in [1, \dots, 2858]$. This makes X a matrix of size 438×2600. When using mean square error as loss function in the autoencoder, it is important that the input data is on the same amplitude scale and shifted to the same mean. Two transformed time series are shown in figure 4.1. When this transformation is done, the invariances presented in section 2.5 are somewhat satisfied with the distance measure used by the autoencoder. Scaling and translation is cared for by the transformation and the complexity invariance will be taken care of by the autoencoder. Financial data is shift invariant and uniformly scaled by default.

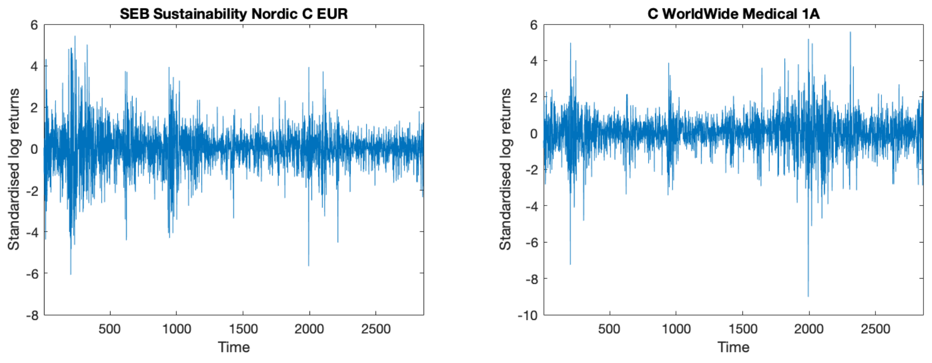

Figure 4.1: *Two transformed time series from the fund data set.*

4.1.3 UCR Times series Classification Archive

The UCR Times series Classification Archive is an archive of time series used to test classification algorithms and compare the results with other methods of classification [19]. The archive contains 128 data sets of times series and displays the out of sample prediction rate for several common methods of classification. Since true labels of the time series exist and are used to train those methods, they are considered *supervised* learning methods. As stated before, this thesis is concerned with *unsupervised* learning and will thus use this archive in another way than the classification methods do. Instead of predicting the right label of a time series, the thesis algorithm will be evaluated on how well it clusters those labels without knowing the true labels. If the thesis algorithm works well, it should be able to find the classes Y without knowing how many they are or what they look like.

The specific data sets used are presented in the following list. What the data sets actually resembles is not discussed further.

- **Plane**. The data set **Plane** contains 209 times series of length 145 and is divided into 7 classes. The best classification method (DTW, section 2.5.1, together with 1 nearest neighbour) manages to correctly classify all out of sample time series, which indicates that the data set should be fairly easy to work with concerning clustering algorithms. All members of 4 out of the 7 classes are shown in figure 4.2

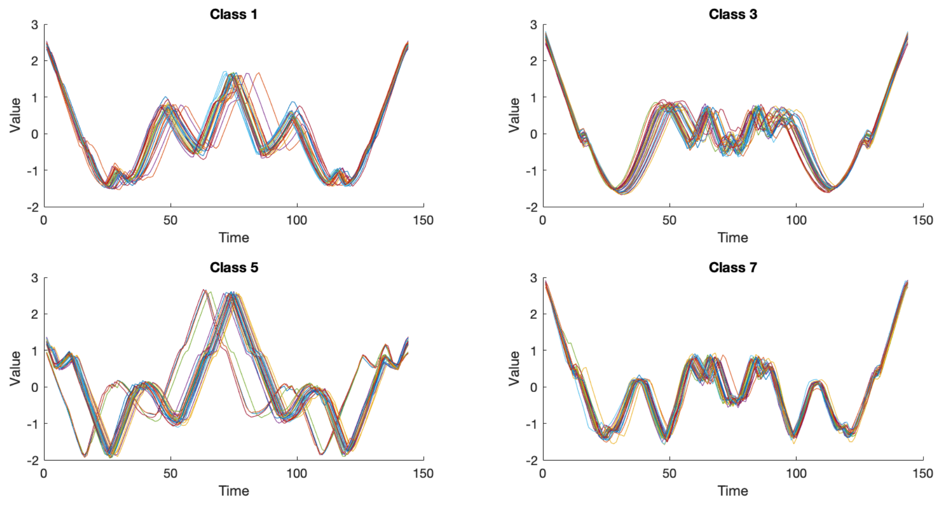

Figure 4.2: *Classes 1, 3, 5 and 7 are shown in each subplot respectively. The classes posses distinguishable and unique features.*

- **Fish**. This data set contains 350 times series of length 463, divided into 7 classes. The top classification rate presented is 0.1543. All members of 4 out of the 7 classes are shown in figure 4.3

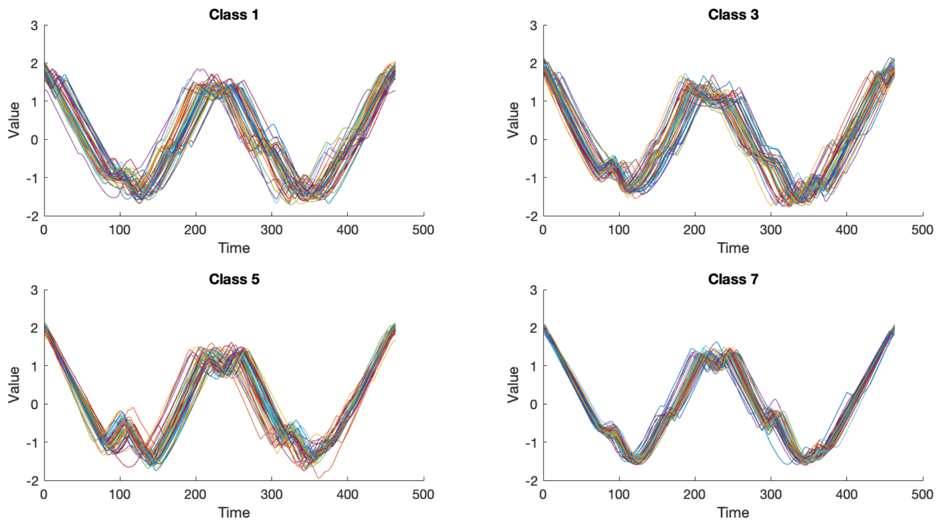

Figure 4.3: *Classes 1, 3, 5 and 7 are shown in each subplot respectively. This data set does not display very clear shapes or features. It is far from easy to classify them by looking at them.*

- **Gun Point**. The data set has 200 time series of length 150, divided into 2 classes. The top classification rate presented is 0.087. All time series are shown together with their class members in figure 4.4.

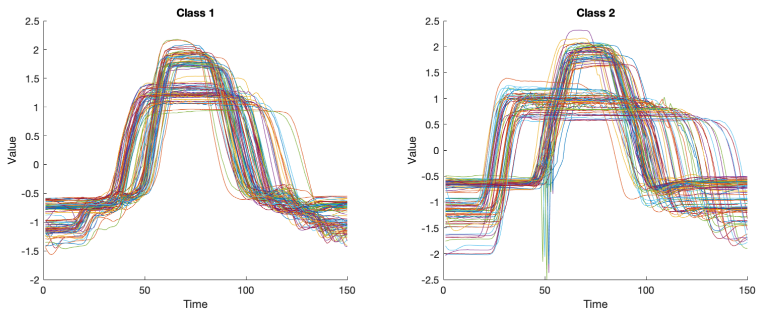

Figure 4.4: *Each plot shows all time series belonging to the specified class. Within both classes there seem to be at least 2 subclasses.*

- **Italy**. The data set has 1096 time series of length 24, divided into 2 classes. They are shown in figure 4.5 grouped by class.

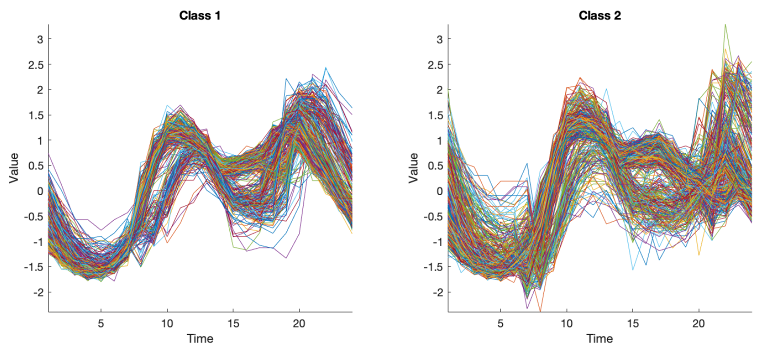

Figure 4.5: *Each plot shows all time series belonging to the specified class. The classes are not clearly distinguishable and covers a large area of the space.*

It is noted that the invariances from section 2.5 are not necessarily satisfied for these data sets. The purpose is however to test if the thesis algorithm will work despite not being given preprocessed data.

4.2 Tests

This section describes the tests made to evaluate and measure clustering performance. Each subsection shows how the data is used and describes the specifics of the tests.

4.2.1 Sanity check

The generated data from section 4.1.1 is fed through the agglomerative hierarchical Gaussian mixture model and is evaluated on whether it finds the true number of clusters in the data. A graph over the total sum of log likelihood for 1 to 30 clusters will be presented. The hyperparameters used are **Pair** = 2/3, **Proportional** = $True$, **Cutoff** = $False$.

4.2.2 Clustering financial data

In this section, the data presented in section 4.1.2, will be clustered using the thesis algorithm. Four application areas are investigated and the performance of the thesis algorithm is measured and compared to a clustering made on correlation. Before the applications are described, the hyperparameters has to be set and the clustering has be done. The thesis algorithm will besides from give each point a cluster assignment also give information on the number of likely clusters in the data through the total log likelihood graph.

Two tries with different hyperparameters will be conducted. Table 4.1 shows the choice of hyperparameters. These tests will be conducted on time series of length 100. The variational autoencoder will contain the following layers.

1. **Input layer**: dimension 100.
2. **Intermediate layer**: dimension 64 with dense layer and an ELU activation function
3. **Intermediate layer**: dimension 16 with dense layer and an ELU activation function
4. **Variational middle layer**: dimension d with ELU activation function.
5. **Intermediate layer**: dimension 16 with dense layer and an ELU activation function
6. **Intermediate layer**: dimension 64 with dense layer and an ELU activation function
7. **Output layer**: dimension 100 with dense layer and a linear activation function

	Test 1	Test 2
Learning rate	0.001	0.001
Epochs	3000	3000
Batch Size	32	32
Dimensions	3	2
β	1	0.001
α	2.5	1
ν	2.5	1
Pair	$\frac{3n}{4}$	$\frac{3n}{4}$
Proportional	True	False
Cutoff	σ_z^2	-

Table 4.1: Hyper parameters for two clusterings. The variable n corresponds to the number of data points in the data set.

Test 1 utilises all the tricks presented throughout the text. The parameters α and ν are tuned on a training data set to be appropriate for the application. The Cutoff function is used to penalise clusters that are much larger than the inherent standard deviation used by the VAE. Test 2 is simpler and uses less tuning. All hyperparameters are set to their default value and no Cutoff function is used. These two tests aim at testing whether the extra hyperparameters and tricks are useful in practice.

Both tests are done 10 times each and their stability is measured, i.e. since the autoencoder is a Monte Carlo algorithm (produces different outputs at different trials despite using the same input data), it is crucial that the resulting clusters do not diverge too much from each other. This can be compared with deterministic algorithm such as a correlation based clustering algorithm, which will produce the same clustering every time it is fed the same data. To measure this, the metrics presented in section 2.2 are used on every pair of clustering, which results in $\binom{10}{2} = 45$ measurements for every metric. The mean of every metric is presented. The Rand index is measured for $\alpha = 0$ and $\alpha = 1$.

For the application below, the best set of hyperparameters is chosen and used. To enable testing, every clustering turn will use 25 clusters. The reason is that even if different data partitions may contain different number of clusters, the test statistics derived are easier to compare with each other if the number of clusters is fixed. The number 25 is chosen from observations from an experimental data set.

Two clusters will be analysed qualitatively to see if they correspond to some real world grouping such as geography or industry, or some other intuitive trait. One cluster will undergo a likelihood analysis to see if outliers in the graph are labelled with a lower likelihood than other cluster members, i.e. if the thesis algorithm considers them less representative of the cluster and possibly being outliers.

At last, the most and least prominent stock throughout the whole time period will be displayed. With prominent it is meant that the fund which is far from the cluster center and is in a large cluster is considered to have low prominency, and a fund close to the cluster center in small clusters are regarded as highly prominent.

Portfolio Optimisation

This test corresponds to reducing the dimensionality of the data to make an optimisation algorithm smoother, faster and more stable. As argued in the first chapter, there are plenty of advantages of clustering funds before optimising a portfolio over them. Provided with the cluster assignments from the section 4.2.2, the following optimisation algorithm, provided by Kidbrooke Advisory AB, is used and works as described by algorithm 1. This optimisation algorithm uses a utility function, defined in section 2.9.2.

Algorithm 1: Portfolio optimisation

```
Result: Obtain portfolio weights over the total set of assets.
Given:
$N$ clusters,
$T$ time horizon
$M$ number of simulations;
Create $N$ series by merging the series in each of the $N$ clusters.
for $i = 1$ to $M$ do
    Sample $T$ time points from $[1, T]$;
    Simulate $T$ days of movements by drawing the return at time $t$
    for every sampled time point and for every cluster.
    Multiply the returns to obtain cluster values $T$ days after purchase.
end
Feed the $M \times N$ end values through a utility function Optimise weights $w$ over the $M$ instances. Buy
  clusters according to $w$.
```

The test will be conducted 25 times with a rolling window approach on the data. A clustering will be formed on 100 time points, which then will be fed to algorithm 1. The suggested portfolio is then bought and the total value of this portfolio is evaluated 50 days after purchase for risk aversion levels r within the interval $r \in \{2, \dots, 15\}$. The results are compared to a portfolio constructed based on a correlation clustering using distance 2.15 and an average linkage agglomerative hierarchical clustering over the same data. As reference, an equally weighted portfolio is also presented and evaluated. This is repeated 20 times by shifting the data 100 time steps, i.e. the first clustering is made on time $[0, 99]$ and the portfolio is evaluated on $[100, 149]$, the second clustering is made on time $[100, 199]$ and evaluated on $[200, 249]$ etc. To evaluate the performance, the win frequencies of the portfolios are presented. Along with the win frequency, the results of a trading strategy over the period is presented. The strategy is to buy the suggested portfolio and sell it at the evaluation day and repeat it for every portfolio constructed. Since portfolio optimisation is a crude way of testing the clustering methods, no hypothesis tests will be conducted and only very conservative conclusions should be drawn.

Temporal stability of clusters

For many financial applications, clusters of stocks or funds are ideally stable over a reasonable period of time. And if they are not stable, there should be some meaningful reason to why they are not. To measure such stability, the three metrics presented in section 2.2 are tested on consecutive, non overlapping clusterings from the portfolio optimisation. The Rand index is measured for $\alpha = 0$ and $\alpha = 1$. The hypothesis is that the thesis algorithm yields stabler clusters than the correlation based clustering algorithm defined in the previous section.

More specifically, each clustering is made of series of length 100. To measure temporal stability, the clustering over time interval $[0, 99]$ is compared to the clustering over time interval $[100, 299]$, and the clustering over time interval $[100, 199]$ is then compared to the clustering over time interval $[200, 299]$ and so on. This results in 25 measurements of temporal stability $T_{m,AE}$ for each metric m. A clustering based on correlation is made (as defined in the previous subsection) and temporal stability of the resulting clusters are calculated, yielding 25 measurements of temporal stability $T_{m,corr}$ for each metric. The hypothesis to test is if the thesis algorithm creates stabler clusters, i.e. if $T_{m,AE} > T_{m,corr}$.

If the hypothesis is false, both $T_{m,1}$ and $T_{m,2}$ should come from the same distribution. Let $z_m = T_{m,AE} - T_{m,corr}$. The null hypothesis and the alternative hypothesis are:

- H_0: $T_{m,AE}$ and $T_{m,corr}$ are from the same distribution: $\bar{z_m} \sim N(0, \frac{\sigma_{z_m}}{\sqrt{n}})$
- H_1: $T_{m,AE}$ and $T_{m,corr}$ are not from the same distribution: $\bar{z_m} \sim N(\mu, \frac{\sigma_{z_m}}{\sqrt{n}})$. $\mu > 0$

To reject the null hypothesis, $\bar{z_m}$ has to be observed to be significantly different from zero. Since σ_z is unknown and estimated, $\bar{z}$ is tested with the t-statistic following a Student's t-distribution, where σ_{z_m} is estimated using

s [20]. A Student's t-test is motivated by the fact that z_m are approximately mutually independent. Thus, by the central limit theorem, the distribution of $\bar{z_m}$ should converge to $N(\mu, \frac{\sigma_{zm}}{\sqrt{n}})$ [20], which motivates the use of normal distribution in the hypothesis test.

The test statistic t is defined as

$$t = \frac{\bar{z_m} - 0}{\hat{s}/\sqrt{n}}$$

With $n = 25$, t follows a t-distribution with 24 degrees of freedom.

Predicting cluster divergence

A clear advantage with GMM in clustering is that it allows for soft cluster borders. Cluster members close to the centre are considered to be stable and cluster members closer to the hard cluster border is by the same logic less stable. This corresponds to saying that some funds are being outliers in a cluster and are more likely to be there by chance than by "true connection" to the other members of the cluster. Equivalently, some funds are more representative of the cluster and are more likely to be clustered together also in the future. The hypothesis is that it is possible to predict which cluster members that are more likely to be in another cluster in the next time period.

To test this hypothesis, the temporal stability test is conducted once more, but with unequal weights put on each fund. Funds that are closer to a cluster centre (thus considered more stable) are weighted heavier than funds closer to a cluster border. This means that when a fund closer to the border ends up in another cluster the next time, this has a smaller impact on the metric than if a stable fund ends up in a different cluster. Thus, if the hypothesis is true, i.e. that it is possible to predict which funds that are more likely to change cluster, higher metric values are to be expected. If the hypothesis is false, the metrics should be the same as when having equal weights on the funds.

The hypothesis test is conducted in the same way as when testing temporal stability, but with the following null hypothesis and alternative hypothesis. $T_{m,AEw}$ corresponds to the weighted metric.

- $z_m = T_{m,AEw} - T_{m,AE}$.
- H_0: $T_{m,AE}$ and $T_{m,AEw}$ are from the same distribution: $\bar{z_m} \sim N(0, \frac{\sigma_{zm}}{\sqrt{n}})$
- H_1: $T_{m,AE}$ and $T_{m,AEw}$ are not from the same distribution: $\bar{z_m} \sim N(\mu, \frac{\sigma_{zm}}{\sqrt{n}})$. $\mu > 0$

Predicting future return distributions

The concept of *regression towards the mean* is a slightly unintuitive concept that says that a spectacular event is likely to be an over shoot estimation of the expected value of such events. This is conceptualised by the fact that very tall parents tends to have shorter children, and very short parents tends to have taller children [21]. The explanation is that an exceptionally tall person is more likely to be exceptionally tall due to environmental circumstances than due to exceptional genes. The children of this person are expected to have a height corresponding to their genetic predisposition, which likely means to be shorter than their tall parent. By including genetic data from other people, the height of the children can be better estimated than when estimating using only the parents genes and height.

The hypothesis to be tested here is that it is true also for return distributions. The hypothesis says that a spectacular stock trajectory is in most cases caused by chance rather than by a true underlying reason. Specifically, those events may be ascribed to chance far more often than less spectacular events (such as steady growth over a decade). The hypothesis is that by including price movements of similar stocks, the distribution future returns for the fund at hand is captured more accurately than if only the historical trajectory of the stock is used. Spectacular events are filtered out and an estimation robust to over fitting is obtained. Figure 4.6 illustrates this.

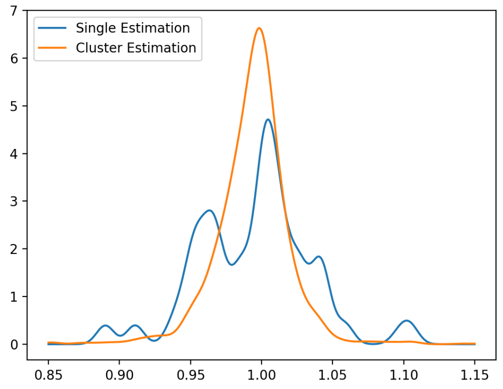

Figure 4.6: *Two estimated distributions are displayed. The blue distribution is fitted using data from a single fund and does probably not generalise well to the future distribution of the price movements. The yellow distribution is fitted using data from the whole cluster that the fund is a member of. This distribution shows no signs of being overfitted and the hypothesis says that it will be a better predictor the future distribution of the price movements.*

With this concept in mind, this section outlines a strategy to predict the future distribution of a fund by using price movements from funds in the same cluster. This strategy is tested by comparing the accuracy of the predicted distribution with the accuracy of a predicted distribution based solely on the historical data of the fund of interest. The quantitative measurements and definitions for this analysis is presented in section 2.9.1. The hypothesis is that the future distribution of a fund $p_{z,\hat{\tau}}(x)$ is better estimated by $p_{C_{z,\tau}}(x,\beta)$ than by $p_{z,\tau}(x)$. This is tested by measuring the mean MISE and mean KL-divergence between $[p_{C_{z,\tau}}(x,\beta), p_{z,\hat{\tau}}(x)]$ and $[p_{z,\tau}(x), p_{z,\hat{\tau}}(x)]$ for every τ, and using a t-test to see if there exists a significant difference in the expected mean value.

There are N fund z_i, $i \in \{1, \cdots, N\}$, and the data set is divided into T non overlapping time periods τ_j, $j \in \{1, \cdots, T\}$. MISE and KL-divergence is measured by the approximations eq. 2.26 and eq 2.27. For this application they are defined as.

$$MISE_{[p_{z,\tau}[x],p_{z,\hat{\tau}}[x]]} = \frac{1}{M}\sum_{i=1}^{M}\left(p_{z,\tau}[x_i] - p_{z,\hat{\tau}}[x_i]\right)^2$$

$$MISE_{[p_{C_{z,\tau}}[x],p_{z,\hat{\tau}}[x]]} = \frac{1}{M}\sum_{i=1}^{M}\left(p_{C_{z,\tau}}[x_i] - p_{z,\hat{\tau}}[x_i]\right)^2$$

$$KL_{[p_{z,\tau}[x],p_{z,\hat{\tau}}[x]]} = \frac{1}{n}\sum_{i=1}^{n} p_{z,\hat{\tau}}[x_i]\cdot\log\Big(\frac{p_{z,\hat{\tau}}[x_i]}{p_{z,\tau}[x_i]}\Big)$$

$$KL_{[p_{C_{z,\tau}}[x],p_{z,\hat{\tau}}[x]]} = \frac{1}{n}\sum_{i=1}^{n} p_{z,\hat{\tau}}[x_i]\cdot\log\Big(\frac{p_{z,\hat{\tau}}[x_i]}{p_{C_{z,\tau}}[x_i]}\Big)\Big)$$

Now, define

$$Z_{MISE,\tau} = \frac{1}{N}\sum_{k=1}^{N} MISE_{[p_{z_k,\tau}[x],p_{z_k,\hat{\tau}}[x]]} - \frac{1}{N}\sum_{k=1}^{N} MISE_{[p_{C_{z_k,\tau}}[x],p_{z_k,\hat{\tau}}[x]]}$$

$$Z_{KL,\tau} = \frac{1}{N}\sum_{k=1}^{N} KL_{[p_{z_k,\tau}[x],p_{z_k,\hat{\tau}}[x]]} - \frac{1}{N}\sum_{k=1}^{N} KL_{[p_{C_{z_k,\tau}}[x],p_{z_k,\hat{\tau}}[x]]}$$

This results in T approximately mutually independent estimations of the difference in predictive performance between the two estimation methods. The hypothesis can then be tested with a t-test. The null hypothesis and the alternative hypothesis for the MISE measure are now defined as

- $H_{MISE,0}$:

  The expected mean integrated squared error is the same for both models : $z_{\bar{MISE}} \sim N(0, \frac{s_{z_{MISE}}}{\sqrt{n}})$

- $H_{MISE,1}$:

  The expected mean integrated squared error is higher for the single model. $z_{\bar{MISE}} \sim N(\mu, \frac{s_{z_{MISE}}}{\sqrt{n}})$. $\mu > 0$

The null hypothesis and the alternative hypothesis for the KL-divergence measure are now defined as

- $H_{KL,0}$:

  The expected KL-divergence is the same for both models : $z_{\bar{KL}} \sim N(0, \frac{s_{z_{KL}}}{\sqrt{n}})$

- $H_{KL,1}$:

  The expected KL-divergence is higher for the single model. $z_{\bar{KL}} \sim N(\mu, \frac{s_{z_{KL}}}{\sqrt{n}})$. $\mu > 0$

The hypothesises will be tested with 3 sets of hyperparameters, presented in table 4.2

	Test 1	Test 2	Test 3
N	438	438	438
T	25	25	25
σ	0.01	0.01	0.01
M	1000	1000	1000
β	0.25	0.5	0.75

Table 4.2: Hyper parameters for predicting future distributions of stock movements.

$M = 1000$ defines the number of point used to approximate the integral. The interval investigated is $x \in [0.85, 1.15]$, i.e. MISE and KL-divergence is only estimated on this interval. Three different $\beta : s$ are chosen to investigate how much influence the other cluster members should be given. The smoothing parameter σ is chosen to create smooth but neither overfitted or underfitted distributions.

4.2.3 Predicting number of classes in UCR data

None of the financial tests above tests the thesis algorithm directly. Instead, they use the results from the algorithm in different applications and see if those clusters perform better than a standard clustering algorithm. Furthermore, the tests above does not use the likelihood graph to analyse the number of clusters in the data, which is a crucial and important part of this thesis. To test these features, the data sets from section 4.1.3 are used and fed through the thesis algorithm. The latent space representations are plotted and the likelihood graph is analysed. Ideally, the data should form clusters corresponding to the true class labels. If they do, the thesis algorithm should be able to detect them as clusters and correctly guess the number of true classes in the data. The hyperparameters for the data sets are set according to table 4.3.

The stability of the algorithm is investigated on the data set with the most promising clustering results. This is done in the same way as the stability analysis above, iterating the thesis algorithm 10 times with the same input data and measure the consistency of the output.

	Plane	Fish	Gun Point	Italy Power Demand
Learning rate	0.001	0.001	0.001	0.001
Epochs	1000	2000	1000	1000
Batch Size	32	32	32	32
Dimensions	2	3	2	2
β	1	1	1	0.1
α	1	1	0.5	2
ν	1	4	3	2
Pair	$\frac{3n}{4}$	$\frac{3n}{4}$	$\frac{2n}{3}$	1000
Proportional	True	True	True	True
Cutoff	-	-	-	-

Table 4.3: Hyper parameters. The variable n corresponds to the number of data points in the data set.

All data sets use the same network architecture as the portfolio optimisation test, except their input and output dimension corresponds to their inherent dimensionalities.

Chapter 5

Results

5.1 Sanity Check

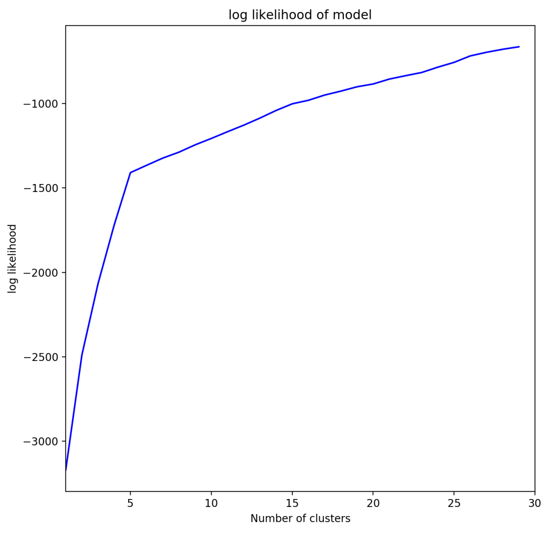

Figure 5.1: *The total log likelihood for each number of clusters for the toy data set portrayed in figure 3.2. The test was to see if the log likelihood plot possesses a significant drop between 4 and 5 clusters. Arguably, it does.*

5.2 Financial time series

5.2.1 Clustering -Test 1

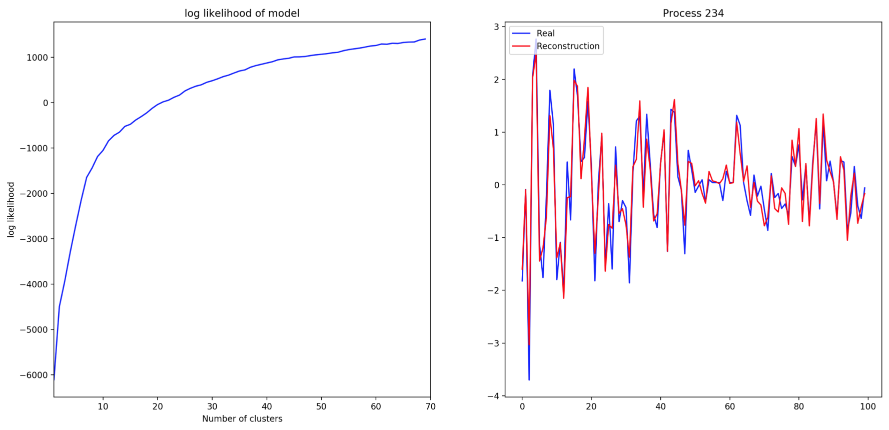

Figure 5.2: *The total log likelihood for each number of clusters a typical clustering of the financial data set. The log likelihood does not show a clear drop as seen in figure 5.1, which makes it harder to tell when to stop the clustering algorithm. The right plot shows a time series with its reconstruction, which seems to be a fairly accurate reconstruction. This plot is used to check that the autoencoder does what it is supposed to do and that the clustering result is reliable. An accurate reconstruction is thus a good result.*

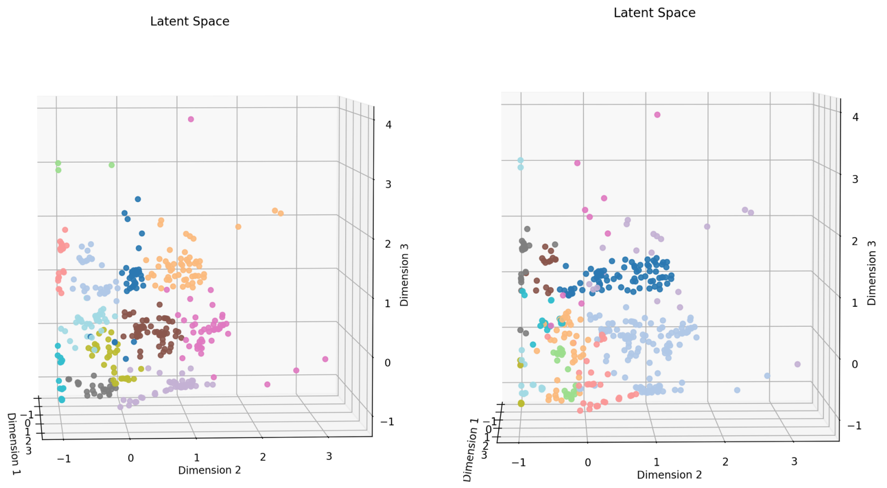

Figure 5.3: *Two colourings of a latent space. The left colouring is made by the GMM clustering algorithm, the right colouring is made by the normal hierarchical clustering algorithm with correlation based distances. The right plot is made as a reference to see if the two methods align. It is clearly seen that both methods create somewhat overlapping clusters, indicating that the methods are equivalent to some degree. For illustration purposes, both displays 12 clusters.*

5.2.2 Clustering -Test 2

The median standard deviation given by the VAE is roughly in the interval $[0.1, 1]$ for every dimension.

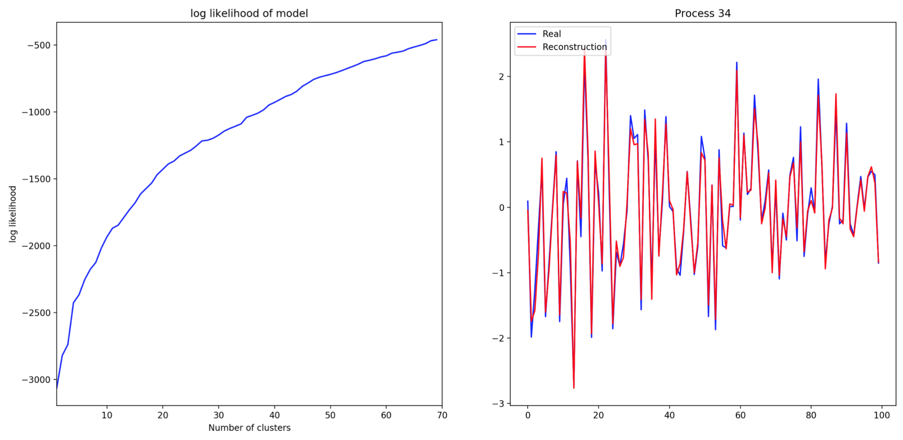

Figure 5.4: *The total log likelihood for each number of clusters a typical clustering of the financial data set. To the right, a time series with its reconstruction is plotted.*

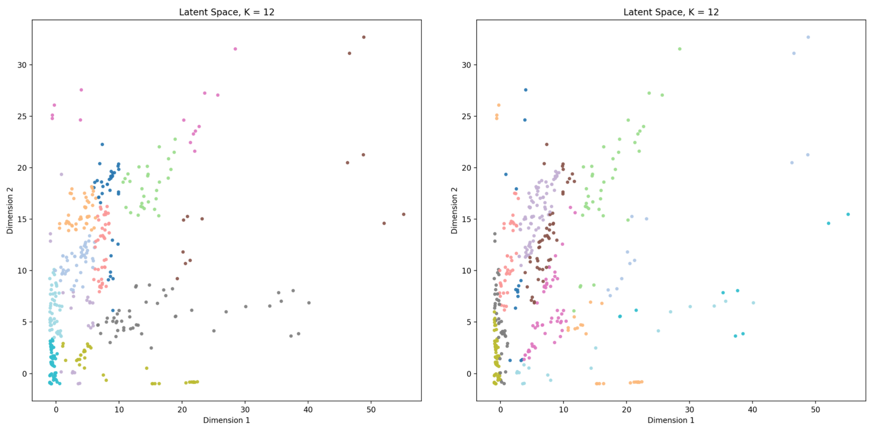

Figure 5.5: *Two colourings of a latent space. The left colouring is made by the GMM clustering algorithm, the right colouring is made by the normal hierarchical clustering algorithm with correlation based distances. Both displays 12 clusters.*

	Test 1	Test 2
$\textbf{Rand}_{\alpha=1}$	0.94	0.93
$\textbf{Rand}_{\alpha=0}$	0.60	0.41
Purity	0.77	0.54
Mutual Information	0.76	0.63

Table 5.1: Mean over 45 values of respective metric when testing the stability of the algorithm fed with the same input. The hyper parameters used in test 1 seem to make a stabler clustering.

Examples of two clusters

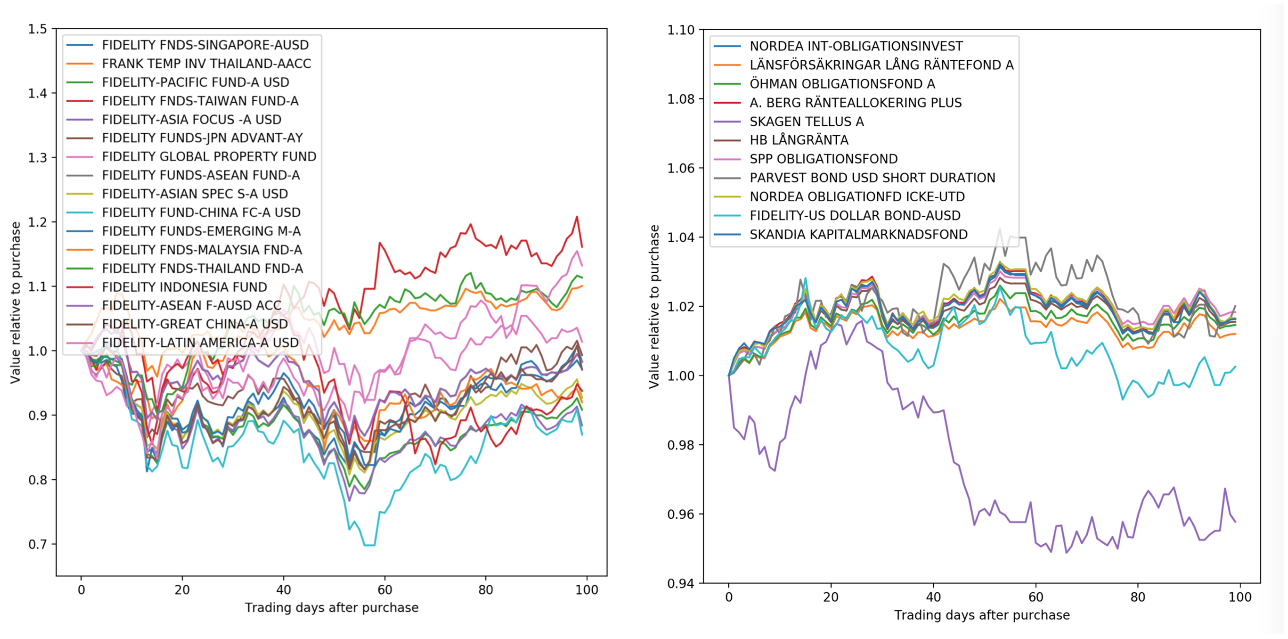

Figure 5.6: *Two examples of fund clusters. Both of them seem to possess similarities in geography, trajectory and type of asset.*

Fund Names	log likelihood
SKAGEN TELLUS A	0.5676
PARVEST BOND USD SHORT DURATION	0.6308
FIDELITY-US DOLLAR BOND-AUSD	1.9486
OHMAN OBLIGATIONSFOND A	3.8444
HB LANGRANTA	4.4303
SKANDIA KAPITALMARKNADSFOND	4.5164
SPP OBLIGATIONSFOND	4.6824
NORDEA INT-OBLIGATIONSINVEST	4.7484
LANSFORSAKRINGAR LANG RANTEFOND A	4.7570
NORDEA OBLIGATIONFD ICKE-UTD	4.7705
A. BERG RANTEALLOKERING PLUS	4.7936

Table 5.2: Log likelihood each fund in the right cluster in figure 5.6, sorted in order. A higher level signifies that the fund is close to the cluster centre.

- The least prominent fund in this analysis was: **JPMorgan Fund - Global Focus Fund A**.
- The most prominent fund in this analysis was: **Danske Invest SICAV Global Corporate Bonds Class A**.

With prominent it is meant that the fund which is far from the cluster center and is in a large cluster is considered to have low prominency, and a fund close to the cluster center in small clusters are regarded as highly prominent.

5.3 Financial applications

Clustering test 1 showed the best clustering abilities and output consistency, and was chosen for the applications.

5.3.1 Portfolio optimisation

- The thesis algorithm portfolio performed better than the correlation based portfolio in 59% of the tests.
- The thesis algorithm portfolio performed better than the equally weighted portfolio in 45% of the tests.
- The correlation based portfolio performed better than the equally weighted portfolio in 38% of the tests.

The relative profit made was:

- If thesis algorithm portfolio was bought instead of the correlation based portfolio, the average relative profit would be 1.12 percentage point.
- If thesis algorithm portfolio was bought instead of equally weighted portfolio, the average relative profit would be 0.46 percentage point.
- If the correlation based portfolio was bought instead of equally weighted portfolio, the average relative profit would be -0.0661 percentage point.

5.3.2 Temporal stability of clusters

All metrics used are presented in section 2.2. The null hypothesis says that the thesis algorithm and the correlation based clustering algorithm create equally stable clusters were rejected in 3 out of 4 tests. Instead the alternative hypothesis, i.e. that the thesis algorithm creates stabler clusters is chosen.

	μ	s	H_0	p-value
Rand$_{\alpha=1}$	-0.0065	0.011	Fail to reject	0.997
Rand$_{\alpha=0}$	0.1062	0.0724	Reject	4.0e-08
Purity	0.1360	0.0514	Reject	2.8e-13
Mutual Information	0.0799	0.0359	Reject	1.17e-11

Table 5.3: Mean and sample variance for observations together with results from the t-test with 25 degrees of freedom.

5.3.3 Predicting cluster divergence

The alternative hypothesis, i.e. that it is possible to predict which assets that are more likely to change cluster over time, was confirmed (the null hypothesis were rejected) in 4 out of 4 tests.

	μ	s	H_0	p-value
Rand$_{\alpha=1}$	0.004	0.002	Reject	6.3e-11
Rand$_{\alpha=0}$	0.019	0.011	Reject	1.9e-09
Purity	0.021	0.011	Reject	7.0e-11
Mutual Information	0.024	0.0082	Reject	2.7e-14

Table 5.4: Mean and sample variance for observations together with results from the t-test with 25 degrees of freedom.

5.3.4 Predicting future return distributions

Table 5.5 confirms that it is possible to make better predictions of the future return distribution when incorporating a small portion (25%) of cluster members into the estimation.

	μ	s	H_0	p-value
$\text{MISE}_{\beta=0.25}$	0.0047	0.0055	Reject	1.4e-04
$\text{KL-divergence}_{\beta=0.25}$	0.0062	0.0093	Reject	0.0013
$\text{MISE}_{\beta=0.5}$	0.0011	0.0087	No reject	0.27
$\text{KL-divergence}_{\beta=0.5}$	0.0061	0.0116	Reject	0.0072
$\text{MISE}_{\beta=0.75}$	-0.0147	0.0109	No reject	1
$\text{KL-divergence}_{\beta=0.75}$	0.0030	0.0127	No reject	0.122

Table 5.5: Mean and sample variance for observations together with results from the t-test with 24 degrees of freedom.

5.4 UCR Times Series Archive

5.4.1 Plane

Plot 5.7 shows the latent space representation of the Plane data set, which has clear and distinguishable cluster structures.

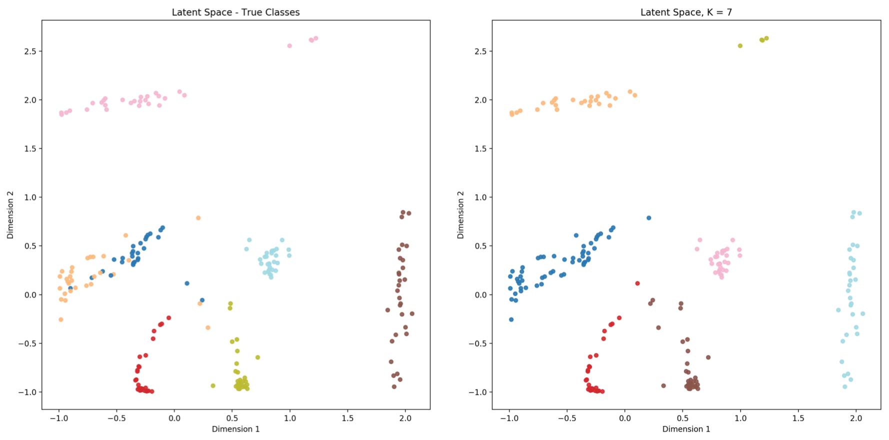

Figure 5.7: *The left plot shows the latent space coloured with the true class colours. The right plot shows the clustering given when the hierarchy is cut at 7 clusters.*

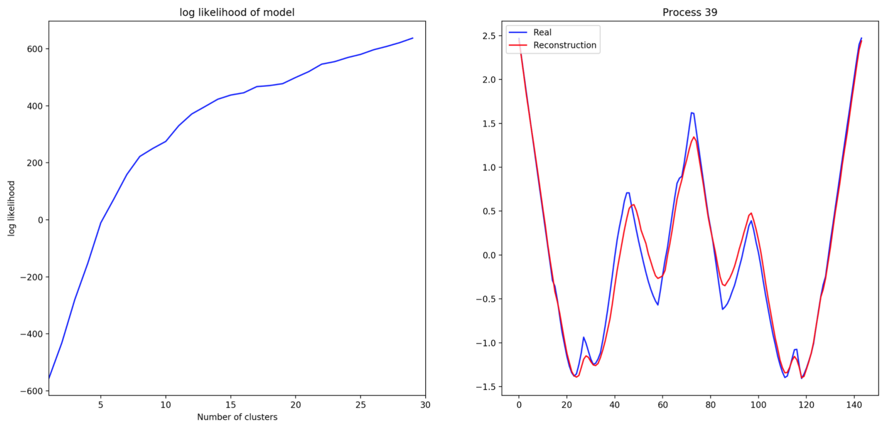

Figure 5.8: *The left plot shows the likelihood for each number of clusters. There is a huge drop in likelihood when for models with 7 or less clusters. The right plot shows a randomly chosen time series and its reconstruction.*

This data set showed the best clustering features and consistency analysis was made on the output. The results are presented in table 5.6.

	Results
Rand$_{\alpha=1}$	0.92
Rand$_{\alpha=0}$	0.80
Purity	0.86
Mutual Information	0.84

Table 5.6: Metrics of output consistency.

5.4.2 Fish

Figure 5.9 shows the latent representation for the Fish data set. The latent representations do not group themselves into clear clusters. Thus, no further analysis is made.

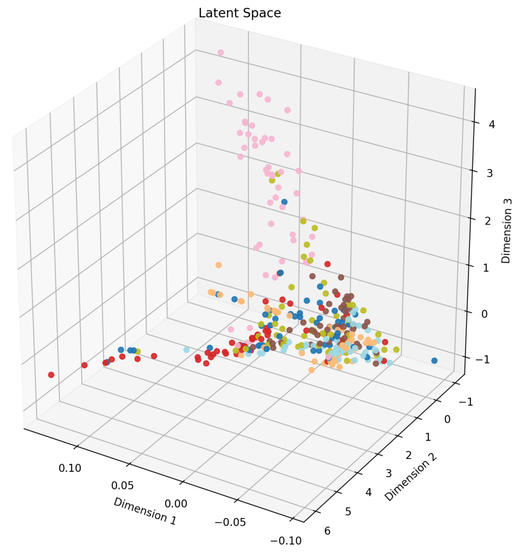

Figure 5.9: *This scatter plot shows the latent space for the Fish data set coloured with the true class values.*

5.4.3 Gun Point

For this data set, there seems to exist a meaningful clustering, shown in figure 5.10. The left plot shows the true classification, and it is seen that there are two clean clusters and one mixed cluster, which also might be expected when looking at the time series.

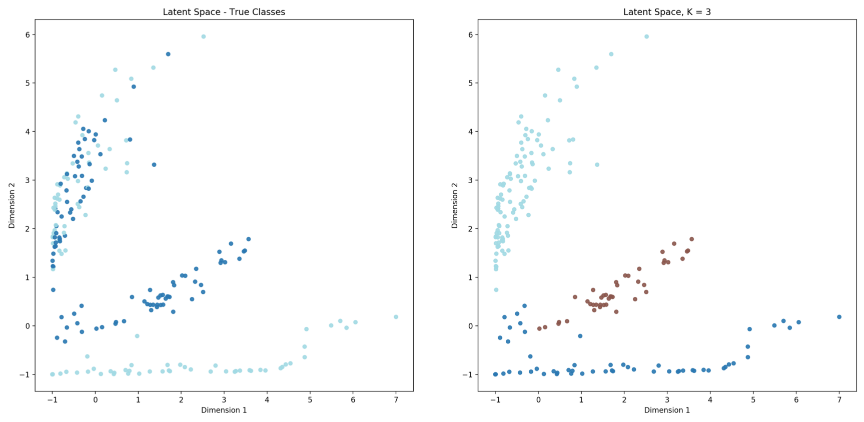

Figure 5.10: *The left plot shows the latent space of the Gun Point data set, coloured with the true class colours. The right plot shows the clustering given when the hierarchy is cut at 3 clusters.*

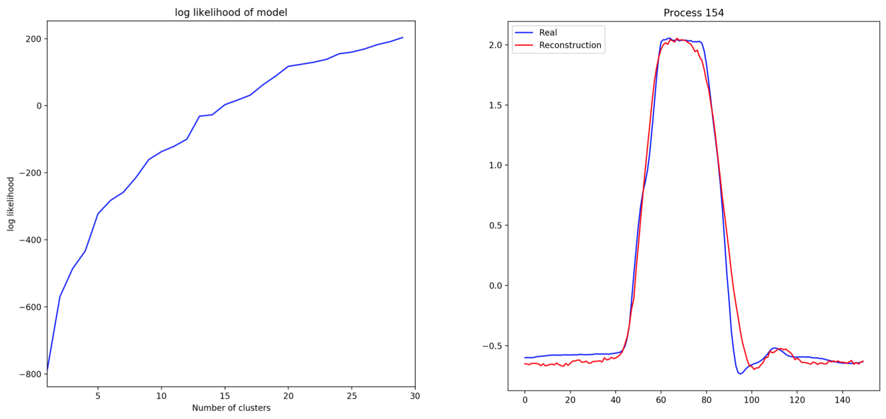

Figure 5.11: *The left plot shows the likelihood for each number of clusters. The right plot shows a randomly chosen time series and its reconstruction.*

5.4.4 Italy Power Demand

Figure 5.13 shows the latent space for the Italy Power Demand data set. There seem to be many clusters within the data set, which is expected when looking at the time series in figure 4.5. The left plot in figure 5.13 shows that there is a trivial partitioning that would split the data into almost completely separable clusters.

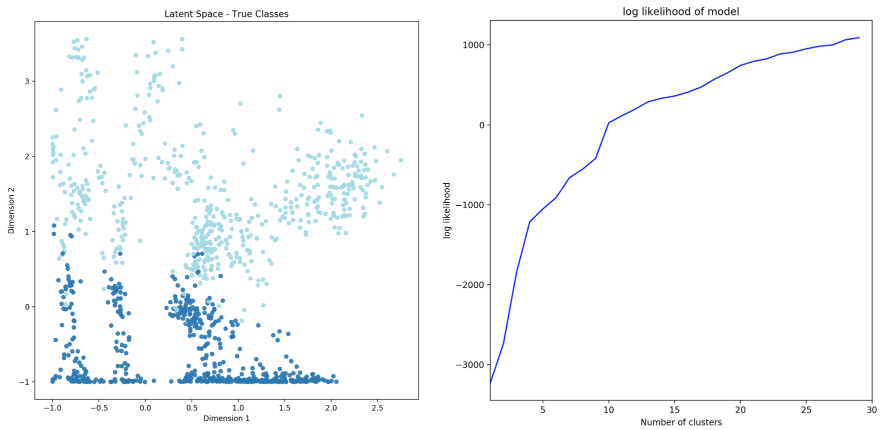

Figure 5.12: *The left plot shows the latent space of the Italy Power Demand data set, coloured with the true class colors. The right plot shows likelihood of the model for each number of clusters.*

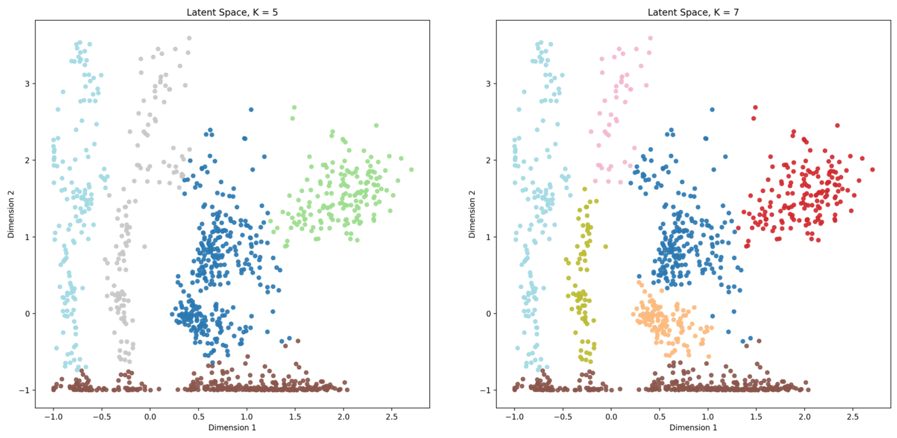

Figure 5.13: *Two plots with different clusterings. The left contains 5 clusters and the right contains 7 clusters.*

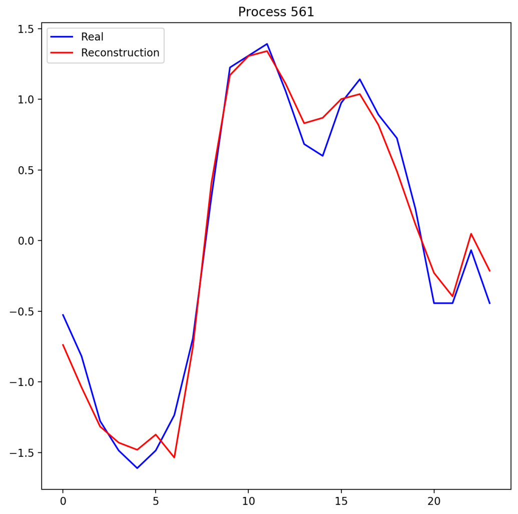

Figure 5.14: *A randomly chosen time series with its reconstruction.*

Chapter 6

Discussion

6.1 Financial time series

As mentioned in section 4.1.2, the original financial data set was shrunk to only include liquid funds. When all funds where used, the VAE failed to reconstruct non-liquid funds. They ended up around the origin in the latent space and their reconstructions where centred around zero on the y-axis, i.e. no structure at all was found and learned by the autoencoder. The reason why is probably that the funds had very few unique y-values and many zero y-values. This leads to a fundamentally different type of structure, and the VAE is not good enough at multitasking.

Non-liquid funds were removed in purpose of testing the thesis algorithm on friendlier data. It is however noted that the following results in the financial applications presented in chapter 5 are conditioned on the use of friendly data.

6.2 Sanity Check

The log likelihood graph shows a clear dip when the number of clusters is lower than 5. As discussed above, the total log likelihood is supposed to shrink with the number of clusters. However, when there is a large drop, the algorithm has merged two clusters that yield a very low likelihood value, i.e. it is very much more unlikely that the data was generated by a model with that n clusters compared to a model with $n + 1$ clusters. This is what happens when the model goes from 5 clusters to 4 clusters, which is exactly what the algorithm is designed to detect. Thus, the results of this test is very pleasing. The results were obtained with no specific tuning of hyperparameters, with makes the results promising for general applications.

6.3 Financial time series

The two tests show some what promising results. It is evident, especially in test 1, that the VAE manages to map the time series into meaningful clusters. There are big clusters of assets and some outliers that not really connect to any cluster. The hyperparameters are quite easy to tune for a given data set, but may generalise poorly. When using a rolling window approach over a time series, there might be a need for different hyperparameters over different time intervals. If the hyperparameters generalise poorly, the thesis algorithm may be hard to use in practice.

Figure 5.3 shows the clustering made by the hierarchical GMM to the left and the clustering made by the correlation based hierarchical clustering. It is clear that the two methods agree on the cluster assignment in general, but disagree to some extent. They certainly agree to the extend that good results can be expected from the VAE method.

Both tests showed poor performance in output consistency, displayed in table 5.1. It is far too unstable to be reliable. The reason is probably that the hierarchical GMM is very sensitive to small changes and prone to overfitting. The VAE outputs different latent spaces each time, but they are all somewhat consistent and a more robust clustering algorithm should definitely produce the same clustering despite the being given different latent representations, which is discussed together with figure 6.1.

The poor results on output consistency may be due to an erroneous choice of number of clusters. For this application, this number was chosen to be 25. If the "true" number is far higher, it may be expected to have inconsistencies in output. The reason is that if too small a number is chosen, clusters that shouldn't be merged are merged. By looking at figure 5.3, it is seen that the algorithm definitely is better at finding similar objects than distinguish between dissimilar objects. Thus, it may be prone to merge different dissimilar clusters at every new try, which in turn contributes to bad consistency scores. If so, better scores should be expected with a larger number of clusters. However, the metrics presented in section 2.2 are trivially higher for a large number of clusters, so it is hard to test this hypothesis.

The hyperparameters from test 1 shows a better visual representation of clusters and also performs better on the consistency test than test 2. Thus, the hyperparameters from test 1 were chosen.

Qualitative analysis of two clusters

Figure 5.6 shows two typical and presentable clusters from figure 5.3. Both will briefly be analysed here.

- The first cluster (to the left in figure 5.6) shows a clear geographical relationship and almost all members are issued by the same financial corporation. Most funds are invested in the asian market with the exceptions of one global property fund, one Latin America fund and one pacific fund. These three outliers follow the general cluster and are thus reasonable cluster members. Otherwise, the cluster seems to be homogenous in trajectories and no outliers can be identified.
- The second cluster also shows a clear geographical relationship. Furthermore, many of the funds are based on interest rates or bonds. The purple fund on the bottom (Skagen Tellus A) does not follow the general cluster path, but is still correlated with the other members. If no transformation of the initial data set would have been made, the VAE would probably not have put this fund in this cluster. A likelihood analysis of the cluster members is made and table 5.2 shows the results. Here it is seen that Skagen Tellus A has the lowest log likelihood in the cluster, indicating that the thesis algorithm is in concordance with intuition. Table 5.2 also says that Parvest Bond USD Short Duration and Fidelity USD Bond-AUSD are somewhat unlikely members of this cluster, but more likely than Skagen Tellus A. These results are in line with Property 2 from section 2.1, i.e. dissimilar objects should be put into different clusters. However, to verify that similar objects are put in the same cluster, comparisons with other clusters have to be made. By just looking at the graph, it is reasonable to suspect some underlying process to be driving the funds. When taking their names into account, this is confirmed to be true.

One of the motivations behind this thesis is to investigate whether common underlying processes can be extracted by grouping the data. The two clusters in figure 5.6 shows that it is reasonable to expect it to be possible, which is investigated further the applications presented below. The results from those tests are to some extent verifying that it is advantageous to extract and work with underlying processes instead of the crude data.

6.4 Financial applications

6.4.1 Portfolio Optimisation

The results from this test happened to be successful with a positive relative return on the investment when using the strategy provided by the VAE. As stated in section 4.2.2, a positive result does not necessary imply a superior clustering algorithm. The portfolio optimisation algorithm is very basic and does not utilise many easily implemented extensions and does not utilise all information given by the VAE. For example, the distribution that the returns are drawn from can be estimated better using the technique from section 4.2.2 and the underlying process can be extracted in a smarter way using more elaborate methods. However, even if a better algorithm was used, it would be hard to evaluate the thesis algorithm, which in the end is the motivation of this thesis. Thus, a positive result does not imply a superior clustering algorithm, but it is indicating that the VAE is reasonable and can be used for portfolio optimisation and that it is encouraged to investigate how this framework can be used for businesses focused on trading, asset analysis and portfolio construction.

6.4.2 Temporal stability of clusters

The t-tests show that the null hypothesis can be rejected in 3 out of 4 tests. This imply that the expected cluster stability is significantly larger for the thesis algorithm compared to the correlation based clustering. It is not evident why the VAE should create more stable clusters, so the strong results (low p-values) should be taken with much doubt. One reason for the significant results may actually be that the VAE is *worse* at creating meaningful clusters. In figure 5.3, 5.5 and 5.13, many points stack up on -1 in every dimension, since the loss function from equation 3.2 discourages point to go lower than -1. When estimating a normal distribution, points that are very close to each other in at least one dimension are unproportionally encouraged to be in the same cluster. Unfortunately, this does not necessarily imply that they are similar in reality. Thus, the significant results in this section may be a product of bad performance. There are extensions to the thesis algorithm to prevent this to effect the clustering, but would imply extra hyperparameters and non-monotone likelihood functions.

6.4.3 Predicting cluster divergence

These t-tests strongly indicate that it is possible to predict which time series that are more likely to change cluster assignment in the future. In contrast to the previous section, there is a clear and strong reason for why it should be possible. Cluster assignments are made on the likelihood of be a part of a cluster, and every point is assigned to the cluster corresponding to the most likely membership. However, some points may be almost as likely to be a member of another cluster, i.e. being closer to the cluster border. Although not a surprising result, it is very useful due to the lack of a corresponding feature in normal agglomerative clustering. The notion of being close to the border between two clusters lacks a clear and intuitive definition in a high dimensional space. The borders are noisier due to the high dimensionality and there is no likelihood framework to use for probability calculations. This is a very advantageous feature of the thesis algorithm that have many real world applications discussed in section 6.7.

6.4.4 Predicting future return distribution

The t-tests show different results for different values of β. With a high value of β, i.e. when the cluster members influence the prediction to a large extend, the prediction accuracy seem to be worse. But when β is low, the performance seem to increase. In the context of the tall parent and his/her child, this means that it is more important to look at the parents height than looking at other peoples genes, but it helps to look at other peoples genes. This is an extremely interesting and useful result that can be used in many applications, for example to improve the sampling accuracy in the portfolio optimisation algorithm 1. However, it is not clear that including data from other cluster members is superior to include data from any stock in general. It is reasonable to suspect that just filling in with more data in general makes the prediction better. On the other hand, it is also reasonable to suspect that the predictions are even better if cluster data is used. This can be tested by including other data in the prediction and compare those predictions to the predictions made when including data from the cluster alone.

Another way to estimate future distributions is to use Bayesian statistics, where a reasonable prior is set for the distribution of the returns. However, it is arguably a weaker assumption to use cluster data to estimate the distribution than to assume a prior, which disqualifies a bayesian approach. The best is probably to take the best from both worlds, i.e. to base a prior on the cluster data and set the bayesian posterior as the return distribution. Such implementations are beyond the scope of this thesis.

6.5 UCR Time Series Archive

6.5.1 Plane

This test reveals a lot of interesting information about the data. First of all, it is easy to see that the thesis algorithm does a good job at finding the clusters and to separate them. Second, it is reasonable to think that the orange and the blue clusters depicted in the left plot in figure 5.7 are quite similar to each other. In fact, the orange and blue cluster correspond to class 1 and 2, which also happens to be the two classes that are most similar among all pair of classes within the data set. Thirdly, the 4 points in the top right of the plot seem to constitute their own cluster in this plot, but are in reality a part of another class. These points actually correspond to the shifted time series in class 5, seen in figure 4.2. This is probably a consequence of the choice of loss function in the autoencoder. The mean square error does not take shift variances into account, which makes the autoencoder blind to such changes. The reason that the little cluster in the top right corner is close to the rest of the true cluster is due to the heavy autocorrelation, which makes the shifted time series quite similar to the non-shifted series. However, it would be much better to shift the series before feeding it through a measure such mean square error. For instance, DTW presented in section 2.5.1 would be ideal in this situation (which is confirmed by its superior classification accuracy on this data set).

Table 5.6 shows the results of the consistency analysis of the output. For such a clear partitioning of the data, it is not satisfying see such low numbers on consistency. The autoencoder does a good job at separating the data into clusters, but the hierarchical GMM seems to be too sensitive to be reasonably stable. The reason is that the autoencoder gives vastly different output for every try, which is seen in figure 6.1. Although the human eye easily can see the similarity between the different outcomes (and even map clusters between the different plots, e.g. the cluster shapes seem to be consistent), it is not easy for a general clustering algorithm to do the same. However, it is reasonable to expect that normal clustering algorithms such as agglomerative hierarchical clustering or DBSCAN would do a better job at keeping a high output consistency.

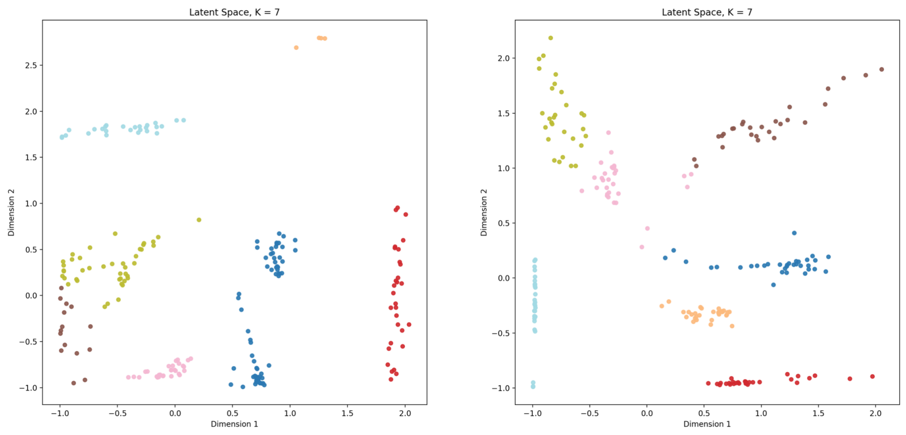

Figure 6.1: *Two different outcomes from the thesis algorithm are plotted. The clusters are placed at very different locations between the two outcomes, even though the shapes are recognisable. For example, the blue and the red cluster in the left plot probably corresponds to the blue, orange and red cluster in the red plot. It is also seen that class 1 and 2 are close to each other in both plots (the yellow/brown cluster in the left plot and the yellow/pink cluster in the right plot). The blue cluster in the left plot is an unfortunate merge of two separate cluster that does not exist in the right plot. Such events makes the output consistency worse than what should be expected.*

6.5.2 Fish

The autoencoder fails to cluster this data set properly. There are a few structures that are visible in the latent space such as the pink and red cluster. The rest end up in the middle without being classified correctly at all. The reason is probably that the time series look more similar than dissimilar, i.e. the differences are relatively subtle and hard to detect for the autoencoder. Instead of finding crucial differences, it gets stuck on the large similarities. On a data set like this one, there is a clear need for preprocessing the data and removing similarities and highlighting dissimilarities.

6.5.3 Gun Point

The Gun Point data set form three big clusters with a few sub cluster structures as well. When looking at the time series in figure 4.4, it is reasonable to expect to find more than 2 clusters in the data. It is even reasonable to argue that a good clustering algorithm should not divide the data into 2 separable clusters. Both classes contain series with a small but high peak, which should be expected to be clustered together. Looking at the left plot of figure 5.10, it is seen that many time series are correctly clustered while some are overlapping, as expected. Over all, the algorithm seems to find true relations and features without overfitting the data.

6.5.4 Italy Power Demand

The data seem to be clustered into somewhat separable clusters. When looking at the true classes in figure 5.12, it is tempting to draw a tilted horizontal line between the two colourings. Even tough not allowed nor encouraged by the algorithm, it is interesting to note that there is a trivial partitioning of the latent space that would separate the two classes, indicating that the VAE actually detects important structures.

The algorithm reveals other structures as well. A clustering with 5 or 7 clusters seem to be appropriate and is supported by the log likelihood graph and the eye. By looking at figure 4.5 it is reasonable to expect the algorithm to find more than 2 types of structures in the data.

6.6 Conclusions

Chapter 5 offers many interesting and promising results. Figure 5.3 indicate that cluster analysis in latent space is not only motivated in theory but also in practice, and figure 5.6 shows that meaningful clusters can be obtained through this method. The variational middle layer together with the loss function presented in section 3.1.3 is doing a great job at clustering the data points, given that the network behaves properly and the data is nice enough. The latent space offers a superior visualisations over traditional clustering algorithms due to its inherent low dimensionality. To be able to communicate an intuitive understanding of a problem is crucial in todays professional world and the variational autoencoder is in that sense not modern only due to its young age.

The VAE fails to recognise several types of times series and the data often need heavy preprocessing in order to be clustered properly. This is illustrated by the test on the Fish data set, which fails to do anything meaningful due to the lack of dissimilarity in the data set. Another example is the Plane data set where the thesis algorithm fails to recognise shifted time series as belonging to the same class. These kinds of errors are mainly due to the use of the mean square error loss function. It is reasonable to believe that a more general reconstruction loss function would make the algorithm run much smoother. However, in a financial setting where autocorrelation is usually absent or very small and differences are highlighted, the mean square error can be expected to work better. This is also what the results from the tests indicate.

Metaphorically speaking, the conclusion is that the VAEs weakest point is its vision. Once it has a clear enough picture of the objects, it is able to distinguish between them and categorise them somewhat properly. To make the thesis algorithm better for general data sets, a reconstruction loss that imitate the DTW distance measure and some automatic preprocessing to high light differences in the data set are probably the appropriate directions to look to.

Although not being very stable, the agglomerative hierarchical Gaussian mixture model delivers powerful results without any tuning or extra preprocessing of the data (other than the dimensionality reduction from the VAE). Including a likelihood framework into a clustering algorithm is probably the most intuitive and easy way of dealing with the problem of finding the right number of clusters in the data. This can be done in a number of ways, and the hierarchical GMM presented in this thesis is one suggestion tailored for a specific purpose. The thesis algorithm is especially interesting in that it includes a soft border in the clustering, yielding an opportunity to predict unstable points and do other kinds of analysis regarding relationships within the data. This may help to improve robustness and detection of outliers.

In general, the algorithm shows promising results and if the points in the next section are implemented, great improvements on weak areas can be expected and on the results in general.

6.7 Future work

There are several areas that can be improved or be extended for better performance in more specific problems and in more general settings. The following list includes numerous examples of further work to be done.

- Implement an automatic preprocessing of the data to highlight differences and disregard any invariances presented in section 2.5.
- Stabilise the hierarchical GMM. Either by constraining the covariance matrix to ”normal” shapes or by further integrating the hierarchical GMM and the VAE to eliminate inconsistencies seen in section 6.3. Sampling the covariance matrix from an Inverse Wishart distribution is another reasonable strategy [13].
- Abolish the hierarchical structure and use other clustering techniques, e.g. DBSCAN. The DBSCAN algorithm has not been touched upon in this thesis, but it is definitely worth investigating. The main reason why is that the standard deviations used by the VAE offer a natural way of initialising the affinity matrix. Furthermore and in contrast to a GMM, DBSCAN can find more general, non-convex cluster shapes. However, due to the variational middle layer with its Gaussian sampling function, there are reasons to not expect grand, non-convex structures. Despite that, there may occur structures that are better suited for DBSCAN than for GMM. Using DBSCAN would eliminate the likelihood framework and effectively not answering the question of how many clusters there actually are in the data set.
- In general, the standard deviations from the variational middle layer can probably be better used than they have been in this thesis. The DBSCAN example above is one such application and the prior on the covariance matrix discussed in section 3.2.3 is another. A good mathematical model should exploit all information as much as possible and the standard deviations contain a lot of information.
- If the question of the true number of clusters is put aside, there are many improvements imaginable. The Deep Temporal Clustering (DTC) algorithm presented in section 2.5.2, forces the latent representations into prespecified clusters during training and obtains good results. Furthermore, it includes advanced network layers to better capture structures that the thesis algorithm fails to capture. One idea that captures the essence of DTC and is implementable in the VAE is to pretrain the latent space so that certain inputs are directed to a specific subspace of the latent space. This would be done by choosing a subset of data points and give them a coordinate in latent space far away from each other. Then, the latter part of the network would be trained to reconstruct the original data point from that coordinate and the first part of the network would be trained to map a data point onto that coordinate. This pretraining would enhance stability, make it easier to separate clusters and make the latent space even less chaotic than it currently is.
- The loss function defined by equation 3.2 can be extended to include a gravitational force. This would discourage overfitting and force ambiguous points to gravitate towards one cluster, making the thesis algorithm more stable and the visualisation clearer. Figure 5.3 has a lot of ”asteroids” i.e. single points

of unclear origin. It might be so that they actually are their own independent entity and should not be grouped with any of the other points. If so, their independent integrity (i.e. the reconstruction loss) should be stronger than the gravitational force. Thus, true individuals would remain in their solitary cluster but ”rebels without a cause” (i.e. points that drift away from the cluster for unclear reasons) would be brought back to their true relatives.

- Most of the tests can easily be improved if needed. For real life purposes, it would be advisable to engineer the estimations to better capture what is sought for. For example, the estimation of the future distribution in section 4.2.2, does not transform the returns in any way before feeding them through to the Gaussian kernel. One easy improvement is to scale the returns to better fit the fund at hand, i.e. use the sample variance of the fund and scale the other returns accordingly. There are several tricks that can be utilised for improvement here. However, the purpose of this thesis is to demonstrate the usage of clusters in financial applications and to test if they can improve results significantly. Optimal algorithms for predictions are left for further work.
- In section 6.3 the likelihood for each cluster member was discussed. This analysis can be further developed and used in applications. For example, when trying to generate the underlying process, it is reasonable to weigh outliers lighter than those members with high likelihood. Such weighting is not done here, but is easily implementable and expected to be useful in a range of situations.
- It is possible (and probably beneficial) to frame the clustering problem in bayesian language. Section 2.1.2 and 6.4.4 discussed such an approach briefly. It would also be possible to approach the parameter estimations in a GMM using a bayesian framework, described in [13] and discussed in section 3.2.3.
- The stopping criteria discussed in section 3.2.3 and the problem of K, discussed in section 2.1.2, should be investigated further to enable an automatic and reliable stopping criteria. The algorithm would gain a lot of legitimacy if this problem was solved and implemented. Possible starting points are discussed in section 3.2.3.

Bibliography

[1] Lopez de Prado, M. 2016. Building Diversified Portfolios that Outperform Out-of-Sample *Science*. May 24. $https://papers.ssrn.com/sol3/papers.cfm?abstract_id = 2708678$

[2] Geron A. 2017. *Hands on Machine Learning with Scikit Learn and Tensorflow* 1. edition. O'Reilly Media, Inc.

[3] Driver H., Kroeber A. 1932. Quantitative Expression of Cultural Relationships *University of California Publications in American Archaeology and Ethnology.* $http : //dpg.lib.berkeley.edu/webdb/anthpubs/search?all = \&volume = 31\&journal = 1\&item = 5$

[4] Cattell, R. 1943. The description of personality: Basic traits resolved into clusters. *Journal of Abnormal and Social Psychology.* $doi : 10.1037/h0054116$.

[5] Marti. G., Nielsen. F., Donnat. P., Andler. S. 2016. On clustering financial time series: a need for distances between dependent random variables. $https : //arxiv.org/abs/1603.07822$

[6] Paparrizos J, Gravano L. 2016. k-Shape: Efficient and Accurate Clustering of Time Series. *ACM SIGMOD Record, Vol. 45, Issue. 1.* March. $https : //dl.acm.org/citation.cfm?id = 2949758$

[7] Kaplansky I. 1972. *Set Theory and Metric Spaces.* Allyn and Bacon, Inc.

[8] Hastie T.,Tibshirani R., Friedman J. 2008. *The Elements of Statistical Learning.* 2. edition. Springer.

[9] Heller. K., Ghahramani. Z. 2005. Baysian Hierarchical Clustering. *ICML '05 Proceedings of the 22nd international conference on Machine learning.* August 7. $https : //dl.acm.org/citation.cfm?id = 1102389$.

[10] Sheather. S. 2009. *A Modern Approach to Regression with R.* 1. edition. Springer, New York, NY

[11] Jakobsson. A. 2015. *An Introduction to Time Series Modeling.* 2. edition. Studentlitteratur AB.

[12] Sai Madiraju N, M. Sadat S, Fisher D, Karimabadi H. 2018. Deep Temporal Clustering: Fully Unsupervised Learning of Time Domain Features. February 4. $https : //arxiv.org/abs/1802.01059$

[13] Murphy. K. 2012. *Machine Learning: A Probabilistic Perspective.* 1. edition. The MIT Press,

[14] Sokal R, Rohlf F. 1962. The Comparison of Dendrograms by Objective Methods *Taxon, Vol. 11, No. 2.* Feb. $https : //www.jstor.org/stable/1217208?seq = 1\#page\_scan\_tab\_contents$

[15] Hinton G., Salakhutdinov R. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science, New Series, Vol. 313, No. 5786.* July 28. $https : //doi.org/10.1126/science.1127647$

[16] Denuit. M., Dhaene. J., Goovaerts. M., Kaas. R. 2005. *Actuarial Theory for Dependent Risks.* 1. edition. Join Wiley & Sons, Ltd

[17] Goodfellow. I., Bengio. Y., Courville. A. 2016. *Deep Learning.* 1. edition. The MIT Press,

[18] Higgins. I., Matthey. L., Pal. A., Burgess. C., Glorot. X., Botvinick. M., Mohamed. S., Lerchner. A. 2017. β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *ICLR 2017 conference submission.* April 18. $https://openreview.net/forum?id = Sy2fzU9gl$

[19] Anh Dau. H., Keogh. E., Kamgar. K, Michael Yeh. C-C., Zhu. Y., Gharghabi. S., Ratanamahatana. C., Chen. Y., Hu B, Begum. N., Bagnall. A., Mueen. A., Batista. G. (2018). *The UCR Time Series Classification Archive.* $https://www.cs.ucr.edu/\%7Eeamonn/time\_series\_data\_2018/$.

[20] Blom. G., Enger. J., Englund. G., Grandell. J., Holst. L. 2005. *Sannolikhetsteori och statistikteori med tillampningar.* 5.8 edition. Studentlitteratur.

[21] Smith. G. 2014. *Standard Deviations* 1. edition. Duckworth Overlook.

Master's Theses in Mathematical Sciences 2019:E27
ISSN 1404-6342

LUTFMS-3367-2019

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/