Clustering and Anomaly Detection in Financial Trading Data

Erik Norlander

Master's thesis
2019:E35

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Master's Thesis

Handelsbanken Capital Markets

Lund Institute of Technology, Dept. of Mathematical Statistics

Clustering and Anomaly Detection in financial trading data

Using the Conditional Latent Space Variational Autoencoder

Author:	Erik Norlander	nat13eno@student.lu.se
Supervisors:	Alexandros Sopasakis Associate Professor Dept. of Mathematics Lund University	Richard Henricsson Senior Quantitative Risk Analyst Model Validation and Quantitative Analysis Handelsbanken Captial Markets

Submitted on June 12, 2019

Lunds universitet

Lunds Tekniska Högskola

Abstract

In this thesis we propose a new form of Variational Autoencoder called the *Conditional Latent Space Variational Autoencoder* or CL-VAE. By conditioning on a known label in a dataset we can decide what points are being mapped to what prior distribution. This makes the *latent space* more understandable and separates the classes further. It also subverts the tug-of-war effect between reconstruction loss and KL-divergence somewhat. This is because we're not trying to map all the data to one simple prior distribution, but rather giving every class its own.

With this method, we can customize the latent space for a specific task like clustering or anomaly detection. This means that we can send in any kind of data, be it numerical or categorical, and the points will be projected to some more easily understandable structure. This is a big advantage over other dimensionality reduction algorithms like PCA that only deals with continuous variables.

The method is applied to trading data from Handelsbanken Capital Markets, a Swedish investment bank. We show that it can be used in modeling the trading behavior of the traders at the bank by performing clustering and anomaly detection in the latent space. CL-VAE outperforms the regular VAE on all our metrics and seems to prepare the data for analysis in a straightforward and interpretable manner. We also discuss the issue of *unsupervised* anomaly detection at length and use a new form of metric for such problems called the EM-MV measure.

Finally, the result is a system that can be used in order to model trading behavior and perform clustering and anomaly detection on the transformed data. We have performed the analysis by conditioning on the traders but the model is not limited to that label. Instead, we can condition on counter parties, instruments, portfolios or any other label in the dataset.

Keywords: *Variational Autoencoder, Generative Models, Latent Space, Dimensionality Reduction, Unsupervised Learning, Anomaly Detection, Clustering, Gaussian Mixture Models, Isolation Forest.*

Sammanfattning

I detta examensarbete föreslår vi en ny typ av Variational Autoencoder kallad *Conditional Latent Space Variational Autoencoder* eller CL-VAE. Genom att betinga på en känd variabel i en datamängd kan man bestämma vilka punkter som ska anpassas till vilken betingad sannolikhet. Detta gör det *latenta rummet* mer förståeligt och ökar separationen av klasser. Metoden undviker också dragkampseffekten mellan återskapningsförlust och KL-divergens något. Detta är på grund av att vi inte försöker anpassa alla punkterna till en enkel distribution utan snarare ger varje klass sin egen.

Med den här metoden kan du skräddarsy det latenta rummer för ett specifikt ändamål så som klustring eller anomalitetsdetektering. Detta betyder att du kan skicka in vilken typ av data som helst. Vare sig den är numerisk eller kategorisk kommer punkterna att bli projicerade till en struktur som är lättare att förstå. Detta är en stor fördel över andra metoder för att reducera dimensioner som PCA som bara handskas med kontinuerliga värden.

Metoden appliceras på data från Handelsbanken Capital Markets som är en svensk investeringsbank. Vi visar att den kan användas för att modellera beteendet av handlarna på banken genom att genomföra klustring och anomalitetsdetektering i det latenta rummet. CL-VAE presterar bättre än den vanliga versionen av VAE med alla mått vi använder och verkar förbereda data för analys på ett enkelt och tolkningsbart sätt. Vi diskuterar också problemet med *oövervakad* anomalitetsdetektering grundligt och använder en ny typ av mått för sådana problem kallad EM-MV-måttet.

Till slut resulterar examensarbetet i ett system som kan användas för att modellera handlarbeteendet och genomföra klustring och anomalitetsdetektering på den transformerade datan. Vi har genomfört analysen genom att betinga på handlarna men modellen är inte begränsad till den variabeln. Istället skulle man kunna betinga på motpart, portfölj, instrument eller vilken annan variabel som helst i datamängden.

Nyckelord: *Variational Autoencoder, Generativa Modeller, Latenta Rum, Dimensionalitetsminskning, Oövervakad Inlärning, Anomalitetsdetektering, Klustring, Gaussiska blandningsmodeller, Isolation Forest.*

Acknowledgements

I would like to thank...

Associate Professor Alexandros Sopasakis, my supervisor at the Department of Mathematics of LTH, for excellent support and guidance.

Richard Henricsson, and the rest of the team at Model Validation and Quantitative Analysis at Handelsbanken Capital Markets for providing the data and commissioning the project.

My parents Kristina and Arne Norlander, for supporting me, and in times of doubt, for encouraging me to persist.

Friends that I made during my time at Lund University and LTH for 6 fantastic years.

Fender Musical Instruments and especially the FujiGen Gakki factory of Matsumoto, Japan, for producing my favorite electric guitar, a 1962 Telecaster Custom Reissue. It has helped me to get out of many ruts.

Lund, May 28 2019

Contents

Chapter 1

Introduction

Financial institutions are becoming increasingly data driven which brings with it new opportunities but also new challenges. One such challenge is data quality which is affecting decision making on all levels in a bank. The quality of the data is essential for making the right predictions and therefore, if the quality is faulty the results will suffer. A related challenge in trading data is detection of financial crimes which has become a major issue for financial institutions. Whether it being money laundering, insider trading or related crimes, it strongly undermines the trust and stability of the financial system. In this thesis, we will study a problem involving trading data but the methods can be applied to problems in data quality as well.

So how do we detect when something goes awry? Ideally, we would like a setup like in Figure 1.1; some filter that can tell us whether a trade is *normal* or *anomalous* and what *category* of trading it belongs to. In order to perform such a filtering we will use *clustering* and *anomaly detection*. Clustering tries to group similar trades together and dissimilar trades apart while anomaly detection tries to estimate what regions of a dataset is normal in order to contrast it with anomalous regions. To understand whether a trade is normal or anomalous has an immediate utility, while clustering might seem a little arbitrary at first. However, the categories found through clustering will help us further down the line with anomaly detection and will give us valuable insights into the data.

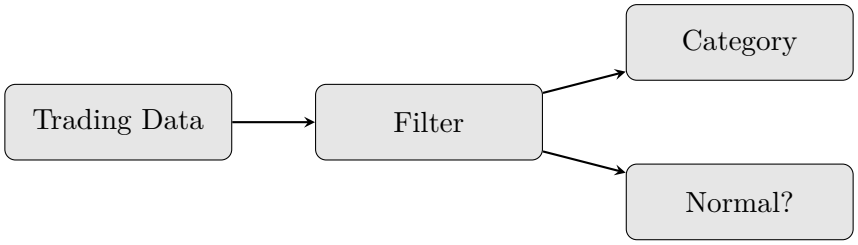


Figure 1.1: An idealization of a process that can classify a data point as normal or anomalous.

In this thesis we have analyzed a dataset of trades made by traders at the Handelsbanken Capital Markets, a Swedish investment bank. Each trade has information like trader name, portfolio, counter party, instrument, currency, nominal value, time of trade and many more. In total there's more than 300 000 trades and 27 columns that describe different traits of each trade. There are over 400 unique traders in this dataset that

trade in different markets using different instruments such as bonds, future/forwards, FX, swaps and many more. We would like to take this data and send it into some process as described in Figure 1.1. Meaning we want to try and extract information about what a normal trade looks like for a certain trader, what trades are anomalous and what category of trading each trader is partaking in.

However, before we can apply clustering and anomaly detection successfully on this dataset there are several issues that need to be addressed, including:

1. Class imbalance.
2. Class overlap.
3. Large amount of features, all being categorical.
4. The dataset is unlabeled, meaning that we *do not know* which data points are anomalies and which are normal.

Class imbalance means that some traders have more trades than others. This phenomenon is quite severe in this data as 120 000 trades are attributed to one trader, about a third of all the data. This is in stark contrast with most traders that have less than 5000 trades and the second most frequent trader having 20 000 trades. It is going to be difficult to model that trader with 120 000 trades with any of the others simply because its overpowering; details in other trading behaviour will be less noticeable in comparison. Meanwhile, we do not want to rely on traders that have too little data. Therefore, for most of this thesis we will limit ourselves to a reduced dataset of about 100 000 trades consisting of trades made by traders in the 5000 to 20 000 range, including eleven unique traders.

Class overlap could occur when two or more traders deal in similar markets with similar counter parties, similar currencies and so on. This in itself would actually help us create categories in the data, but it also means that trades that would be considered normal if one trader did them might be considered anomalous if another trader did them. For example, if a trader mainly dealing in bonds with government customers all of a sudden starts trading in future/forwards with private customers. An occurrence like that will be considered anomalous in this thesis.

We touched on clustering and anomaly detection earlier which are two methods that will help us perform our analysis. However, neither of these methods can work with *categorical* data which is what all our columns consist of. Therefore, some sort of transformation is needed which will at some point involve reducing the number of dimensions in the dataset. This is because many clustering and anomaly detection algorithms are distance-based, so we will have to evaluate some distance between trades. They are also heavily dependent on the *shape* of the data. Unfortunately, because we have categorical data, traditional algorithms for dimensionality reduction such as PCA [1] will not create any meaningful transformation as the data is non-continuous. Therefore, we will use a set of machine learning algorithms know as *autoencoders* for this task. They are algorithms that aim to represent a high dimensional dataset in some lower dimensional space called the *latent space*. Remarkably, an autoencoder can make non-linear transformations to the latent space and is not dependent on continuous input data.

All of the four aforementioned difficulties are quite serious but particularly difficult is the fourth issue that makes it apparent that we need to do some form of *unsupervised learning*. This is a setting in machine learning where you do not know your target and you can not check whether the result is true or not, or at least not in any straightforward

manner. Because of this we will have to define what we mean with anomalies which is inherently difficult, but nevertheless important. Without getting too mathematical (yet) we will try and find two types of anomalies:

1. *Trades that are not regarded as part of any category.*
2. *Trades that would be normal if made by one trader, but anomalous if made by another.*

The first type of anomaly may seem straightforward but really is not. Imagine trying to find a boundary between two adjacent sets of points (normal and anomalous) in some two-dimensional plane, but you don't actually know which on is which. It might be possible to do this but it is going to be impossible to validate. Meaning that any algorithm we throw at it will be hard to tune because we will not know whether we're getting closer or further away from the optimal solution. This will be discussed using some new developments in the field of unsupervised anomaly detection in Section 2.7. The second type of anomaly is relying on our ability to classify trades as part of a category (clustering). If one trader has been classified as part of some category or a number of categories and trades are starting to come in that are not part of any of the previous categories we will try and identify them as anomalies.

Preferably we would like to estimate some distribution of each trader and from that find statistical anomalies. This led us down the road of *generative* models and especially *Variational Autoencoders* [2]. As the name suggests they are autoencoders with a statistical spin which gives us the possibility of estimating underlying probability distributions. This makes them both theoretically satisfying and useful in practice for statistical applications. Often they are used for generating new data that looks like it was sampled from a real dataset. However our primary motivation for using them is that they create a continuous latent space, which is not the case for the regular autoencoder [7]. Hopefully, this will transform our data such that it is shaped in a helpful manner for performing clustering and anomaly detection. We will refer to such a transformation as *pre-processing*. Furthermore, we will try and improve upon the existing VAE in order to make it more suitable to out needs by introducing the *Conditional Latent Space Variational Autoencoder* or CL-VAE in Section 3.2. Aiming to make the latent space more interpretable and applicable with existing algorithms.

Finally, in this thesis we will try and show results that can be used to create a system that does clustering and anomaly detection automatically by using some form of VAE to prepare the data. We do not propose such a system directly, but rather aim to show that it would be possible if developed further.

1.1 Problem Setting

In this thesis we aim to answer the following three questions:

1. ***Can Variational Autoencoders be used to pre-process the data for clustering and anomaly detection?***
2. ***What kind of Variational Autoencoder is most successful at this job?***
3. ***Can a system be developed using this method to perform clustering and anomaly detection automatically on financial trading data?***

1.2 Technologies

Recently many developments in the field of artificial intelligence have been made. This is largely thanks to a few extremely large companies on the west coast of the United States that have made these methods widely available. But also thanks to the flexibility of existing programming languages to accept such models and a vibrant open source community.

Python

Python is a general purpose language that have become ubiquitous for machine learning applications. It is very useful because of its simplicity, flexibility and large community. There are also many open source packages for Python that makes our life a lot easier. For example Pandas which makes managing data very easy and Jupyter Notebook which makes it possible to run all our code in a browser.

Many popular algorithms are included in the package SciKit-Learn for Python. These include some algorithms that will be discussed later such as Gaussian Mixture Models and Isolation Forest. Moreover, it also includes great tools for measuring performance of machine learning algorithms and some tools for pre-processing data. Because of the simplicity of this package it has been used throughout this thesis.

TensorFlow & Keras

TensorFlow is a library, developed by Google, to bring modeling with neural networks to the public. It is a very powerful API that lets us build any kind of neural network. However, it is quite limited in user friendliness. Therefore, Keras was invented by Francois Chollet in 2015 [7]. Keras is a high-level API that can run with TensorFlow backend. This means that we can build these models much easier than before. Since TensorFlow 2.0, Keras is the definite high-level API for TensorFlow, now included in the installation.

Chapter 2

Theoretical Background

2.1 The basics of Machine Learning

Machine learning is a widely discussed topic, but it is also largely misunderstood. Therefore,we will offer a brief overview before we delve further into the topic. Machine learning can be divided into many classes, but for the purpose of this thesis we will discuss two:

1. *Supervised Learning* has a limiting pre-requisite which is *labeled data*. This means that you must know from the data what class an instance belongs to. A simple version of this is a classifier between cats and dogs. If you feed an algorithm pictures of cats and dogs and also tell it which is which you're doing supervised learning. This is an example of a binary classifier which can be extended into a multi-class classifier if you would like to tell the difference between cats, dogs and horses for instance.

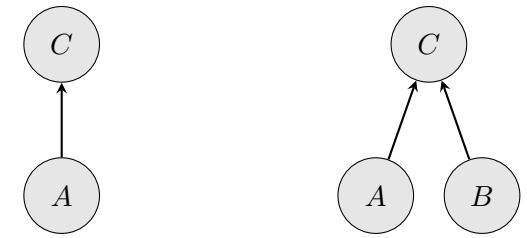
2. *Unsupervised Learning* is not limited to labeled data, but is conceptually a harder problem to solve. If you feed an algorithm pictures of cats and dogs without specifying which is which, you are doing unsupervised learning. In that case, the algorithm has to find features of the animals that makes it possible to divide them in to two classes. However, it is not at all certain that these classes will be "cat" and "dog". Therefore, such problems often require many more considerations than supervised problems and can prove tricky to train. Solving this problem was called "the holy grail of machine learning" by Yann LeCun, head of AI research at Facebook.

Many different algorithms are either supervised or unsupervised. Typically, classification is a supervised problem while clustering is an unsupervised problem. However, whether a problem is supervised or not is more a question of setting than algorithm, because some algorithms can work in both settings.

2.2 Artificial Neural Networks

Technology is often inspired by nature and Artifical Neural Networks (ANN) is no exception. Naturally, to build intelligent machines we would look into how our own brain learns for inspiration. That is exactly what Warren McCulloch and Walter Pitts did in

1943 when they proposed the *artificial neuron* [3]. The basic idea of this is to solve logical operations by modeling them as a network of graphs. An artificial neuron as seen in Figure 2.1 activates its output when more than a certain amount of inputs are active and such a simple setup was shown to solve any logical proposition [4].



(a) $C = A$ (b) $C = A \wedge B$

Figure 2.1: Examples of different logical operations possible to represent in an artifical neuron.

The next step in the evolution of ANN was *the perceptron* invented by Frank Rosenblatt in 1958 [5]. This introduced the concept of *weights* to the world of ANNs which made it possible to *train* a neural network. This also expanded their usage beyond binary on/off operations to all numeric values. A perceptron consist of a layer of what Rosenblatt called Linear Threshold Units (LTUs). Essentially, an LTU (Definiton 1, Figure 2.2) computes a weighted sum of its inputs and then applies a step function to that sum and outputs the result.

Definition 1 *The Linear Threshold Unit. Given a set of inputs x_i and weights w_i where $i = 1, \ldots, n$, compute the output as a step function θ of the weighted sum of inputs x_i.*

$$h_w(\bar{x}) = \theta\left(\sum_{i=1}^{n} w_i x_i\right)$$

A common step function is the the Heaviside: $\theta(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$.

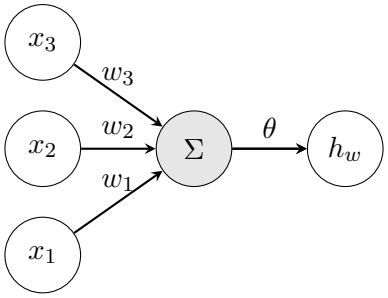


Figure 2.2: An example of a Linear Threshold Unit, here x_i are inputs, w_i are weights, Σ represents summation and θ the heaviside step function.

A simple LTU can perform binary classification, but if we would like to create a more complex model we would pass the weighted inputs to more LTUs into a perceptron which is described in Figure 2.3.

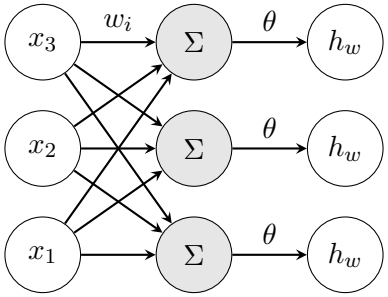


Figure 2.3: An example of a perceptron, here x_i are inputs, w_i are weights, Σ represents summation and θ the heaviside step function.

In order to train the perceptron, Rosenblatt found inspiration in biological learning just as he's predecessors McCulloch and Walter did. In 1949 Donald Hebb suggested that the connection between a biological neuron that triggered another biological neuron grew stronger. This became known as Hebbian Learning which for the perceptron meant that the weight between two artificial neurons is increased whenever they have the same output. [4]

Definition 2 *The Perceptron Learning Rule. The perceptron is fed one training instance at a time and makes a prediction. For every wrong output the input that would have made a correct prediction is increased.*

$$w_{i,j}^{n+1} = w_{i,j}^{n} + \eta(y_j - \hat{y}_j)x_i$$

In this case $w_{i,j}$ describes the weight from the ith input neuron to the jth output neuron, n is the time step, η is the learning rate (a constant), y_i is the target output, $\hat{y}_i$ is the prediction and x_i is the input to neuron i.

However, the decision boundary of the output neuron from a perceptron is linear, so it is uncapable of learning complex patterns. It also does not output a class probability but is rather based on a hard threshold which is a limitation even in comparison with a simple algorithm like Logistic Regression. Therefore, the Multi-Layer Perceptron (MLP) was introduced.

The MLP consist of stacked layers of perceptrons that are called hidden and output layers respectively. This model proved capable of solving some of the issues of perceptrons but it also had limitations. It proved much harder to train than the regular perceptron. But in 1986 the back-propagation algorithm was introduced to the MLP in the groundbreaking paper "Learning Representations by Back-Propagating Errors" by Rumelhart, Hinton and Williams [6]. The idea for this algorithm is to try and find the gradient through the network by optimizing a loss function.

Back-propagation was known earlier as a general optimization algorithm but Rumelhart, Hinton and Williams discovered that it could be used for MLPs to find good "internal representations". This was quite revolutionary as researchers spent a lot of time engineering features in order to train neural networks. A good feature is a feature that a neural network can easily learn and usually are simple shapes like lines, circles etc. This requires a lot of domain knowledge and is a slow and tedious process, still used in some other machine learning cases.

Today, it is somewhat unhelpful to call these models Artificial Neural Networks as they have become quite different from their biological counterparts. This seems to lead to

unnecessary confusion. Nevertheless, the analogy has a rich history, it remains in the narrative about machine learning and is likely to do so in the future as well.

2.3 Autoencoders

An autoencoder is an artificial neural network that finds a lower-level representation of higher dimensional data. Therefore, it is essentially performing the job of dimensionality reduction. But it also helps us getting hidden classes in our data, which solves the unsupervised learning problem in some instances. Autoencoders are especially interesting because they tell us some very interesting things about what kind of jobs neural networks actually does for us.

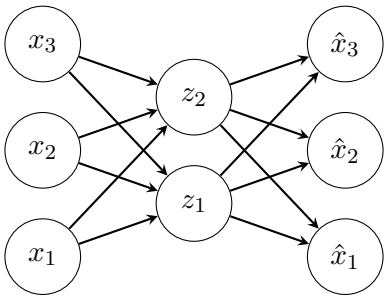


Figure 2.4: An example of an autoencoder. Here x_i is the input, z_i the hidden representation with 2 latent dimensions and $\hat{x}_i$ the reconstruction of the input x_i.

Assume we have a picture that we want to represent as a point in two dimensional space. By using an autoencoder this would be done by setting up 2 neural networks - an encoder and a decoder. In the most simple version the encoder is a densely connected layer with input size equal to the input x and output dimensions 2. Likewise, the decoder is a densely connected layer with input size 2 and output size equal to that of the input.

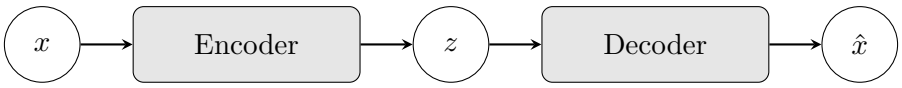


Figure 2.5: A flow chart of the autoencoder. Here the white circles represent input or output data and the grey rectangles represent neural networks.

Definition 3 *Autoencoder. Given an input $x \in \mathbb{R}^d$ we assume that there is a mapping E to $z \in \mathbb{R}^p$ s.t. $E : x \to z$ This mapping is called the encoder and can be defined with an activation function σ, a weight matrix W and a bias term b as*

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

Conversely, we assume that there is a mapping D s.t. $D : z \to x$ which is called the decoder and can be defined in a similar way to the encoder with the corresponding terms $\hat{\sigma}$, $\hat{W}$ and $\hat{b}$ as

$$\hat{x} = \hat{\sigma}(\hat{W}z + \hat{b})$$

What this type of setup is actually doing is finding some non-linear transformation from x to z and its inverse, which becomes clear from Definition 3. In order to train a network

to estimate these transformations we need a loss function which we find by minimizing the reconstruction error of x which can be found in equation (2.1). From this we have all necessary components to train such a network.

$$L(x, \hat{x}) = ||x - \hat{x}||^2 = ||x - \hat{\sigma}(\hat{W}z + \hat{b})||^2 = ||x - \hat{\sigma}(\hat{W}(\sigma(Wx + b)) + \hat{b})||^2 \tag{2.1}$$

Now, conceptually it is not hard to imagine that you can use this kind of setup for categorizing data in an unsupervised way. It seems reasonable that the latent representation z can group some inputs together and make a kind of clustering from high dimensional data. This was the initial idea that made us think of using autoencoders to cluster trades that have many dimensions and are hard to categorize otherwise. As it turns out, classic autoencoders don't produce very useful or nicely structured latent spaces because the latent space is *non-continuous* [7]. This means that two trades that *should* be close to each other does not necessarily end up that way. Because we are only training on reconstructing the input some trade will just be mapped to a single point in the latent space and taking a small step in any direction will likely just produce nonsense. Therefore, we'd like to find some method that creates a *continuous* latent space, introducing *Variational Autoencoders*.

2.4 Variational Autoencoders

When we are not trying to find a function but rather a distribution of some underlying process we can use a Variational Autoencoder (VAE) initially described by Kingma and Welling in their 2014 paper "Auto-encoding Variational Bayes" [2]. People most often use a VAE to generate new images from a known dataset of images but it has far reaching implications if properly understood [8]. In fact, a VAE solves the problem of density estimation and is a true generative model, meaning you can generate new samples from *any* unknown distribution [9].

Taking our observed values x (trades) that we assume follow some unknown stochastic process X that we want to sample from, we can use some prior z that we assume to be Gaussian. By then taking the expectation of the conditional distribution of x given z, under z, we get the distribution for x by equation (2.2).

$$p_\theta(x) = \mathbb{E}_z\Big[p_\theta(x|z)\Big] = \int p_\theta(x|z)p_\theta(z)dz \tag{2.2}$$

Definition 4 *Bayes Rule. Here x and y are two stochastic variables with density functions $p(x)$ and $p(y)$. Knowing the conditional density function $p(x|y)$ can give us the conditional density function $p(y|x)$ by*

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}.$$

If we think of this as an autoencoder, we can think of the conditional distribution $p_\theta(x|z)$ as a probabilistic decoder from z to x. However, it is not trivial to compute $p_\theta(x)$ as the integral in equation (2.2) is intractable [2], meaning it can not be solved for all z. Using Bayes Rule from definition 4 to try and compute the posterior instead we get

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} \tag{2.3}$$

This is also intractable because it has the troublesome distribution $p_\theta(x)$ in the denominator. To get any further we need to estimate the posterior distribution $p_\theta(z|x)$. Therefore we want to estimate it by some distribution $q_\phi(z|x)$ where ϕ are estimations of the model parameters.

As we learned from the previous section about regular autoencoders, a neural network is suitable for such a task. But a neural network needs some loss function to be able to train. We are in the unfortunate position were we can't optimize directly because of the intractability of (2.2). This means that we have to define and optimize a lower bound of the likelihood instead.

Assuming that z now can be estimated using the estimated distribution $q_\phi(z|x)$ we can take the logarithm of the expectation in (2.2) and get the log-probability.

$$\log p_\theta(x) = \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x)\right] = \tag{2.4}$$

$$= \mathbb{E}_z\left[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\right] = \qquad \text{(Bayes Rule)} \tag{2.5}$$

$$= \mathbb{E}_z\left[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\frac{q_\phi(z|x)}{q_\phi(z|x)}\right] = \tag{2.6}$$

$$= \mathbb{E}_z\left[\log p_\theta(x|z)\right] - \mathbb{E}_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \qquad \text{(log rules)} \tag{2.7}$$

To get any further in our quest for the lower bound we must now introduce the concept of entropy in information theory and KL-divergence as defined below. KL-divergence has some important properties (Properties 1 and 2) which we will exploit later.

Definition 5 *Entropy in information theory. Given two probability distributions Q and P the entropy of P is given by*

$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$$

and the cross-entropy of Q and P is given by

$$H(P, Q) = \mathbb{E}_{x\sim P}[-\log Q(x)]$$

Definition 6 *Kullback-Leibler Divergence. Given two probability distributions Q and P the Kullback-Leibler Divergence measures how well Q approximates P by taking the cross-entropy minus the entropy.*

$$D_{KL}(P||Q) = H(P, Q) - H(P)$$

Simplifying by using definition 5 and log rules we get

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right]$$

Property 1 *Properties of KL-divergence [10].*

1. *if* $P = Q$ *then* $D_{KL}(P||Q) = 0$
2. *if* $P \neq Q$ *then* $D_{KL}(P||Q) > 0$

Property 2 *Solution to* $D_{KL}(q(x)||p(x))$ *in the multivariate Gaussian case [2]. Let's assume that* x *is some random variable in* $q(x) \sim N(\mu, \sigma)$ *and* $p(x) \sim N(0, I)$

$$D_{KL}(q(x)||p(x)) = \mathbb{E}_{z\sim q_\phi(z)}\left[\log \frac{q(x)}{p(x)}\right] = \int \log \frac{q(x)}{p(x)} q(x)dx =$$
$$= \int (\log q(x) - \log p(x))q(x)dx$$

This can now be evaluated as two different integrals. The first being

$$\int q(x) \log q(x)dx = \int N(x; \mu, \sigma^2) \log N(x; \mu, \sigma^2)dx =$$
$$= \frac{J}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^{J}(1 + \log \sigma_j^2)$$

where J *is the number of samples. The second being*

$$-\int q(x) \log p(x)dx = -\int N(x; \mu, \sigma^2) \log N(x; 0, I)dx =$$
$$= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(\mu_j^2 + \sigma_j^2)$$

giving us the closed form solution

$$D_{KL}(q(x)||p(x)) = -\frac{1}{2}\sum_{j=1}^{J}\left[1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2\right]$$

Now, with all that math out of the way we continue from (2.7) and get

$$\log p_\theta(x) = \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big]}_{\text{Reparameterization trick}} - \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Closed form}} + \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z|x))}_{\text{Intractable, but} \geq 0} \quad (2.8)$$

The first term in (2.8) can be computed with the decoder network and sampling with a reparameterization trick, that will be discussed later. The second term is a KL-divergence between a general and a normal Gaussians which has a nice, closed form solution shown in Property 2. Finally, the last KL-term is intractable because we don't have $p_\theta(z|x)$, but by Property 1 we know that it is always greater or equal to 0.

Now, from equation (2.8) we can get a tractable lower bound (conveniently called ELBO for evidence lower bound [11]). ELBO is a lower bound and not a pure optimization because of the final term in (2.8).

$$L(\theta, \phi; x) = \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - D_{KL}(q_\phi(z|x)||p_\theta(z)) \quad (2.9)$$

Great, but we are not done yet. In order to estimate our parameters θ and ϕ we have to minimize the ELBO (2.9) with respect to these parameters. Therefore, we meet a problem as the first term of (2.9) is stochastic.

$$\phi^*, \theta^* = \underset{\phi,\theta}{\operatorname{argmin}} L(\theta, \phi; x) \tag{2.10}$$

This means that we can't optimize it right away. Instead we perform the reparameterization trick.

The Reparameterization Trick

In fact most of the math up until this point was already established before the paper by Kingma and Welling in 2014. As early as 1995 the Helmholtz machine was described by Hinton et al [12] but the problem was that it is very hard to optimize the ELBO with respect to the model parameters ϕ and θ and led to estimates with high variance [13]. The key insight that Kingma and Welling introduced was the reparmeterization trick which is a straightforward change of variables.

Note that the random variable $z \sim q_\phi(z|x)$ can be written as a deterministic transformation g_ϕ as a function of another random variable ϵ and input x with parameters ϕ [11].

$$z = g_\phi(x, \epsilon), \quad \epsilon \sim p(\epsilon)$$

Where $p(\epsilon)$ is a simpler base distribution which does not depend on x or ϕ. We can now write the ELBO as an expectation of $f(x, z) = \log p_\theta(x, z) - \log q_\phi(z|x)$ over the distribution $q_\phi(z|x)$. If we then substitute z with $g_\phi(x,\epsilon)$ and take the gradient of ϕ we get

$$\begin{aligned}\nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}[f(x,z)] &= \nabla_\phi \mathbb{E}_{g_\phi(x,\epsilon)\sim p(\epsilon)}[f(x, g_\phi(x,\epsilon))] = \\ &= \mathbb{E}_{g_\phi(x,\epsilon)\sim p(\epsilon)}[\nabla_\phi f(x, g_\phi(x,\epsilon))].\end{aligned}$$

And now we are home, as the expectation of the gradient and the gradient of the expectation commute we can optimize this in a regular fashion with back-propagation! Since we want the latent variables to be Gaussian we set $p(\epsilon) = N(0,1)$ and z becomes

$$z = g_\phi(x,\epsilon) = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim N(0, 1)$$

VAE as a model

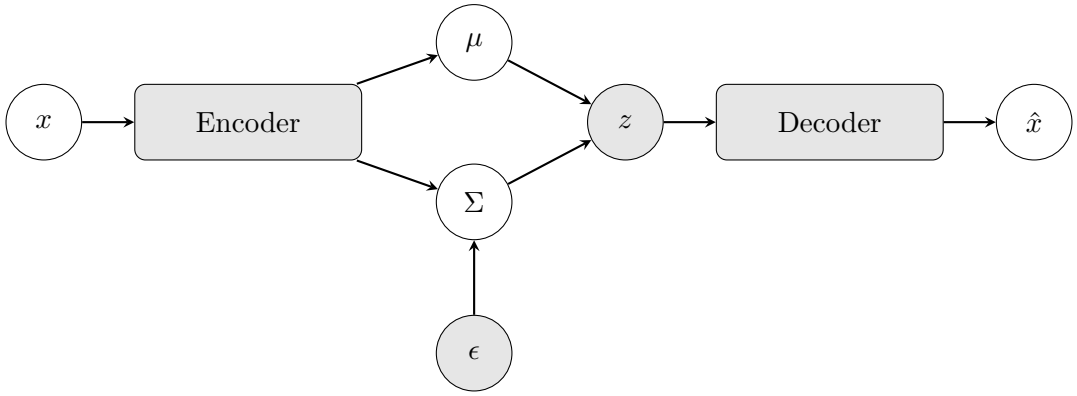


Figure 2.6: A flow chart describing the VAE. The Encoder is a neural network that maps the input x to the latent space described by z. This is done by mapping the mean and variance of these points with the stochastic layer described by ϵ, μ and Σ. Finally, the Decoder is another neural network that takes us back to the input space and creates a reconstruction of the input in $\hat{x}$.

So, putting all these concepts together we arrive at the model in Figure 2.6. This model is very similar to the original autoencoder in Figure 2.4 except for the latent space. In this case, the latent space is sampled by estimating μ and Σ. One interesting feature of the latent space of a VAE is that it is continuously meaningful. This means that, since every point has a variance, the latent space representation is forced to have feasible features throughout the latent space. This is unlike a regular autoencoder that often simply maps one numeric value in the latent space without a continuous meaningful latent space [7].

While some deep learning models benefit massively from stacking, VAEs have been shown to not find any further representational power through the process [14]. Therefore, if we're not happy with the result in the latent space we'll have to change something else in the model.

Intuition on the loss

I now want to offer some intuition on the ELBO loss function in equation (2.9) to help us in our reasoning further down the line.

$$L(\theta, \phi; x) = \underbrace{\mathbb{E}_{z \sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big]}_{\text{Reconstruction objective}} - \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regulizer to } p_\theta(z)}$$

The first term $\mathbb{E}_{z \sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big]$ can be interpreted as the reconstruction objective. This is trying to make the reconstruction of samples better. While the second term $D_{KL}(q_\phi(z|x)||p_\theta(z))$ is trying to regularize $q_\phi(z|x)$ to be close to the prior $p_\theta(z)$. A good reconstruction is reached when samples are easy to tell apart and underlying classes are not construed. A good regularization is achieved if we look in the latent space and see that all points have been mapped to the same Gaussian. This was brilliantly described

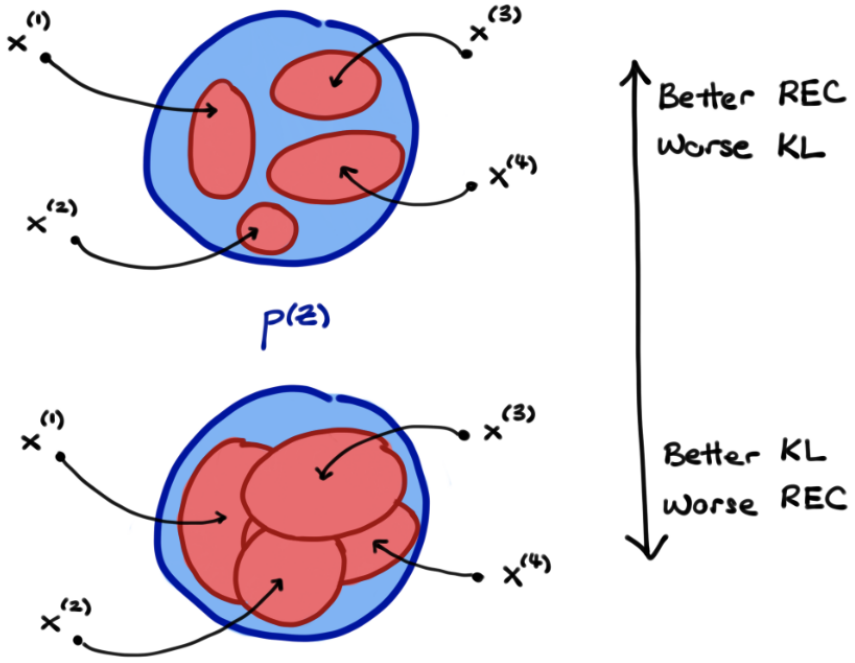


Figure 2.7: An idealization of the tug-of-war effect between reconstruction loss and regularization to the prior distribution $p_\theta(z)$ (described as $p(z)$ in the figure). The reconstruction loss will try and force the classes apart as it makes the latent space less ambiguous, meaning less overlapping classes. The KL-divergence on the other hand is trying to force the classes on top of each other close to the origin as it will make them better suit the prior distribution $p_\theta(z)$ [15].

by R. Shu [15] that also offered a great idealization of the problem in Figure 2.7. In this case four samples are being stochastically encoded into the space of the latent variable z. REC denotes the reconstruction cost, while KL denotes the KL-divergence to the prior. When there is little overlap over the stochastic embeddings, the reconstruction cost is low but the KL-divergence is high. When the KL-divergence is low, there is too much overlap between the embeddings and thus reconstruction cost is high.

Thinking a little further on this, it becomes clear that the extremes at each end of Figure 2.7 represent two different philosophies of finding anomalies. If we have great reconstructive ability, then we can measure the reconstruction loss in order to find some subset that is particularly badly reconstructed. If we assume that most of the data in our dataset is considered "normal", then anomalous points will have a bad reconstruction loss. This seems to be the mainstream way of doing anomaly detection with autoencoders [33] [34] more on this later.

On the other end of the spectrum, if we have all our data closely mapped to a Gaussian, then we can use all nice properties of a Gaussian in order to find anomalies. These could include simply taking the p percentile of the distribution and saying that it is anomalous or using more sophisticated techniques such as Isolation Forest to find anomalies.

$$L(\theta, \phi; x) = \gamma \mathbb{E}_z \Big[\log p_\theta(x|z) \Big] - \beta D_{KL}(q_\phi(z|x)||p_\theta(z)) \tag{2.11}$$

This tug-of-war effect with the VAE is central to this thesis and we will discuss it extensively. To make this easier, we will introduce 2 constants γ and β to the ELBO loss function (2.11) that acts as weights. This makes it possible to manipulate it to our desire. Perhaps we can take this one step further and give each class their own distribution?

Why not use a GAN?

The paper that introduced VAEs was ground-breaking but did not get as much attention as Goodfellows "Generative Adverserial Nets" [16] which was published the same year. At least not in the public eye. This was probably because VAEs produced less sharp images than GANs when trained for image generation (check out https://thispersondoesnotexist.com for a cool application of GANs).

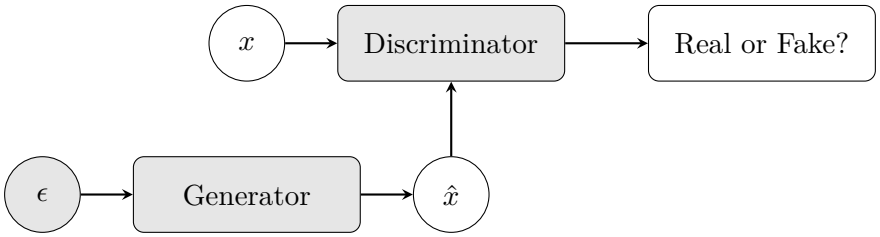


Figure 2.8: A flow chart describing the architecture of a GAN. The Generator is a neural network that takes us from random noise ϵ to a reconstruction (fake) of the input x, here denoted as $\hat{x}$. The Discriminator is another neural network that is trained to tell the difference between the real or fake data.

The technologies are similar in the way that they are both generative models, capable of sampling from an unknown distribution to create samples that look real, but also fundamentally different. A GAN, described in Figure 2.8, takes a game theoretic approach to the problem, trying to reach the Nash equilibrium of one network generating images and another trying to tell the difference between real and fake [16]. The VAE on the other hand is derived from a purely statistical point of view and the fact that you can use if for generation is almost just a bonus [17].

There are methods that try to combine the two like the amazingly named VAE-GAN (pronounced 'vegan') proposed by Larsen et al. that is basically a VAE with a GAN attached to the decoder [18]. There's also Adverserial Autoencoders described by Goodfellow et al. [19] that impose a GAN to the latent space of a VAE, forcing it to conform to some pre-determined distribution.

However, when training a GAN most people will experience at least one of the following problems [20]:

1. Non-convergence. The model parameters oscillate, destabilize and never converge.
2. Mode collapse. The generator collapses which produces limited varieties of samples.
3. Diminished gradient. The discriminator gets too successful that the generator gradient vanishes and learns nothing.

4. Unbalance between the generator and discriminator causing overfitting.
5. Highly sensitive to the hyper parameter selections.

Therefore, because we wanted some form of dimensionality reduction with statistical properties rather than a purely generative model the choice to use a VAE was rather obvious. Also, generation of images are often more blurred with a VAE because of the statistical nature of the latent space. This is not necessarily an issue when working the categorical data that will be hard drawn anyway. However, a VAE is not without its limitations, which will be discussed at length in this thesis.

2.5 Clustering

We want to cluster the data into a few different trading categories in order to get a better understanding of the data. As was explained in the first section of the thesis, clustering is the task of grouping data together that have similar properties and separating data apart that have dissimilar properties. It is an unsupervised task, meaning in this case that we do not know the outcome before hand and cannot check whether we were right or wrong. Therefore, there will be a section on measuring the quality of clusters later in the thesis.

k-means

The aim of k-means clustering is to partition a dataset up into k clusters in such a way that a loss function is minimized. This is done by finding the means of these clusters and classify points as belonging to a cluster based on the Euclidean distance to its mean. If a point is categorized as part of the cluster, the mean is updated and the algorithm is repeated [21]. A popular way of performing k-means clustering is called Lloyd's algorithm (Algorithm 1). It works by alternating between two steps, assignment and update as seen in the algorithm below. k-means is likely the most common clustering algorithm, and is often the first one you learn in an introductory machine learning course.

Algorithm 1: Lloyd's algorithm (k-means)

Data: Your dataset, number of clusters k
Result: Cluster label for each point
Randomly initialize a set of k means $\{m_1^1, \ldots, m_k^1\}$
while *Assignment* $S_i^t \neq S_i^{t-1}$ **do**
  for *Every point* x_i *in the dataset* **do**
    $S_i^t = \{x_p : ||x_p - m_i^t||^2 \leq ||x_p - m_j^t||^2 \quad \forall j, 1 \leq j \leq k\}$
    $m_i^t = |S_i^t|^{-1} \sum_{x_j \in S_i^t} x_j$
  end
end

As is evident from Algorithm 1, we have to assign the number of clusters. This poses a problem, as we do not necessarily know the perfect number of clusters before hand. Therefore, this is a hyper-parameter that has to be tuned for successful clustering.

GMM and EM

Gaussian Mixture Models (GMM) are probabilistic models that assume that your data is a mixture of Gaussian distributions. Conceptually, these are very applicable to clustering the latent space of a VAE. This is because we're forcing it to be Gaussian through training with the KL-divergence defined in the previous section as loss. For generality, we define GMM for the N-dimensional case in Definition 7.

Definition 7 *Gaussian Mixture Model in N dimensions. Here, $p(x)$ is the probability we wish to estimate, ϕ_i is a normalization factor s.t. $\sum_{i=1}^{N} \phi_i = 1$ and $N(x; \mu_i, \Sigma_i)$ are N-dimensional Gaussian distributions with mean μ_i and variance Σ_i.*

$$p(x) = \sum_{i=1}^{N} \phi_i N(x; \mu_i, \Sigma_i)$$

Definition 7 makes it clear that we are trying to find a sum of distributions that are Gaussian distributed with unknown mean and variance. One should suspect that this type of model can be applied if the data seems to be multimodal.

The Expectation Maximization (EM) algorithm is a way to estimate these Gaussian distributions and was formally described in 1977 by Dempster et al. in a now classic paper [22]. It works by alternating between two different steps until convergence like Lloyd's algorithm presented earlier.

The first is the expectation (E) step, which estimates the function for the expectation of the log-likelihood of the component assignment C_k for each data point $x_i \in X$ given the model parameters $\theta_k = [\phi_k, \mu_k, \sigma_k]$. The second step is the maximization (M) step, which maximizes the parameters of the expectation estimated in the E-step. The goal being to update the values of the parameters $\theta_k = [\phi_k, \mu_k, \sigma_k]$ [23].

As the reader might realize, this is a bit of a chicken-and-egg problem. We need the expectation to perform maximization and we need parameters to get the expectation and so on. Therefore, much like the beloved Lloyd's algorithm (k-means), EM randomly assigns the parameters in θ_k as an initialization step.

After this initialization each set of parameters is tested against each point in the data set by computing the probability that a point is part of that distribution. This performs "soft-clustering" on the data, meaning that we don't have hard borders between clusters but rather a probability of each point belonging to a certain cluster. If a point is considered to be part of a cluster, the parameters are updated accordingly. There's a great animation on Wikipedia on how this works that we strongly recommend the reader to check out (if you're reading the PDF, please click "Wikipedia").

Mathematically, we're trying to maximize the expectation value of the log-likelihood. We have a set of observed values X, a set of unobserved values $\{x_1, \ldots, x_n\}$ of length n, that belong to the same stochastic process as X and are related with some set of unknown parameters θ. The unobserved values in this case can be seen as points that should be included in the cluster, but are not yet. The log-likelihood of these unknown parameters θ can then be expressed as equation (2.12).

$$L(\theta; X) = p(X|\theta) = \int p(X,x_i|\theta) dx_i \tag{2.12}$$

However, this is intractable in many cases, which is why we have to estimate it with the EM-algorithm (Algorithm 2).

Algorithm 2: Expectation Maximization

Data: Your dataset, number of clusters K
Result: Cluster for each point
Randomly initialize the parameters θ_k randomly for K different classes.
while θ_k *has not converged* **do**
  for *Every point* x_i *in the dataset* **do**
    $Q(\theta_k|\theta_k^t) = \mathbb{E}_{x_i|X,\theta_k^t}\Big[\log L(\theta; X, x_i)\Big]$
    $\theta_k^{t+1} = \text{argmax}_{\theta_k^t}\, Q(\theta_k|\theta_k^t)$
  end
end

Obviously, there are some pros and some cons to this method. One benefit is that if we already know our data to be Gaussian, EM will find these Gaussian distributions for us. This is especially good if we *know how many* Gaussians we have. Otherwise, a con would be that we'd have to tune the parameters K, which can prove tedious. Since the `while`-loop only stops when θ_k has converged a standard EM-algorithm can converge quite slowly if the hyper parameter K i.e. the number of clusters is not correctly set.

2.6 Anomaly Detection

A clear application of this work would be anomaly detection for the bank. However, since we do not know if a point is anomalous or not before-hand, we are faced with an unsupervised setting. It might seem difficult but we will show a way of evaluating even such a setting in Section 2.7. We will also transform the problem into a supervised setting, which means that we'll first have to cluster the training data in order to find what can be considered normal. After that, we can talk about what is anomalous and how to set up an experiment for that. Each anomaly detection algorithm presented in this section are estimating some sort of 'degree of abnormality' rather than a binary prediction.

Percentile Discrimination (Baseline)

After performing EM-clustering, we assume that the clusters are defined as simple Gaussians. This means that they are very easy to work with. For instance, anomalies can simply be detected by labeling all points on some percentile p as anomalous. How? If we have a random variable $x \sim N(\mu, \Sigma)$ following a multivariate Gaussian of dimension D, we can say that

$$\underbrace{\Sigma^{-\frac{1}{2}}(x-\mu)}_{G} \sim N(0_D, I_D)$$

Where 0_D is an D-dimensional vector of zeros and I_D is the identity matrix of dimension D. Let's call this set of normally distributed variables $G = \{g_1, \ldots, g_D\}$. Then we know that

$$G^T G = \sum_{k=1}^{D} g_k^2 \sim \chi_d^2. \tag{2.13}$$

Which implies

$$\begin{aligned} G^T G &= (\Sigma^{-\frac{1}{2}}(x-\mu))^T \Sigma^{-\frac{1}{2}}(x-\mu) \\ &= (x-\mu)^T \Sigma^{-1}(x-\mu) \sim \chi^2_d. \end{aligned}$$

This means that we can evaluate the PDF of a χ^2_d-distribution for $c(x) = (x-\mu)^T\Sigma^{-1}(x-\mu)$ and get the percentile that x occupies in the original distribution by (2.14).

$$p(x) = \chi^2_d(c(x))) \tag{2.14}$$

Isolation Forest

Percentile discrimination tries to find some kind of "normal" behavior to contrast with "anomalous" behavior. Another way to find anomalies would be to focus on isolating them, finding them explicitly. This is what Isolation Forest was made to do, first described in 2008 by Liu et al. [24].

But first, a word on decision trees and the Random Forest Classifier [25]. Everyone has likely used a decision tree sometime in their life, perhaps without them knowing it. Say you want to predict whether you can play tennis tomorrow. The weather forecast says that it will be sunny and the humidity will be normal so taking a look in our *decision tree* in Figure 2.9 we see that indeed, the conditions will be suitable for tennis.

A Random Forest Classifier would create some number of these trees (a bunch of trees becomes a forest) and then train every one of them on a small subset of the data to classify whether or not the conditions are right for tennis, then averaging the result. This form of learning is what's called *ensemble learning*. Using many weak classifiers to create a stronger one.

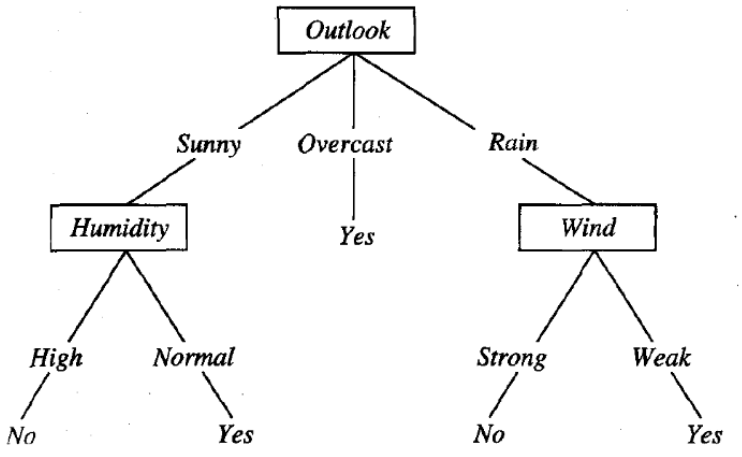


Figure 2.9: A decision tree describing the weather conditions suitable to play tennis. This is a classic example found in any introductory machine learning book [4].

In some sense, Isolation Forest is close to a Random Forest Classifier. But, instead of making classifications we're trying to divide the dataset up into as many parts as possible. In a sense, this is over fitting severely until every point is in its own leaf node. Then we can measure the depth at which all these leaves lie. If the depth is small, then it didn't

take many divides to isolate it, and therefore it is an anomaly. If the depth is deep, it is deeply nested in some cluster, and therefore is not an anomaly.

Algorithm 3: Isolation Forest

Data: Your dataset, number of trees N
Result: Anomaly score for each point

```
for i = 1 to N do
    Initialize a random tree
    Choose attribute j at random
    while Not every data point is in its own leaf do
        Choose splitting threshold θ_j uniformly from [min(x_j), ..., max(x_j)]
        Recursively split the dataset on θ_j
    end
end
Let d(x_i) be the average depth of x_i
Let r(x_i) be the expected depth of x_i
return Score of each point s(x_i) = 2^{-d(x_i)/r(x_i)}
```

As is clear from Algorithm 3 this is a procedure to find the points that get the lowest average depth. Then an anomaly score is given to each point as $s(x_i) = 2^{-\frac{d(x_i)}{r(x_i)}}$. Interestingly, this works in higher dimensions as well because you are not limited to spacial dimensions and distances. The anomaly score is given by the depth in the tree and not some distance. Isolation Forest is also completely unsupervised in its approach which is suitable for our purposes in this thesis.

According to the initial paper this is can be performed in linear time so we do not need to worry about the dataset growing in size. It has also shown very big promise in numerous applications in fraud detection [26] [27]. However, Isolation Forest is sensitive to the shape of the density regions that you wish to apply it to. If the clusters are "bad" i.e. not very densely packed, it will struggle to find the right anomalies. Therefore, it is important to get "good" clusters. If it is going to be successful it'd be great if the clusters were Gaussian.

The SciKit-Learn implementation of Isolation Forest is helpful because you can simply put in the contamination rate you wish to have and it will find outliers for you. The contamination rate is the percentage of data that are classified as anomalous.

Local Outlier Factor

The local outlier factor [28] is a density based anomaly detection algorithm. It is an unsupervised algorithm that computes the local density deviation of a given data point with respect to its k nearest neighbors by computing the distance to these neighbors. By doing this for each point, regions with similar densities can be identified and points that have lower density than their neighbors will be considered to be anomalies.

This approach has some similarities with DBSCAN [29], which is primarily made for clustering of unpredictable shapes but also has an anomaly-detection feature. However, both for clustering and anomaly detection DBSCAN has a major disadvantage in its two hyper parameters ϵ and minPts. LOF is only concerned with the parameter k (minPts) so,

if you are only interested in the anomaly detection aspect, it will be easier to tune.

In order to explain this algorithm we introduce the notion of k-distance(A) in Definition 8. This is equivalent to the set of distances to the k nearest neighbors. We use this set to compute what we call *reachability distance*, also defined in Definition 8.

Definition 8 *Local Outlier Factor. First we introduced k-distance(A)* $= \{d(A,n_1), \ldots, d(A,n_k)\}$ *where* $d(A,n)$ *is the distance measured from A to one of the k nearest neighbors* $n \in N_k(A)$. *Reachability-distance is then defined as reachability distance as*

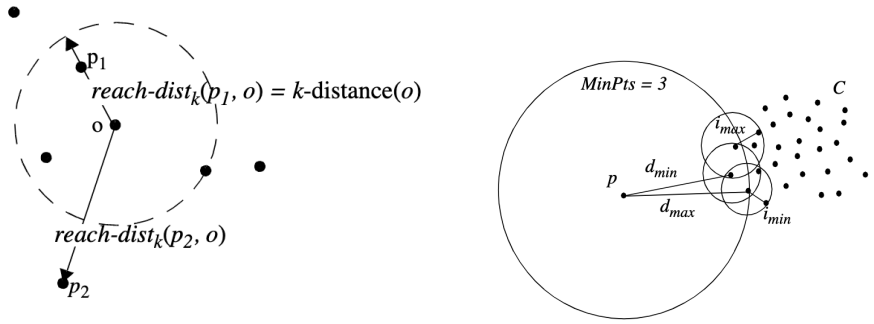
$$\textit{reach-dist}(A,n) = \max\{k\textit{-distance}(n), d(A,n)\}$$

Now the local reachability density can be computed as

$$lrd_k(A) = |N_k(A)|\Big(\sum_{n \in N_k(A)} \textit{reach-dist}_k(A,n)\Big)^{-1}$$

Finally, the local outlier factor (LOF) can be defined as

$$LOF_k(A) = \frac{1}{|N_k(A)|} \sum_{n \in N_k(A)} \frac{lrd(n)}{lrd(A)}$$



(a) Reachability (b) Description of k neighbor region.

Figure 2.10: These two figures describe the concepts of reachability distance and k-distance. k-distance(p) is a set of distances from point p to the k nearest neighbors. The reachability distance is the maximum of these distances. If the reachability distance is high for p in comparison with the k nearest neighbors, it will likely be an anomaly.

From the original paper we have Figures 2.10a and 2.10b that illustrates these concepts clearly. In 2.10a we have a point o and we measure the *k-distance* where $k = 3$ so that the reachability distance makes it possible to reach point p_1 but not p_2. In Figure 2.10b the densities of different regions are evaluated using LOF. In this case, p has a much lower density than the other points in the set C.

The local outlier factor is measuring the average *local* density. If $LOF_k(A) \approx 1$, point A has a similar density to its neighbors, while $LOF_k(A) < 1$ means it has a higher density and $LOF_k(A) > 1$ means a lower density. Because of this local approach, LOF can identify outliers that wouldn't be considered outliers if they were somewhere else in the dataset.

One-Class Support Vector Machine

Support Vector Machines are a set of machine learning algorithms that can perform classification, regression and anomaly detection. They work by mapping the data into a higher dimensional space where it is more easily separable. An SVM is usually a supervised algorithm but the OCSVM works slightly differently. It is assuming that we train on some normal data and then try and predict on some test data whether it belongs to the same class as the training data.

Consider the training data $\{x_1, \ldots, x_k\} \in X$ where k is the number of observations and X is some set. Then let Θ be a feature map $X \to F$, where F is a dot-product space, s.t. the dot product can easily be evaluated using some kernel function $k(x, y)$.

$$k(x,y) = (\Theta(x) \cdot \Theta(y)) \tag{2.15}$$

A common pick is the Gaussian kernel $k(x, y) = e^{-||x-y||^2}$. Then we can separate all the data points from the origin in F and maximize the distance from a hyper plane to the origin. This is done by solving the quadratic problem below

$$\min_{\omega,\xi_i,\rho} \quad \frac{1}{2}||\omega||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \tag{2.16}$$

$$\text{subject to} \quad (\omega \cdot \Theta(x_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0 \tag{2.17}$$

Here, the parameter ν characterizes the solution in two ways. First, it sets an upper bound on the fraction of outliers (training examples regarded out-of-class), second, it is a lower bound on the number of training examples used as support vector. If ω and ρ solve this problem, the decision function $f(x) = \text{sgn}(\omega \cdot \Theta(x) - \rho)$ will be positive for most examples of x_i. Solving it with Lagrange techniques and using a kernel function we get

$$f(x) = \text{sign}\Big(\sum_{i=1}^{n}\alpha_i k(x, x_i) - \rho\Big). \tag{2.18}$$

The decision function $f(x)$ is a binary function which captures regions in the input space where the probability density of the data lives. Thus the function returns 1 in a 'small' region (capturing the training data points) and -1 elsewhere, which will be considered anomalous [30].

SVM's are versatile and can be used with high dimensional data. However, the computational complexity of an SVM is between $\mathcal{O}(n_f * k^2)$ and $\mathcal{O}(n_f * k^3)$ where n_f is the number of features and k the number of samples. Making it scale poorly with growing amounts of data [31].

2.7 Metrics

In order to evaluate the performance of clustering and anomaly detection we need some way of measuring their performance. Luckily there are a few well established and some new

metrics that we present in this section. The results of a metric will be a way of evaluating which of the VAE or CL-VAE prepares the data for these tasks the best. Metrics can also be used for hyper-parameter tuning which will give us a target to optimize our algorithms on.

V-score

Clustering is an unsupervised task, which can make it slightly difficult to measure. However the question of hyper-parameter tuning, in this case the optimal amount of clusters, has to be addressed. Therefore, the V-score comes in handy [32]. It is an entropy-based approach to the problem that consists of two parts:

- Homogeneity - maximized if cluster contains only members of a single class.
- Completeness - maximized if all members of a given class are assigned to the same cluster.

These two properties are both desirable traits for clustering which can easily be measured using V-score (V stands for validity).

For the homogeneity requirement to be satisfied, the class structure in each cluster should be dominated by a single class. In the ideal case were *only* members from a certain class are in one cluster, that cluster would have zero entropy. If we consider a set of classes $C = \{c_i | i = 1, \ldots, n\}$ and a set of clusters, $K = \{k_i | i = 1, \ldots, ,m\}$ then a perfectly homogeneous case would be if $H(C|K) = 0$. In practice we normalize the conditional entropy by $H(C)$ to eliminate the dependency on class size. A perfect solution is reached when this normalized entropy equals zero. By convention, we define the measure to be maximized at 1 and minimized at 0 by Definition 9.

Definition 9 *Homogeneity.*

$$h = \begin{cases} 1, & \textit{if} \quad H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)}, & \textit{else} \end{cases}$$

If a clustering algorithm assigns *all* members of the same class to a single cluster, the completeness requirement is satisfied. This is symmetric to homogeneity, so the perfect clustering would occur when $H(K|C) = 0$. The worst case occurs when $H(K|C) = H(K)$. And again, by convention we define the completeness measure to be 1 at maximum and 0 at minimum in definition 10

Definition 10 *Completeness.*

$$c = \begin{cases} 1, & \textit{if} \quad H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)}, & \textit{else} \end{cases}$$

Now that we have these two entropy-based measures, we can take their weighted harmonic mean and get the expression for the V-score in definition 11.

Definition 11 *V-score.*

$$V_\beta = \frac{(1+\beta)hc}{\beta h + c}$$

In the standard case, $\beta = 1$ and $V_1 = \frac{hc}{h+c}$. Meaning that both homogeneity and completeness are measured equally. By changing the weight β we can change the emphasis of the measure. If $\beta > 1$, completeness is emphasized and if $\beta < 1$, homogeneity.

F1-Score

The F1-score is a widely used metric for classification problems. It relies on two parts - precision and recall. This is similar to the V-score, but instead of measuring the entropy of clusters against their classes, we measure the predicted class with the true label. To understand this a little deeper we'll consider Figure 2.11. If we think of this as binary classification (true/false) a true positive (tp) occurs when the we predict true and the point actually is true. A false positive (fp) occurs when our prediction is true but it actually is false. A false negative (fn) is a point that is true but is predicted as false and vice versa for true negatives (tn).

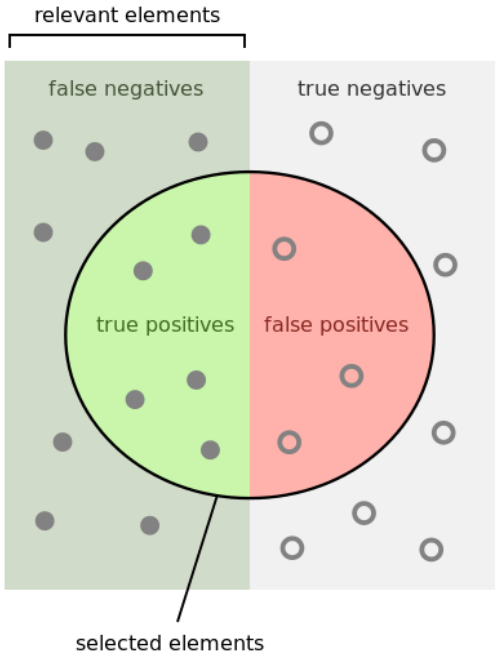


Figure 2.11: A descriptive image on how precision and recall is measured. Precision is computed as a percentage of how many of the selected elements are true positives, while recall is the number of true positives that were selected at all (source: `https://commons.wikimedia.org/w/index.php?curid=36926283`)

From these simple building blocks we can define our two desirable traits for binary classification in Definitions 12 and 13. Precision measures a percentage of points that were actually true when predicted as true while recall measures the percentage of true points that was classified as true.

Definition 12 *Precision. How many selected elements are relevant.*

$$p = \frac{tp}{tp + fp}$$

Definition 13 *Recall. How many relevant elements are selected.*

$$r = \frac{tp}{tp + fn}$$

Like the V-score, the F1-score is then the harmonic mean of precision and recall, as seen in Definition 14.

Definition 14 *F1-score.*

$$F1 = 2 * \frac{pr}{p + r}$$

However, for some problems we might want a higher recall at the expense of precision and vice versa. For instance, if we're making an image recognition system that tries to detect shop lifters it might be good to catch as many as possible, at the expense of getting some false positives (higher recall than precision), this is quite clear in Figure 2.11. On the other hand, if we're worried about accusing people of shop lifting when there is a chance they're innocent, we should prioritize precision rather than recall. Both these approaches are relevant and have to be considered by the engineer designing the system. Sometimes these questions can be hard to answer and depend on many more things than simple economics. For instance, too many falsely accused customers could severely affect brand image and public perception.

The Precision-Recall curve

As was alluded to in the previous section, it's a good idea to study the interaction between precision and recall. For a binary classification problem, it is crucial to get this right, because getting a great measure on one will often make the other measure worse.

Usually, this is done in one of two ways. Either we plot the precision and recall against a threshold, or we plot precision against recall. By studying these two plots we can find a good threshold for one measure that we want - say precision for anomaly detection - without sacrificing too much of the other. This can be done by getting the probabilities of our predicted class from some binary classifier, as well as the precision and recall for these thresholds.

Another measure that's often used in binary classification problems is the Receiver Operating Characteristic (ROC). In that metric we put the true positive rate on the y-axis, the false positive rate on the x-axis and measure how it changes as we change the threshold. However, this is only useful when we have a decent balance between the classes. This certainly not the case in anomaly detection, but has uses elsewhere.

Reconstruction error

Often used as a measure for unsupervised anomaly detection [33] [34], reconstruction error will in this thesis be used for evaluating the reconstructive ability of the VAE compared with the CL-VAE. This is done in order to get a better understanding of the respective latent space representations both as a sanity check and a confirmation that anomalies are also badly reconstructed.

If we consider the vast majority of our data to be normal, then it's reasonable to assume that our VAE learns how to generate normal points better than anomalous ones. If we then measure the error of reconstruction by (2.19) and sort the results, we can find some subset that is badly reconstructed.

$$\mathbb{E}_z\Big[\log p_\theta(x|z)\Big] = ||x - \hat{x}||^2 \tag{2.19}$$

Recalling our previous discussion on the loss function and the tug-of-war effect, it is unclear if this method will find anomalies or simply overlapping classes. Perhaps tweaking the loss could give some interesting results here, but we're primarily after an evaluation metric of the latent spaces of the regular VAE and the CL-VAE. Again, a GAN could have been used to get better samples, but a VAE will still produce samples with a measurable loss which is what we're after.

EM-MV Meassure

Evaluating unsupervised anomaly detection is quite a hard task. As we described earlier, it is not possible for us to confirm the anomalies that are found outside experiments that we device our self. Therefore, we need some other way of measuring the performance of different anomaly detection algorithms on our dataset. Ideally, we would like to find some kind of measure that can be shown to agree with existing supervised methods like ROC or PR. This is where the EM-MV measure comes in [35].

Instead of measuring anomalies directly, this approach aims to measure how well an algorithm can estimate a *level set*. A level set of a function f of n real variables is the set s.t. $L_c(f) = \{(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n) = c\}$. Meaning, a set of values where the function equals some given constant c. The function in this case is the probability density estimated by our autoencoder in the latent space.

In order to estimate the function f we will return some scoring function $s : \mathbb{R}^d \to \mathbb{R}$ which aims to capture the "degree of abnormality". In order to optimize the scoring function we would like to measure some distance from it to f. However measuring $||s - f||$ does not work because the level sets of any strictly increasing transform T of f has the same level sets as f. So instead we introduce the criteria $C^\Phi(s)$ that measures the distance by introducing the function $\Phi(s)$ which is the mass-volume (MV_s) or excess-mass (EM_s) curves of s.

$$C^\Phi(s) = ||\Phi(s) - \Phi(f)|| \quad \text{s.t} \quad \Phi(T \circ s) = \Phi(s) \tag{2.20}$$

Definition 15 *The mass-volume and excess-mass curves of s.*

$$\begin{aligned} MV_s(\alpha) &= \inf_{u \geq 0} \quad Leb(s \geq u) \quad s.t. \quad \mathbb{P}(s(X) \geq u) \geq \alpha \\ EM_s(t) &= \sup_{u \geq 0} \quad \{\mathbb{P}(s(X) \geq u) - tLeb(x \geq u)\} \end{aligned}$$

This means that we can divide up our previous criteria in the two measures $||EM_s - EM_f||_{L^1(I)}$, where $f, t \in I$, and $||MV_s - MV_f||_{L^1(J)}$, where $f, \alpha \in J$. Here the intervals $I = [0, EM^{-1}(0.9)]$ and $J = [0.90, 0.999]$ are subsets, suggested by the author [35], on which we evaluate the measures such that the area under the curves does not become infinite. These two criteria can then be minimized which will be the basis for our discussion on

anomaly detection evaluation. From this we also realize that there is some optimal curve for each measure. These are found by maximizing EM and minimizing MV by

$$MV^*(\alpha) = \min_{\Omega \text{ borelian}} \quad Leb(\Omega) \quad \text{s.t} \quad \mathbb{P}(X \in \Omega) \geq \alpha,$$
$$EM^*(t) = \max_{\Omega \text{ borelian}} \quad \{\mathbb{P}(X \in \Omega) - tLeb(\Omega)\}.$$

In Figure 2.12 we see an idealization of what the curves will look like. The important thing to remember is that we want to maximize the area under EM_s and minimize the area under MV_s to make them as close to their optimal curves as possible in the regions I and J respectively. This is a fairly recent development in the field of anomaly detection and is subject to ongoing research but has been shown to agree with ROC and PR on many popular datasets [35]. In this section we only discussed the basics of this measure as found in the simplified presentation of the original paper [36]. For a full description we refer the reader to the original paper.

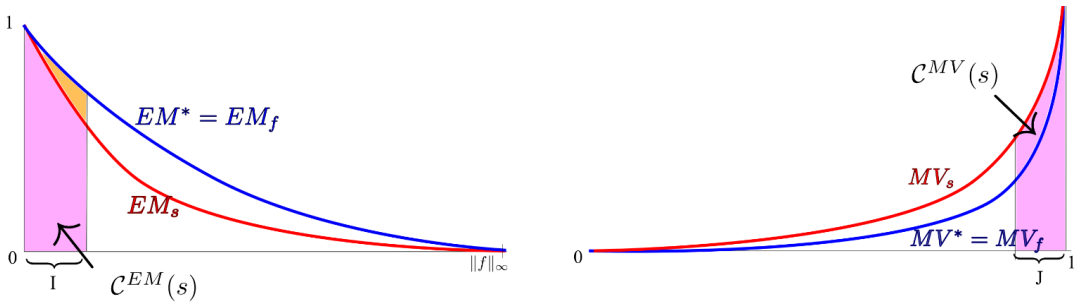


Figure 2.12: An idealization of what the EM- and MV-curves are going to look like for the scoring function s, denoted by EM_s and MV_s. We want to minimize the difference between these measures and their optimal curves, denoted by EM^* and MV^*. This will be done in the intervals $I = [0, EM^{-1}(0.9)]$ and $J = [0.90,0.999]$ such that the area under the curves does not become infinite. Remember that this optimization will aim to *maximize* the area under the curve of EM_s and *minimize* the area under the curve of MV_s.

Chapter 3

Methodology & Proposal

In this chapter we'll discuss the data and describe how to work with some different models as well as how to use metrics to evaluate them. We will also provide some implementation details and introduce the proposed method: CL-VAE.

At first, we approached this problem as a time series prediction problem but quickly ran into issues. One of the problems is that the time stamps are not evenly sampled. This is a requirement if we want to use most common time-series prediction methods - including most recurrent neural networks [4] [7]. In hindsight this is natural to expect as we are only given information when something is happening, which might sound trivial now but was a bit frustrating at first. Another dataset with numeric values for margin and nominal value was also considered but issues arose in data quality when trying to predict these numeric values. If the quality is not superb, this quickly becomes a very challenging problem.

Abandoning that approach, we became interested in generative models [2] [16] and the idea of estimating an underlying distribution. This would mean we could use the attributes of the distribution in order to find out what is an anomaly and what is not. Since this seemed like a rather straight forward approach, it was the path we embarked on.

3.1 Data & Pre-Processing

Any project in machine learning start with a data exploration. Unfortunately the data we have is highly classified trade secrets that fall under the current regulations of bank secrecy. Therefore, the raw data cannot be shared in this thesis, so we'll have to discuss the data in a more high level domain. Let's first recap what we said about the data in the introduction then we'll discuss it a little further and how we will work with it in practice.

We have a dataset of trades done by traders at the investment bank Handelsbanken Capital Markets. Each trade has features such as `traderName`, `portfolioName`, `cptyName` (counterparty name), `acquirerName`, `margin` (ranges of numeric values), `instrumentName` which includes bonds, swaps, future/forwards, FX and many more, `currency`, `timeOfTrade`, whether it was a buy or a sell, `nominal` (again, ranges of numeric values) and the list goes on. Each trade has 27 of these columns that all have varying amounts of categories in them. There are about 250 acquirers, over 5000 counter parties and about 400 portfolios.

Any given trader will not trade with all of these but rather specialize in a few. For the purpose of finding trading categories we are really after what traders trade in similar markets, meaning that they have many of these attributes in common.

After looking at a few traders that were trading with similar instruments we realized that they indeed had many features in common. In some cases traders that trade mainly in bonds have about 50 % of portfolios and even more counter parties in common. The same goes for traders that deal mainly in future/forwards. However, the trading behavior of some traders are a bit more difficult to grasp. Some categories in these different features are extremely rare which is to be expected, and is also what we want to model. Therefore, we can't sort out rare events as they are important to figure out what an anomaly is in the latent space.

A common theme among all the features in the data is that they are categorical or can be made into categorical features. This means that we'll have to one-hot encode them in order to feed them into a neural network. That is, we transform the categorical columns to simple 1/0 values instead, an example follows in table 3.1 and 3.2. Since we have many different features that we perform this kind of transformation on we simple stack the arrays together horizontally in an algorithm. To deal with the timestamps we pick out the hour, day and month and make them into categorical features instead. After that not much pre-processing is done. This is because we want the neural network to do the heavy lifting in finding good internal representations.

Car id	Color
1	Blue
2	Red
3	Green
4	Blue

Table 3.1: A dataset describing the color of cars.

Car id	Blue	Red	Green
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

Table 3.2: A one-hot encoding of table 3.1

As the observant reader will realize, one-hot encoding categorical information becomes very memory expensive. This dataset has more than 300 000 entries with about 6000 unique labels. Meaning, that we would have to deal with a 300000 by 6000 matrix if we wanted to one-hot encode everything at once. Luckily we can use a generator to avoid memory issues. A generator is simply a call that reads a batch of data from disk to the Keras model during training. Still, using so many columns will take a long time to train as the amount of nodes that have to be trained would grow. Later we want to do hyper parameter tuning which would also be very time consuming and using the entire dataset is not necessarily needed in order to make some theoretical points before looking at the full data.

Therefore, we decided to divide the dataset into 2 smaller datasets based on the amount of trades that the traders had done. The first one is a set of traders that have between 5000 and 20 000 trades. That resulted in a dataset with 11 traders and the total amount of trades about 100 000. Remember that the data is very imbalanced when considering the amount of trades that a trader has done. The trader with the most trades in the big dataset have more than 120 000 trades while most of the traders actually have less than 5000 trades. It is inevitable to train on a large dataset to confirm my results later, so the second dataset is all the traders that have more than 1000 trades which turned out to be

50 traders. This threshold was set as we need some amount of trades in order to know what is "normal" for that trader.

After shuffling, the data was divided up into training and test sets, where 80% of the data was considered training and 20% test. The decision to divide the data up like this is not always the way we want to go when using an Autoencoder. Sometimes we'd like to train on all the data if we view it as simply a transformation. But we want to show the effects of predicting where new data will land in the latent space without it being viewed by the Autoencoder before hand. During training of the VAE the test set will be used as validation.

VAE in Keras

I created a VAE using the Keras Functional API, inspired by the excellent tutorial on VAEs by Tiao [11] and the even simpler implementation by Su [38]. This kind of implementation is very flexible and makes it possible to do many interesting things with VAEs.

Layer (type)	Output Shape	Param #	Connected to
input_x (InputLayer)	(None, 1539)	0	
hidden_layer (Dense)	(None, 256)	394240	input_x[0][0]
z_mean (Dense)	(None, 2)	514	hidden_layer[0][0]
z_log_var (Dense)	(None, 2)	514	hidden_layer[0][0]
sampling (Lambda)	(None, 2)	0	z_mean[0][0] z_log_var[0][0]
Decoder (Sequential)	(None, 1539)	396291	sampling[0][0]

Total params: 791,559
Trainable params: 791,559
Non-trainable params: 0

Figure 3.1: Summary of the VAE model, built in Keras. The input layer is set to the dimensions of the reduced dataset (1539, the traders not included) with only 11 traders. Using the larger dataset this number would be 3239, meaning that the number of total parameters would be 1 663 759, more than twice that of the reduced set.

The basic building blocks of a VAE are 2 neural networks and the stochastic layer in the middle recalling Figure 2.6. In Figure 3.1 we can see the different building blocks we used to create this model. The input layer has some number of inputs which correspond to the number of inputs in our data, the one-hot-encoded trades. A hidden layer follows with some number of nodes that outputs the number of latent dimensions that we desire. For most purposes in the thesis we use 2 dimensions for the sake of visualization, but there will be a section discussing this later.

The type of layer used here are densely connected layers, the most simple type of neural network layer in Keras, similar to the multi layer perceptron described earlier. The hidden layer acts as input to both the mean and the variance of the latent space representation of the data, described by 2 additional dense layers. These are then sampled with a Lambda layer, which is then input to the decoder. Which consists of a hidden layer with the same

dimensions as the first hidden layer and has the same output dimension as the input layer.

Keras also lets we define custom loss functions, which we'll have to do in order to combine the reconstruction loss with the KL-divergence. The KL-divergence was defined as the closed form solution we described in property 2 and the reconstruction loss by Keras built-in metric for binary cross-entropy. In fact, the mean squared error is the cross-entropy between the empirical distribution and a Gaussian prior [**?**], which we will exploit later.

3.2 Proposal: CL-VAE

Another way of preparing the data for clustering would be to not fit the latent representation of the data to a normal Gaussian but rather some N Gaussians in a GMM. As was described in the previous section, we now rely on classical clustering algorithms of the latent space in order to find the representations we want. Instead, we now want to try and initialize some N Gaussians and then update them in training, in order to make the latent space easier to cluster.

Again, let's consider an unknown probability distribution that we want to estimate $p_\theta(x)$. Just like before for the regular VAE we want to estimate it with a Gaussian in the latent space but now we want to conditionally chose Gaussian on what trader did what trade.

$$p_\theta(x) = \sum_{y=1}^{N} \int p_\theta(x|y,z)p_\theta(z|y)p_\theta(y)dz \tag{3.1}$$

Here $p_\theta(y)$ is simply a weighted uniform distribution since the traders are described as discrete constants and $p_\theta(z|y)$ is a set of N Gaussians with unknown mean and variance. By using Bayes rule we find out that

$$p_\theta(z|x,y) = \frac{p_\theta(x|y,z)p_\theta(z|y)p_\theta(y)}{p_\theta(x)} \tag{3.2}$$

which can be estimated with the unknown conditional distribution $q_\phi(z|x,y)$. Noting that $p_\theta(y)$ is known and gives $p_\theta(z|y)$ we can write $p_\theta(x|y,z) = p_\theta(x|z)$. Now we have to find a loss function that we can optimize which is achieved by exactly the same logic we applied to the regular VAE. Now, the ELBO becomes

$$L(\theta, \phi; x) = \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - D_{KL}(q_\phi(z|x,y)||p_\theta(z|y)) \tag{3.3}$$

Further studying the second term, and acknowledging that y is discrete we find that

$$D_{KL}(q_\phi(z|x,y)||p_\theta(z|y)) = \mathbb{E}_{z|y}\left[\log \frac{q_\phi(z|x,y)}{p_\theta(z|y)}\right] = \sum_{y=1}^{N} p_\theta(z|y) \log \frac{q_\phi(z|x)}{p_\theta(z|y)} \tag{3.4}$$

As we may recall from our description of the ELBO, we need to find some closed form solution to (3.4) in order to use (3.3) as a loss function. A solution to this was presented by [38] and goes as follows.

Property 3 *Solution to $D_{KL}(q_\phi(z|x,y)||p_\theta(z|y))$ in the non-normal Gaussian case.*

Let $q_\phi(z|x)$ be some Gaussian distribution that is conditioned on x

$$q_\phi(z|x) = \frac{1}{\prod_{i=1}^{d}\sqrt{2\pi\sigma_i^2(x)}} \exp\left\{ -\frac{1}{2}\left|\left|\frac{z-\mu(x)}{\sigma(x)}\right|\right|^2 \right\}$$

and $p_\theta(z|y)$ some Gaussian with mean defined by μ_y and variance 1.

$$p_\theta(z|y) = \frac{1}{(2\pi)^{d/2}} \exp\left\{ -\frac{1}{2}||z-\mu_y||^2 \right\}$$

then

$$\log\frac{q_\phi(z|x)}{p_\theta(z|y)} = -\frac{1}{2}\sum_{i=1}^{d}\log\sigma_i^2(x) - \frac{1}{2}\left|\left|\frac{z-\mu(x)}{\sigma(x)}\right|\right|^2 + \frac{1}{2}||z-\mu_y||^2$$

Note that because of the reparameterization trick, we know that $z = g_\phi(x,\epsilon) = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$ where $\epsilon \sim N(0,1)$. This means that the second term actually becomes $-\frac{1}{2}||\epsilon||^2$ regardless of the parameter, so

$$\log\frac{q_\phi(z|x)}{p_\theta(z|y)} = -\frac{1}{2}\sum_{i=1}^{d}\log\sigma_i^2(x) + \frac{1}{2}||z-\mu_y||^2$$

Finally, finding $\sum_{y=1}^{N} p(z|y)\log\frac{q_\phi(z|x)}{p_\theta(z|y)}$ becomes a matrix operation which is easily computed numerically.

Revisiting the section on intuition of the loss function, especially Figure 2.7, we realize that the issue of REC vs KL is now quite different. Let's say we use the name of the trader as the prior y. Then, we can now create clusters that are fitted to that traders trades by having a lower KL. If two clusters would then overlap it would still mean that their trading behavior was similar, but they would have reached that structure without the limitation of having the same prior distribution. In other words, if there's now overlap, it's a stronger argument for the classes to be part of the same cluster. However, if there's some structure within the clusters it means that there's still classes to be found within this domain.

Algorithm 4: Proposed Algorithm: CL-VAE

Data: Dataset of trades X, dataset to condition on y with length k.
Result: Cluster for each point

```
Initialize k Gaussians all with zero mean;
for e in epochs do
    Fit the VAE on X, with prior distribution conditioned on y;
    Update the parameters θ;
end
```

We propose to name this algorithm *Conditional Latent Space Variational Autoencoder* or CL-VAE for short. A similar algorithm was cleverly described by J. Su [38] but it includes a classifier that we don't really need here since we know the class we want to fit. This in turn is related to the DEC-algorithm by J. Xie et al. [39] and the VaDE-algorithm by Z. Jiang et al. [40]. Yet another related work is GMVAE by R. Shu [41]. These are all different from CVAE (conditional variational autoencoder) [42] [43] because

we're now conditioning the latent space itself on the class with a pre-determined amount of Gaussians, not assuming we can map all classes to the same Gaussian as CVAE does. Instead a CVAE conditions the encoder and decoder, keeping the one-gaussian assumption in the latent space.

CL-VAE in Keras

As is evident from Figure 3.2, CL-VAE it is very similar to the original VAE. The big difference is the extra layer we call priors of type "Gaussian". This is a custom layer, which is easily done in Keras. One other difference that wont show up in the summary is that there's actually another input as well. We're now inputting the one-hot encoded `y_train` vector which is not run through the network itself. This will be used in conjunction with the Gaussian layer to condition prior distribution. All the code details will be provided with the thesis of course.

Layer (type)	Output Shape	Param #	Connected to
input_x (InputLayer)	(None, 1539)	0	
hidden_layer (Dense)	(None, 256)	394240	input_x[0][0]
z_mean (Dense)	(None, 2)	514	hidden_layer[0][0]
z_log_var (Dense)	(None, 2)	514	hidden_layer[0][0]
sampling (Lambda)	(None, 2)	0	z_mean[0][0] z_log_var[0][0]
Decoder (Sequential)	(None, 1539)	396291	sampling[0][0]
priors (Gaussian)	(None, 11, 2)	22	sampling[0][0]

Total params: 791,581
Trainable params: 791,581
Non-trainable params: 0

Figure 3.2: Summary of the CL-VAE model in Keras. Note the addition of the new layer `priors`. This will house the information about the different priors that we will use and every new sample in training will be conditioned on its corresponding prior.

3.3 Setup

First, we want to find categories of typical trading behavior. This will be done through the setup shown in Figure 3.3. Clustering is done on the latent space representation z using k-means and EM. Quality of these clusters is evaluated using the V-score. We will compare the results using a regular VAE and the CL-VAE on k-means clustering compared with EM-clustering. We will vary the number of clusters to get measurements on the V-score, completeness and homogeneity in order to decide the optimal amount of clusters for each combination of autoencoder (pre-processing) and clustering algorithm. When we have found the best combination of these we present the clusters as categories of trading behavior using the V-score. After that, we go into the anomaly detection stage.

We take two different approaches to anomaly detection. One unsupervised and one supervised. Because we do not know which trades are real anomalies and therefore they will we impossible to confirm. For the unsupervised setting we will use Percentile Discrimination, Isolation Forest, Local Outlier Factor and One-Class SVM. These will be evaluated using the EM-MV measure. The training and test data in this case is the latent space representation of the original training and test data. This approach is not detecting points within other clusters but rather points that don't belong in any cluster. By optimizing the EM-MV measure we can tune the algorithms and find one that performs the best.

In order to make a supervised problem we have to label the data ourselves. We select some cluster, then randomly draw some number of trades from the other clusters and label them as part of the currently selected cluster. These randomly drawn trades will be labeled as anomalies, meaning that we can verify that we have found them after classification. We will perform this on the test data as we want the clustering to predict the label of trade it has not seen before.

The idea of this is that we simulate the situation when new trades are coming in and some trades end up in places which is not normal. Ideally, this will be able to show that it's possible to detect trades that end up in the "wrong" cluster. We will evaluate anomaly detection using precision and recall and find a setting that performs best on the data. The reason for picking trades that have actually been done in other categories is that they are real trades that would perhaps be normal if they came from some other cluster, but if they came from our selected cluster they would be anomalous. If we were to simply input white noise in the latent space, the trades would be very hard to interpret. They could be some completely ludicrous trade that likely would be caught by some other filter.

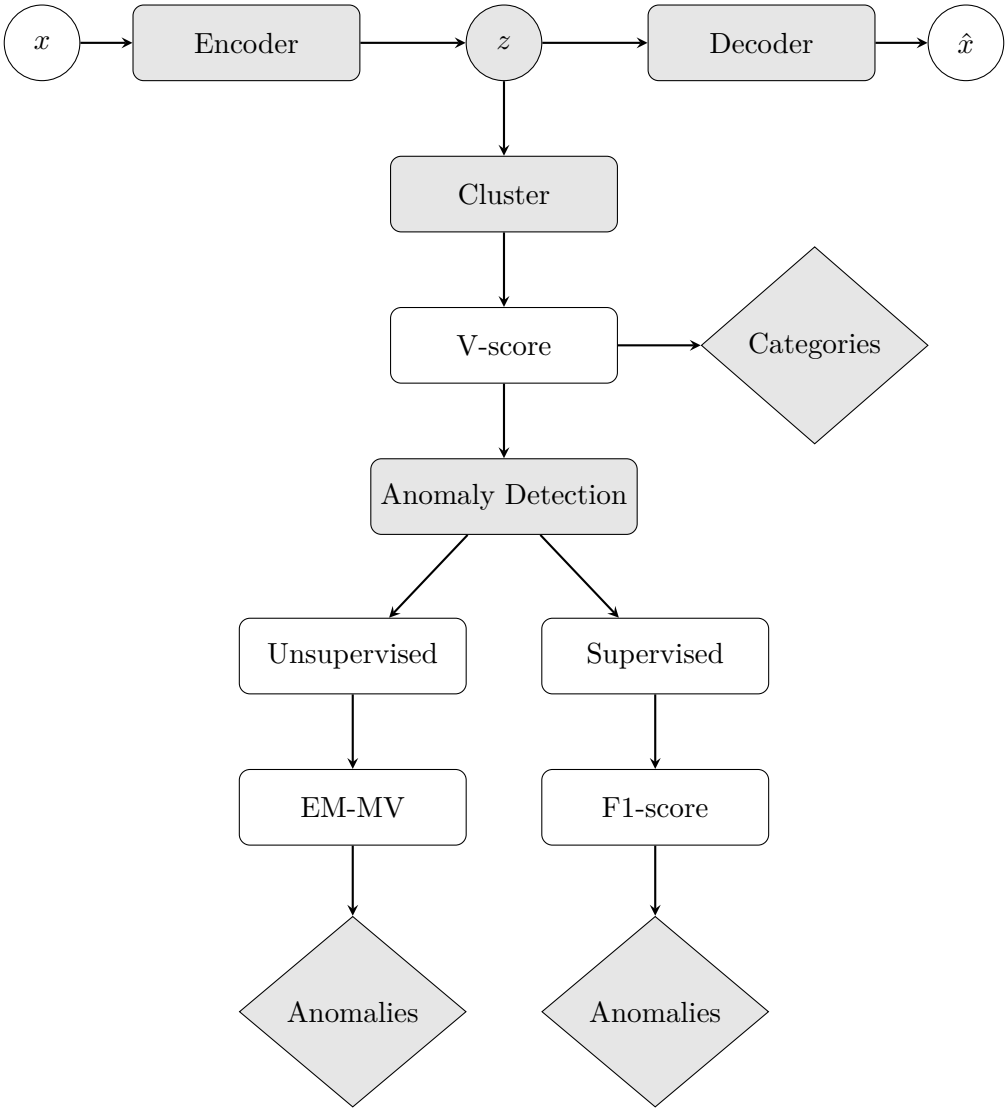


Figure 3.3: A flow chart of the setup used to measure the quality of the categories found by clustering and the anomalies found through anomaly detection. White circle indicates regular data, gray circle indicates stochastic layer (latent variables), white rectangles are measurements, gray rectangles are algorithms and gray diamonds are results. First, the trading data x is encoded to a latent space representation z using a VAE or CL-VAE. Then, the data is clustered and evaluated using the V-score. When the optimal amount of clusters is found we say that they represent *categories* of trading. Then we perform both unsupervised and supervised anomaly detection, evaluating the algorithms with EM-MV and F1-score (and precision-recall) respectively.

Chapter 4

Results

4.1 Clustering

In this first section of the results we're showing optimal hyper parameters for each clustering technique and the optimal V-score using the different pre-processing algorithms VAE and CL-VAE. The goal of clustering is to find a few *categories* of trading that we can use for further analysis. Also, if a category is found, it is interesting to see what traders are in that category.

VAE

As with all machine learning evaluations we have to see that the loss function decreases for both training and validation data which is confirmed in Figure 1, found in the Apendix. In this case, the loss function is the ELBO function and therefore describes the trade-off between reconstruction loss and KL-divergence to the Gaussian normal prior.

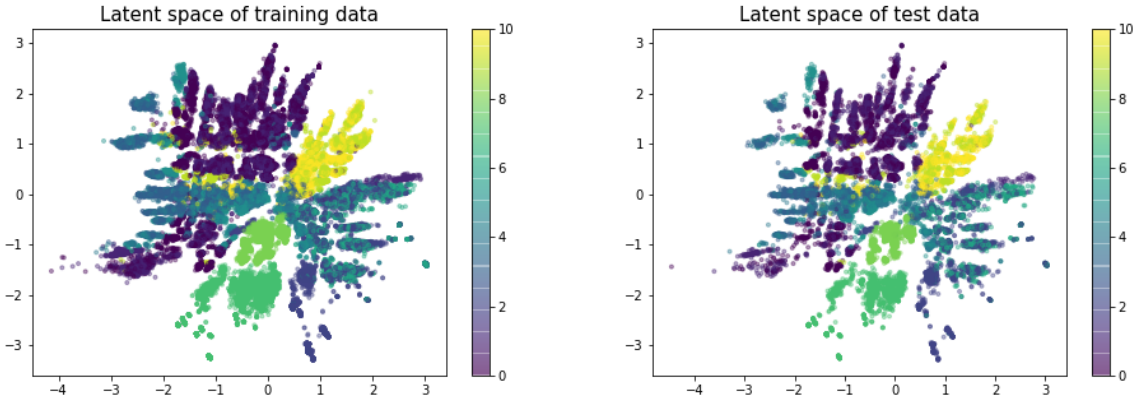


Figure 4.1: An example of a 2D-latent space. Here the colors represent different classes (traders). We can see that there's some form of separation between traders but there's a lot of class overlap. Remembering Figure 2.7, this is because all the different classes are trying to be matched to one Gaussian prior distribution. Because the latent space is non-deterministic, it will look slightly different each time that the algorithm is run.

By using a regular VAE we can find a latent space representation with the dimensionality of our choosing, in this case 2. The results of which can be found in Figure 4.1. In this case, we had 11 traders, indicated with color, that have between 5000 and 20 000 trades each. The model is trained on minimizing the ELBO loss function and with a Gaussian normal prior. This is clear as the points are all expanding outward from the origin were the normal Gaussian has its mean.

Now, from this image we can see that there is substantial separation between the different traders analyzed in this data. This is promising for clustering and categorizing the data, but the clusters that are formed are very spread out and have unfortunate shapes. For instance, the borders between different classes often seem to be ill-defined. This will obviously make it hard for an algorithm to distinguish between clusters.

There appears to be some overlap for some traders, which would be expected if they traded in a similar market, towards similar counter parties and portfolios. However, this will also hurt the reconstruction error as we discussed earlier; class overlap is the enemy of reconstruction.

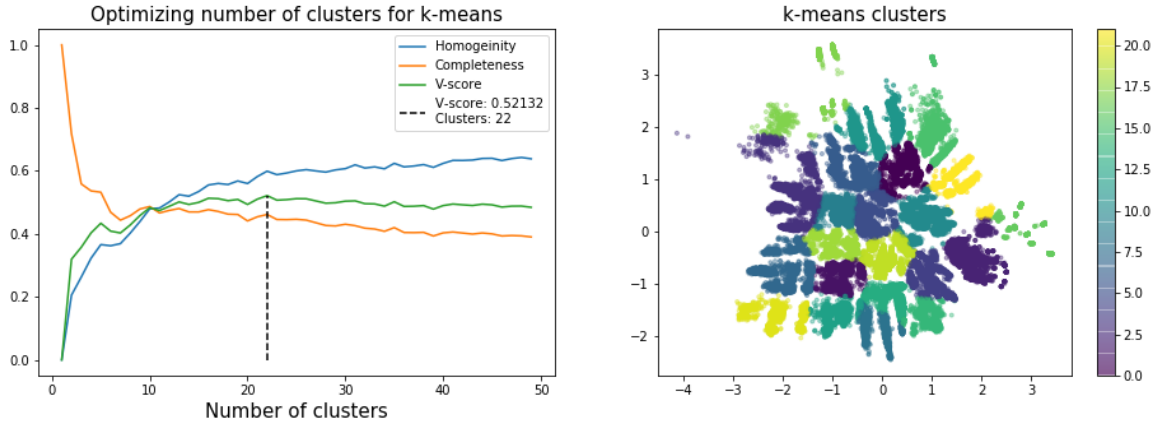


Figure 4.2: Hyper parameter tuning of clustering the latent space of a VAE with k-means clustering. In the left image we see how the V-score, homogeneity and completeness varies dependent on the number of clusters specified in the model. Also, the best V-score and the number of clusters that achieved it is highlighted. On the right we see the result of clustering with 22 clusters. The V-Score varies very little after around 15 clusters as the completeness gradually gets worse and homogeneity better.

Clustering the latent space using k-means with different number of clusters we find that the optimal number of clusters is 22 based on the V-score. However in the case of Figure 4.2, we see that the V-score converges already at around 10 clusters and the bump in 22 is quite small. We also see that the V-score slowly drops of as the number of clusters increase beyond 22 clusters.

The unfortunate thing here is that an optimal V-score of 0.52132 means that we don't get much new information by clustering. If both homogeneity and completeness were at 0.50 it would mean that as many points of a class is within the same cluster as outside. In other words, we have a coin-flip situation if a point is part of the right cluster or not. However, homogeneity is still growing (slowly) as we introduce more clusters. Naturally, if we introduce more clusters we can reach a point where each point is in its own cluster

and then homogeneity would be maximized, but you'd have as many clusters as we have points. In other words, no new information. Completeness is shrinking (slowly), because if the number of clusters is higher than the amount of classes it is past its possible peak and will only get worse.

Performing the exact same test using the EM-algorithm we arrive at Figure 4.3. Here the plot of the V-score are less stable and the optimal V-score is computed to 0.52209 at 26 clusters. Again, there was very little improvement in the V-score when using more than 11 clusters. Such a minor improvement is not particularly noteworthy and the fact that we achieve this results with more clusters than in the case of k-means would suggest that the superior algorithm in this case is actually k-means. This is further motivated by the observation that k-means seem more stable than EM as the number of clusters increase.

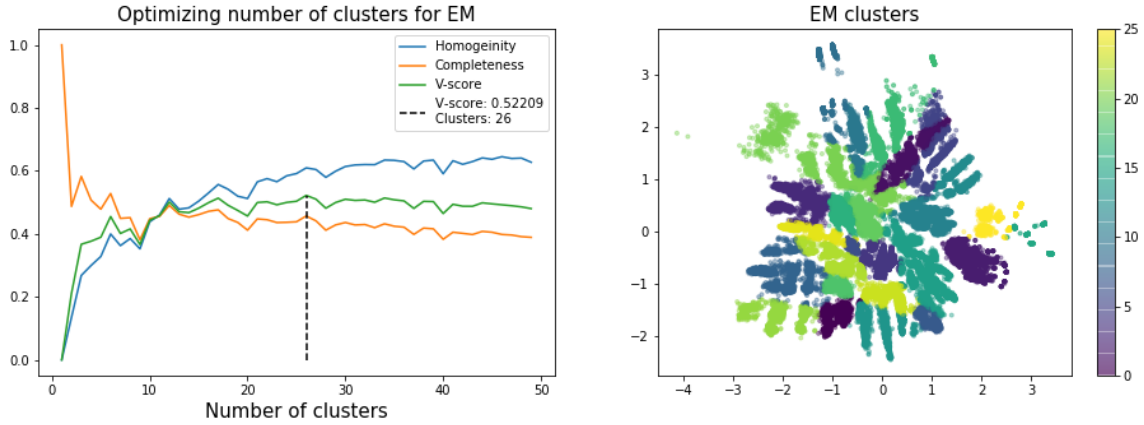


Figure 4.3: Hyper parameter tuning of clustering the latent space of a VAE with the EM-algorithm for clustering. In the left image we see how the V-score, homogeneity and completeness varies dependent on the number of clusters specified in the model. Also, the best V-score and the number of clusters that achieved it is highlighted. On the right we see the result of clustering with 26 clusters. The V-Score varies very little after around 11 clusters as the completeness gradually gets worse and homogeneity better, albeit with a bit choppier curve than in Figure 4.2.

Since we're only using a Normal Gaussian as a prior, it seems like the fact that we get some kind of separation at all is more of a coincidence rather than the objective. If we think about it, Figure 4.1 should look like a plain 2D-Gaussian. But because of the tug-of-war effect of the loss function (Figure 2.7) it separates the different classes in order to have some reconstructive power. Therefore, no assumption is actually made on the clusters that do form other than that they're all part of the same Gaussian.

The clusters we see forming in the image could be seen as trading categories that have been found in an unsupervised manner. However, this may lead to some confusion. Is it the class-structure that we should consider to be a cluster or the points relation to the Gaussian prior? It is not obvious what part of the classes that do form actually indicate "normality". This class overlap would likely not be as severe if more latent dimensions were used. However, we tried using 3 latent dimensions and did not find a measurable improvement. More than that and it will be difficult to visualize what is going on in the latent space.

CL-VAE

Using the same dataset and training the CL-VAE we are presented with Figure 4.4. In this case, we train the CL-VAE on the training data `x_train` while also inputting the one-hot encoded version of the `traderName` vector and the test data as validation. The loss minimization can be found in Figure 2 in the Appendix. As is evident, the classes are much further separated than the equivalent latent space of the regular VAE of Figure 4.1. There's still overlap between some of the traders, but as discussed earlier, this is most likely due to the fact that their trading behavior is actually similar (this will be discussed later). Confirmed by the fact that they all have different Gaussian priors. Another notable feature of this latent space is the size of each cluster and the scales on the axis. In the latent space of the regular VAE the entire dataset fits within the range -3 to 3 on both axes. For the new model, the distance between the points furthest away from each other is about twice the size of that.

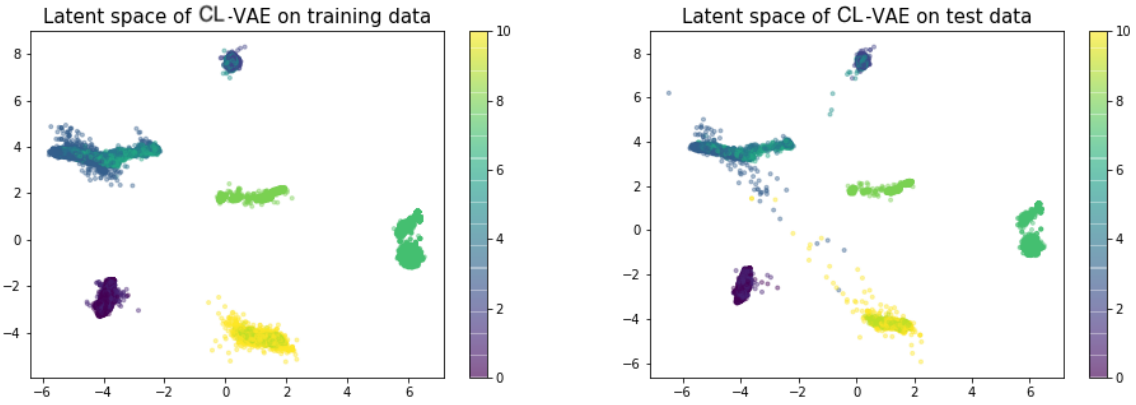


Figure 4.4: An example of a 2D-latent space representation using the CL-VAE. As before, the colors represent different traders (classes). The traders have now ended up in different parts of the latent space. Their groupings are also much more well defined and look closer to their original Gaussian priors than that of Figure 4.1. However, there's still class overlap but this has not happened because they're forced together by the KL-divergence since every trader has its own prior distribution. We also see that some trades in the test data end up in between two groups of traders. The validation loss doesn't show any sign of overfitting (Figure 2 in Appendix). These are the kinds of points that will be discussed in the section on anomaly detection.

Following the structure laid out when clustering the latent space of the VAE, we will first try and cluster the test data in this new latent space using the k-means algorithm. Again, letting the number of clusters go from 1 to 50 we can find some amount of clusters that maximize the V-score. As we see in Figure 4.5, we get a completely new shape to our curve. While previously it grew slowly until some convergence, we now have a V-score that is clearly maximized at 6 clusters with a score of 0.81775.

This is obviously a major improvement from the equivalent Figure in 4.2. Not only do we reach the maximum V-score with a lesser amount of clusters, we also get a significantly higher V-score. Another way of saying this is that we capture more information, using less features, making the modeling both richer and more simple. As before, the completeness

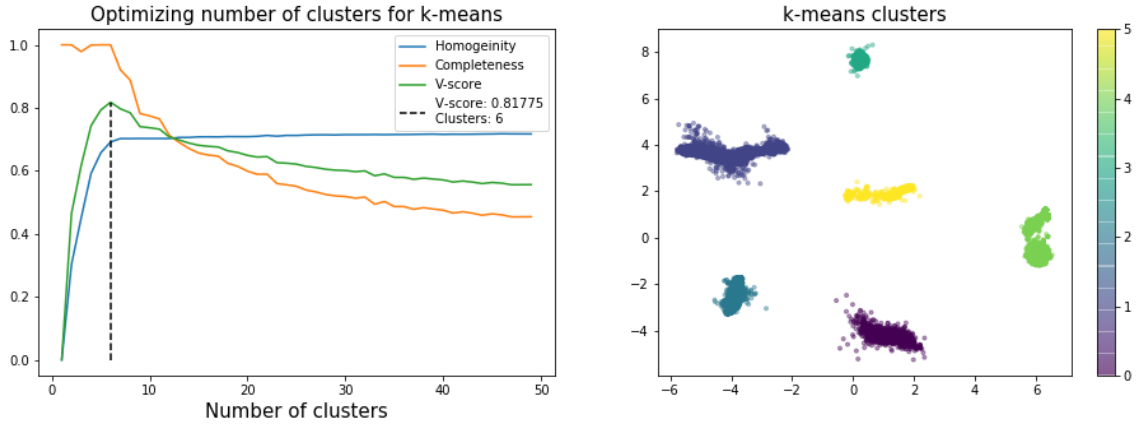


Figure 4.5: Hyper parameter tuning of clustering the latent space of a CL-VAE with k-means clustering. In the left image we see how the V-score, homogeneity and completeness varies dependent on the number of clusters specified in the model. Also, the best V-score and the number of clusters that achieved it is highlighted. On the right we see the result of clustering with 6 clusters. The V-Score has an interesting elbow with 6 clusters which gives a clear maximum for the hyper parameter. After that, the completeness drops of while the homogeneity converges. This convergence means that k-means doesn't find any better way to divide up the data in order to give each trader a its own cluster.

measure is decreasing as we increase the amount of clusters. However, the homogeneity measure has a peculiar behavior. Instead of continuing to grow as the amount of clusters increase, it converges right at the optimal V-score. Because the homogeneity is measuring to what extent each cluster contains only members of a single class, it will reach some point where it can not divide the dataset further. This is reached earlier here perhaps because the clusters are more well defined. If there's a well defined cluster where more than one trader reside, it will be difficult to divide it up further. This goes hand in hand with the completeness decreasing, as the more clusters we introduce will categorize the data further. If that is the case, then naturally clusters will form such that not all members of a class is part of the same cluster.

Performing the same experiment using the EM-algorithm we reach quite similar results to the k-means experiment. A maximum V-score of 0.81769 and the optimal amount of clusters is again 6. However, a notable difference is the behavior of the completeness measure and by consequence the V-score. Instead of decreasing as the number of clusters grow, it converges close to the homogeneity. This is actually quite a sought after property, because it implies that the clustering is *robust* to hyper parameter change. Robustness is important because cross-validation is not possible in real-world applications of cluster analysis, as described by Xie et al. when the DEC-algorithm has introduced [39]. The reason for the convergence is likely due to the nature of the EM-algorithm. At some point it will reach a level where not much is gained from introducing more clusters because it is fundamentally doing *soft* clustering. It is finding some probability of each point to belong to a certain cluster, and if no probability exceeds the previously established clusters, points will not be assigned to any new clusters. This can be illustrated in Figure 4.7 where the EM algorithm being run with 16 clusters compared with 50 clusters. In both pictures, some amount of larger clusters are found and then as the amount of clusters grow higher, individual points seem to be assigned to clusters instead of actual new groups.

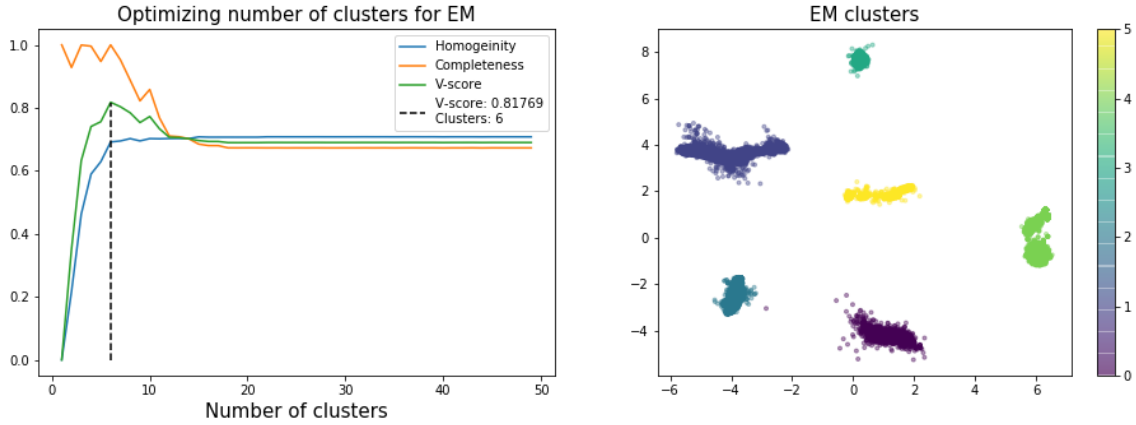


Figure 4.6: Hyper parameter tuning of clustering the latent space of an CL-VAE with the EM-algorithm for clustering. In the left image we see how the V-score, homogeneity and completeness varies dependent on the number of clusters specified in the model. Also, the best V-score and the number of clusters that achieved it is highlighted. On the right we see the result of clustering with 6 clusters. Just as in Figure 4.5 we find a clear elbow to the V-score by using 6 clusters. What's also interesting in this image is that both homogeneity and completeness converge to almost the same rate.

Summary on clustering

Results discussed in the above section is summarized in table 4.1. Clearly, the CL-VAE is more suited for this job than the VAE as it outperforms it on optimal V-score as well as number of clusters. EM and k-means perform almost equally well on the CL-VAE dataset. k-means got an ever so slightly higher V-score, but it is more susceptible to hyper parameter change. Going forward, we will use the CL-VAE in combination with EM-clustering because of its robustness to hyper parameter change in comparison with k-means.

Latent space	EM, V-Score	k-means, V-Score	EM, clusters	k-means, clusters
VAE	0.52209	0.52132	26	22
CL-VAE	0.81769	0.81775	6	6

Table 4.1: Optimal V-score and clusters for each clustering algorithm and pre-processing algorithm. Clearly, CL-VAE is a superior pre-processing algorithm for clustering with this purpose as it gets better V-scores with fewer clusters in each test. k-means performs ever so slightly better in V-score for the same amount on clusters on the CL-VAE. However, the robustness of the EM-clustering shown in Figure 4.6 is not to be neglected which is why we choose that algorithm.

This result is not only exciting due to the better class separation but also the density of the classes. We're basically forcing the categorical input data into Gaussian distributions. We're not relying on first letting a VAE with a normal Gaussian distribution do the mapping and then layering another clustering algorithm on-top without thinking about the structure of the latent space. Using the CL-VAE, we can prepare the data so that it is customized for clustering.

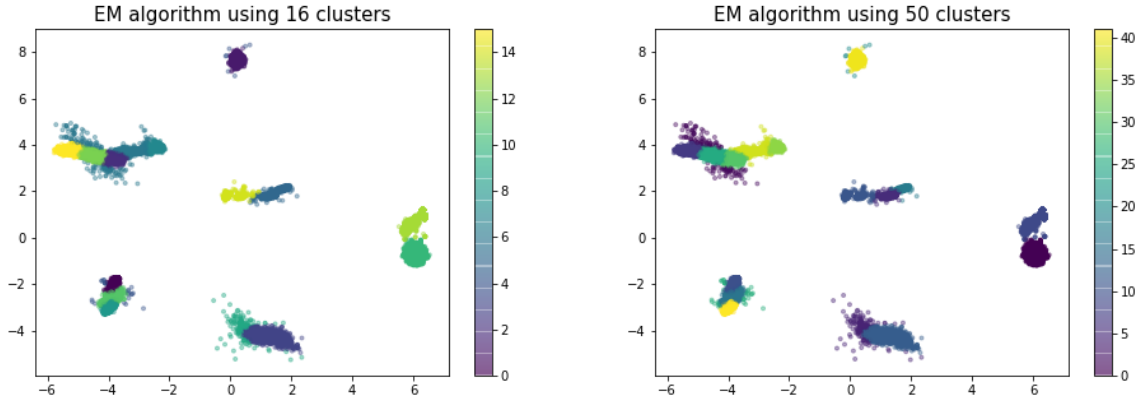


Figure 4.7: A comparison between using 16 and 50 clusters in the EM algorithm. Note that the larger clusters that are formed in either plot is largely the same. Many individual points get there own clusters which doesn't change either completeness or homogeneity. Which is why it converges.

Interpretation on clustering

What ends up in each cluster can be quite a difficult question to answer. After all, the only condition we have is that each trader is given its own Gaussian. How these distributions end up in relation to each other is completely up to the algorithm. Regardless, if we take two highly overlapping classes in the dataset, trader 0 and trader 1 from Figure 4.4, we can study if they are similar in any way. These two traders are classified by our EM-algorithm to be part of cluster 3 as seen in Figure 4.6. In fact, both these traders deal primarily in what is known as fixed income execution.

This means that they are trading against similar counter parties, portfolios and acquirers. In fact, more than 50% of their trades are against 1 portfolio that do not appear frequently in any other cluster. They are almost exclusively dealing in bonds, which is not a deciding factor in itself, but putting it in combination with the other features described this far makes it rather unique. Albeit, not that unique. If we study the 'typical' types of trades that all trader do in the bottom left corner of the latent space we will find bonds as a very common type of instrument. This is one of the deciding factors in the latent space as bonds are very rare up in the right corner where we instead will find swaps and future/forwards being more common. What kind of instrument is traded is also correlated with the other categories in the data which can be a reason for this divide.

Because of the ability of the CL-VAE to divide the dataset up in such clear clusters we can really get a better understanding of what goes on in the trading data and which traders belong grouped together. A seasoned specialist could most likely do the same kind of divide by knowing what to look for, but here we just feed an algorithm the data and it sorts it out itself. Because of this, we can also be more certain that the clustering *means* something, which is not at all necessarily the case with the regular VAE. Considering that we're clustering more than 80 000 trades in the training data and another 20 000 in the test data, this is a convenient way of classification.

Running clustering on more data

Until this point we have used a reduced dataset, only looking at traders that have between 5000 and 20 000 trades. That's 11 traders, but the dataset includes many more. The reason for doing this is for the sake of visualization and preventing clutter. Instead using the larger dataset with a lower bound of 1000 trades will give us 50 traders. With one trader that has about 120 000 trades and the total dataset is consisting of about 316 000 trades. Obviously, that one trader would dominate the latent space if we let everything have the same prior. Instead, using an CL-VAE, we would expect to get a much more well defined latent space, as all traders, including the over represented one, would get a Gaussian normal distribution each.

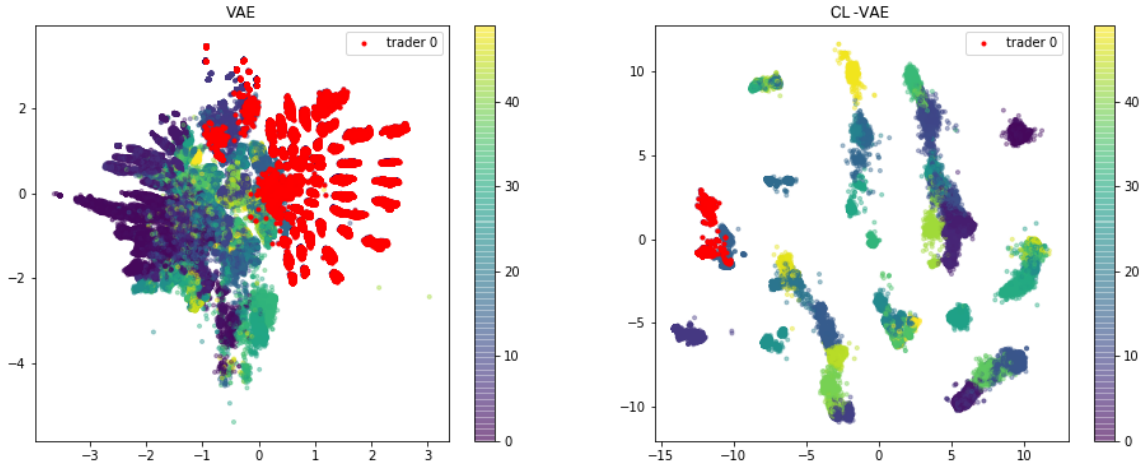


Figure 4.8: A comparison of the latent space of a VAE and an CL-VAE using the same weights on the loss function ($\gamma = \beta = 1$). In these two images, trader 0 with more than 120 000 trades is highlighted in red. The trader with the second most amount of trades has slightly less than 20 000 and from there the next 48 traders has between 20 000 and 1000. By having trader 0 occupy the same prior distribution as all the other ones, he or she will completely dominate the latent space. Affecting the structure of all the other traders, making them overlap more than if trader 0 was not included. If each trader gets their own prior distribution, this competition phenomenon is eliminated and each traders cluster will find its spot in relation to all the other ones as seen in the right image. Therefore, the CL-VAE can be helpful if we want to cluster the latent space and we have severe class imbalance.

This is confirmed in Figure 4.8. The VAE will set aside a large part of the latent space for the representation of the trader with about 120 000 trades. It also finds plenty of distinct small classes within that class. At the same time, the CL-VAE will map most to a simple Gaussian, with a few outliers. Having one class dominate the latent space like that will also make it difficult to find anything meaningful for the other classes. This problem is completely subverted when using an CL-VAE.

An issue could be that we loose the some of the structure of that dominant trader. This is not necessarily the case though as again, the latent space of the CL-VAE becomes much larger than that of the VAE. Albeit, the reconstruction of the latent space might be worse

in the CL-VAE. However, if we're concerned about clustering and categorizing data in order to get an understanding of what is "normal" and "anomalous", we don't get much information from the VAE in this case. The CL-VAE however will put all the different traders into some more interpretable space with some traders clearly belonging together into a category. This means that datasets with a high degree of class imbalance are more easily delt with using this method.

Using the best combination of pre-processing and clustering from Table 4.1 we will try and cluster the data of the larger dataset. This was done on an example of the latent space using the same data as in Figure 4.8 and can be Found in 4.9. The EM-algorithm is once again proving stable on the data as the number of clusters grow. It doesn't display as clear an elbow as in Figure 4.6, but a maximum is easily found at 27 clusters and a V-score of 0.88925.

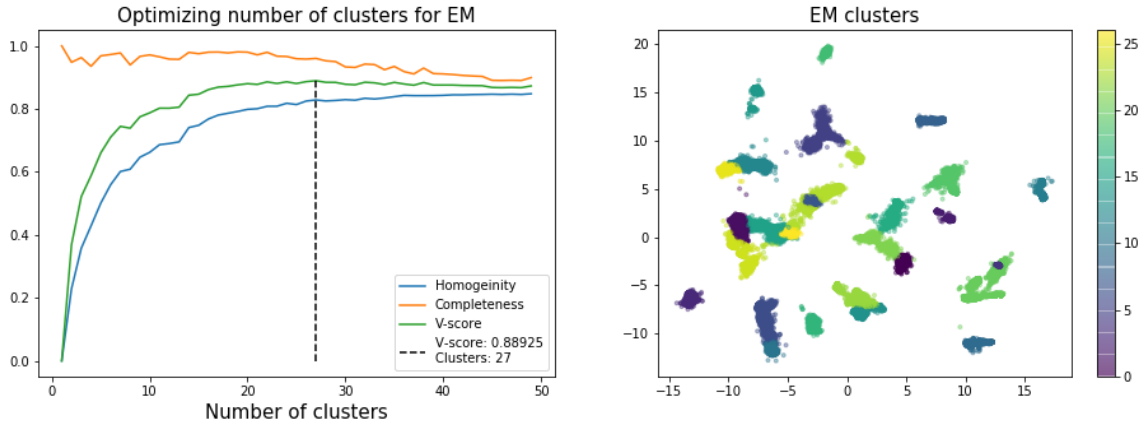


Figure 4.9: Hyper parameter tuning of clustering the latent space of an CL-VAE with the EM-algorithm for clustering, now using the larger dataset. In the left image we see how the V-score, homogeneity and completeness varies dependent on the number of clusters specified in the model. Also, the best V-score and the number of clusters that achieved it is highlighted. On the right we see the result of clustering with 27 clusters. Unlike Figures 4.5 and 4.3 we don't get any clear elbow in the V-score. Instead it smoothly converges together with completeness and homogeneity at around 20 clusters. This is natural as there isn't some clear number of clusters that we can divide this data into as there were for the smaller dataset.

Compare weights on loss function

As an experiment we will change the weights on the loss function to see what happens. Weighting up the KL-divergence we find that the points are more closely mapped to the Gaussian prior as can be seen in Figure 4.10. This means that all the subcategories of trading are more closely aligned with their Gaussian prior than in the case of the unweighted case in Figure 4.4. On the other hand, we also have an example of an CL-VAE with a heavier weight on the reconstruction error. Clearly, there's still some classes that are not studied completely. In fact, the right image starts to resemble the regular VAE with slightly better class separation. Studying each and every single class is outside the scope of this thesis but could theoretically be done to gain insight in what categorizes the structure.

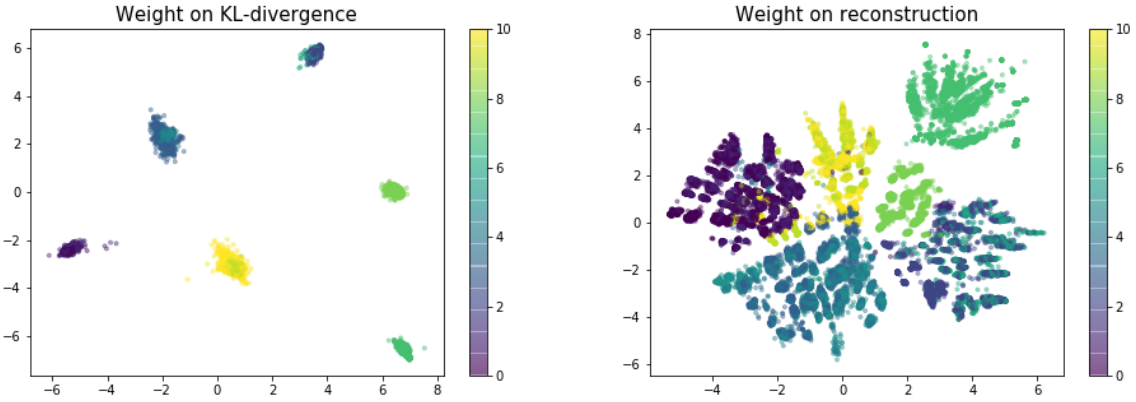


Figure 4.10: Visualization of the 2D-latent space of an CL-VAE with 11 Gaussians as priors. In both cases the algorithm was run for 100 epochs. For the left image the reconstruction loss was weighted down so that $\gamma = 0.5$ and $\beta = 1$, recalling equation (2.11). Conversely, the right was weighted more heavily on the reconstruction with $\gamma = 5$ and $\beta = 1$. Clearly, the points in the left image closely follow their prior distribution. The points in the right image however become far less well organized and start resembling the latent space of the regular VAE from Figure 4.1. These images are showcasing the tug-of-war effect of Figure 2.7 at play, and we see some new classes forming within each prior distribution in the right picture. This is a clear indication that we could divide this data further (beyond traders) so that every one of these small structures would get their own prior distribution.

These two extremes of the loss function can obviously be used for different tasks. If we want to have a more interpretable latent space with clearly defined, simple clusters we might want to go with a lower γ. If we want maximum reconstructive ability but still be able to condition on what trader we want to generate while losing some interpretability we might want to go with a higher γ. Clearly, the tug-of-war effect described from Figure 2.7 can be manipulated to your choosing by using the CL-VAE.

Conditioning on other features

We're not limited to categorizing traders. If we want to categorize on counter party instead we would simple condition on that feature instead. In Figure 4.11 we see 6 counter parties in the range from 5000 to 20 000 trades that have been run through the CL-VAE. Again, we see some class overlap in counter parties 0 and 1, and the separation is quite clear for the other counter parties. The same results in clustering and categorization could be done on this dataset instead just as easily as for the traders.

Technically, we could define our conditioning further and start conditioning on more than one feature. Say, if we wanted to see how a *typical* trade for trader x, in portfolio y, against counter party z, using currency a and instrument b looks like, it is possible! We could continue conditioning like that. However, the amount of priors in the latent space would grow rapidly for each extra conditioning. Finally, taking this logic to the extreme, each and every trade would have its own Gaussian. However, being able to see what a typical trade with multiple conditions looks like is clearly useful.

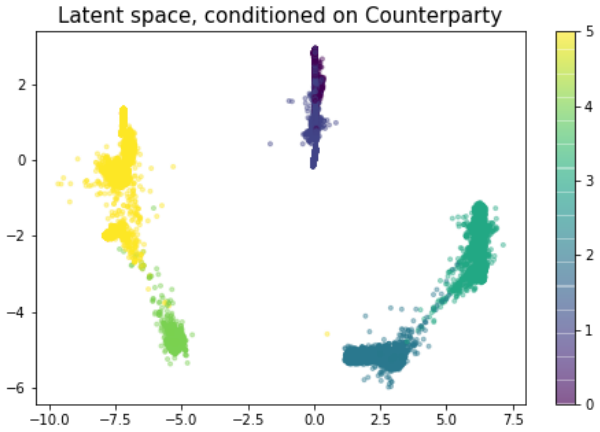


Figure 4.11: The latent space of training data using an CL-VAE, conditioned on counter party with the standard weights on the loss function. Clearly these 6 counter parties can be separated into what would likely be 5 categories. All the same clustering and analysis done for traders could then be applied to counter parties instead.

4.2 Anomaly detection

As discussed earlier, anomaly detection will be done in two ways. First, we will evaluate the performance of the different anomaly detection algorithms in an unsupervised setting, meaning we do not know the target. Here, we will use the EM and MV curves in order to measure their performance and to optimize hyper parameters. All algorithms are fitted on the training data and tested on the test data. This is done in order to simulate a real situation where an algorithm would first learn the behavior of the clusters and then make predictions on new trades as they are coming in.

Second, we will set up a supervised setting by randomly assigning some trades to traders that did not perform them and see if we can detect it. This means that the trades would be considered normal if some other trader did them and the anomaly will be hidden inside an existing cluster.

Unsupervised

Producing the EM and MV curves as described in section 2.7 we can evaluate the performance of different anomaly detection algorithms on our datasets. Recall that the area under the curve of EM should be maximized and minimized under MV.

VAE

By first trying this on the latent space created by the regular VAE we get Figures 4.12 and 4.15. As is evident, the OCSVM is the strongest candidate followed by Isolation Forest, a baseline predictor and finally LOF. The baseline prediction is created by computing the probability of each point based on the mean and covariance of its cluster described in Section 2.6.

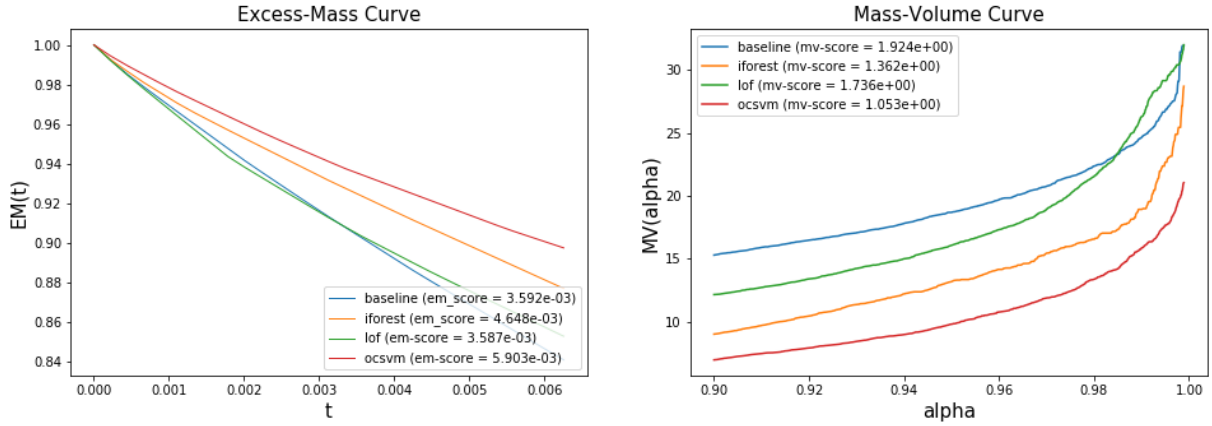


Figure 4.12: The EM and MV curves for different anomaly detection algorithms on the latent space of the regular VAE. The algorithms have been optimized such that they perform their best on the data and EM-MV-curves. Here, the OCSVM performs the best with `gamma`=100, Isolation Forest with a contamination rate of 0.01 and LOF with minPts=200.

All the algorithms are optimized in accordance to hyper parameters as good as possible. Once the contamination rate of the Isolation Forest gets below 0.01 it doesn't get any better but not any worse either, so it appears to be quite stable. The OCSVM seems to view the entire latent space as a single cluster if we don't tune the hyper parameter `gamma` correctly in the SciKit-learn implementation. This parameter is essentially a measure on how well the hyper plane should be fitted to the training data. If `gamma` is too high it will be prone to overfitting, so it is less stable than Isolation Forest, but at `gamma` $= 100$ no such problems appear and it performs better than any algorithm on this data as seen in Figures 4.12 and 4.15. Worth noting is that we haven't manipulated the kernel in any way, using the default Gaussian kernel which felt natural to our problem. Although it does score anomalies a little excessively low, most of the structure remains in the latent space.

However, the most interesting thing is the odd case of LOF. As is expected, LOF finds some local structures and gets its best scores with minPts set around 200. Further optimizing with respect to the area under EM and MV, some strange things start happening. As we decrease the number minPts the AUC scores will get better as seen in Figure 4.13 but once we reach minPts < 100 it will start labeling some lonely outliers with not many neighbors as extreme outliers. It gives some such points scores in the range of -10^9 and almost all others a "sensible" score close to 1 as seen in Figure 4.14.

Even though the area under the curves of EM and MV gets better, they start showcasing some strange shapes. The EM-curve gets a strange tilt around t=0.001 and the MV-curve caps out at $\alpha = 0.98$ and then juts becomes flat up until $\alpha = 1$. At first, this might seem natural in a way, because if the anomaly detection algorithm is labeling some points as extreme outliers, then it would be natural that a measure would show good results. However, revisiting the original EM-MV paper [35] the author is really suggesting smooth curves. The EM curve should be close to a diagonal line and the MV-curve should be as close to the bottom right corner as possible. Such shapes are much closer to what Isolation Forest is producing. So the area under the curve is likely not the only thing that

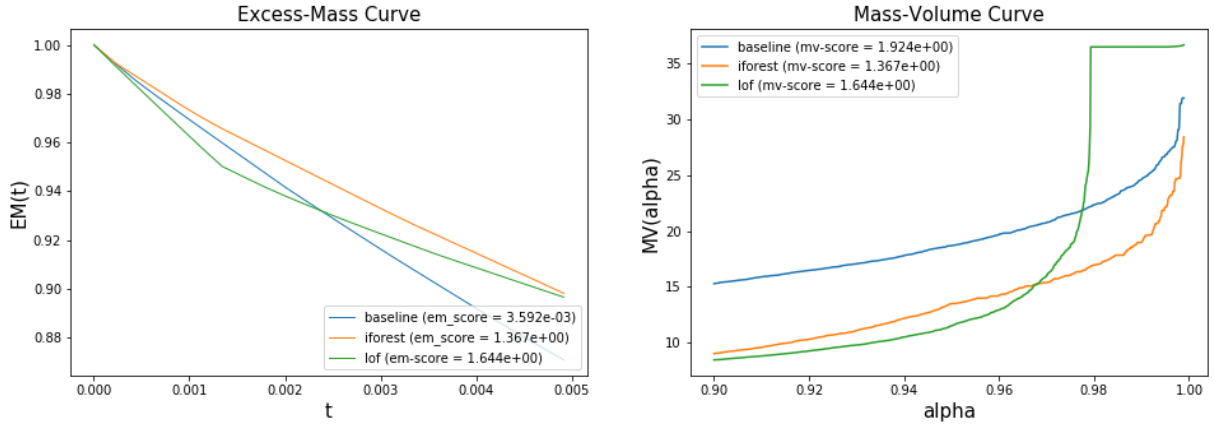


Figure 4.13: Optimizing the area under the EM and MV curves for LOF we find some strange shapes. Even though we get a better AUC score for LOF on both the EM and MV measures this way, their odd shapes should not be ignored. They may indicate in this case that some outliers are labeled *too* extreme in comparison with the other points as seen in Figure 4.14. The OSCVM was left out of this image to showcase the behavior of LOF.

matters.

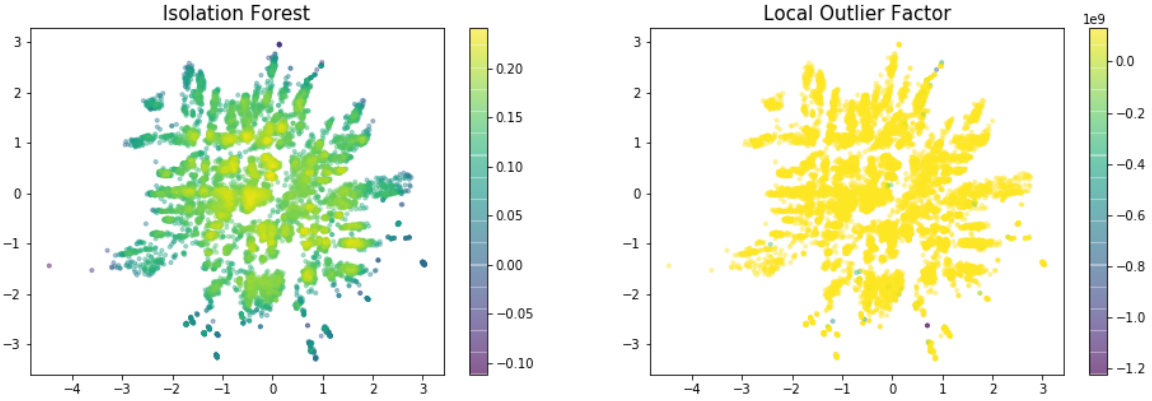


Figure 4.14: A comparison of the optimized Isolation Forest with a contamination rate below 0.01 and the optimized LOF algorithm with minPts=100. Clearly, Isolation Forest captures the structure of the data in a much better way than LOF which simply labels a few outliers as extreme and most other points as normal. This will produce a better AUC score though, even though the result is unreasonable.

Also, the results of the anomaly detection can quickly be verified by looking at the actual data in Figure 4.14. Clearly, LOF is not scoring the points properly. This is also because of the shape of the data itself as we have all these interconnected chunks that, given the wrong settings will score good everywhere using LOF. One should never trust a measure blindly without thinking about the data.

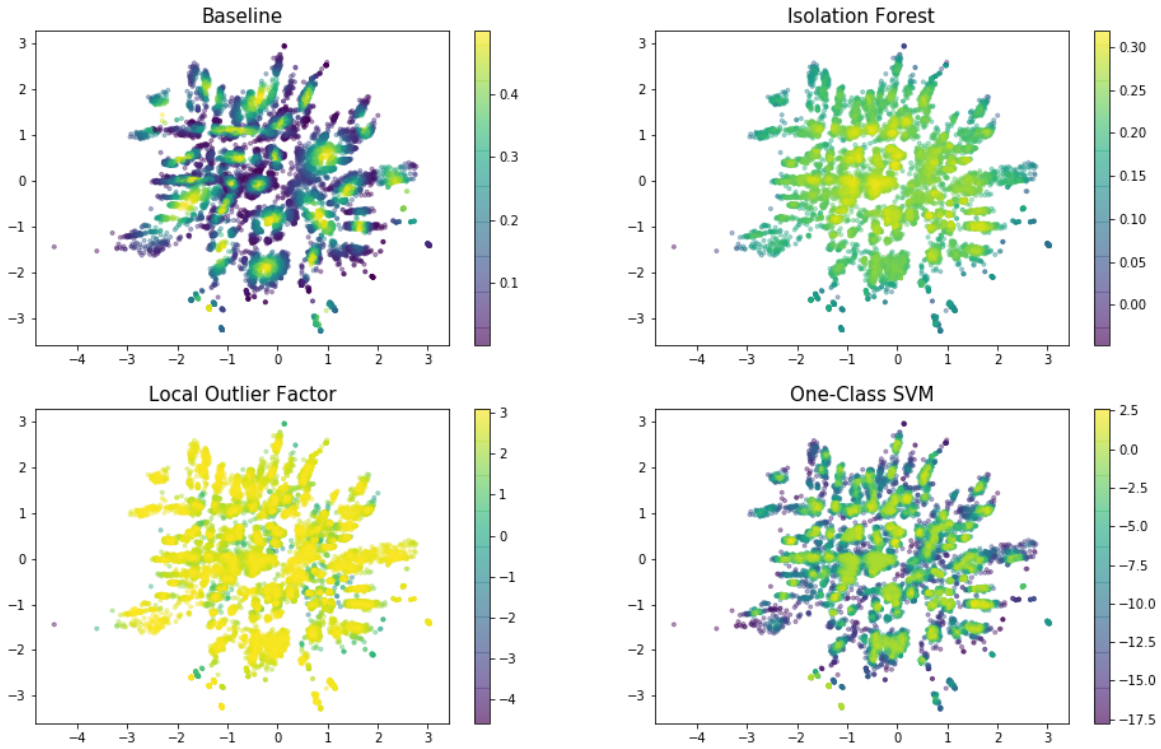


Figure 4.15: The anomaly predictions on the latent space of the VAE with the best performing hyper parameter settings with both respect to the EM and MV-curves and the data itself. Isolation Forest performs the best with a contamination rate below 0.01, followed by LOF with minPts=200, the baseline which doesn't have any hyper parameter and finally the OCSVM that performs the best around `gamma`=100. In this setting, it outperforms the other anomaly detection algorithms.

CL-VAE

Next, let's perform the same analysis on the data created by the CL-VAE. Reusing the same parameters found for the VAE, the OCSVM were actually the one with the strange shapes in the EM and MV curves. This might be due to the simpler shapes in the CL-VAE which might be more prone to overfitting because similarly to the LOF in the last section, it started labeling a few anomalies as extreme, giving them scores of -100 and normal points slightly above zero. This gave the EM-curve the same kind of tilt and a slight cap in the MV-curve as with the LOF in Figure 4.13.

Instead letting it go back to standard settings where `gamma` is equal to the number of features, being 2 in this case, we get another strange behavior in the EM and MV curves as seen in Figure 4.16. We still get a tilt in the EM-curve, even more pronounced this time, and a bump in the the MV-curve, starting in $\alpha = 0.96$ and continuing til $\alpha = 1$ where it rapidly runs away. Looking at what kinds of scores it gives to anomalies in the latent space we get Figure 4.17 where it is compared with Isolation Forest at optimal

settings as reference. As with `gamma`=100 we see that some anomalies are given very extreme scores, as low as -700, while normal points are being scored slightly above 1. Although it doesn't produce such clear cut extreme points as LOF did with its strange settings on the VAE. Also, some whole clusters are now classified as anomalous like the center cluster and the right cluster have all their points below the score of -200. Clearly, this is a sub-optimal setting which thankfully doesn't even optimize the AUC-scores so it likely wouldn't be miss-characterized as an optimal solution. The issue may arise from underfitting, meaning that it will only fit parts of the data, as it labels whole clusters as anomalous.

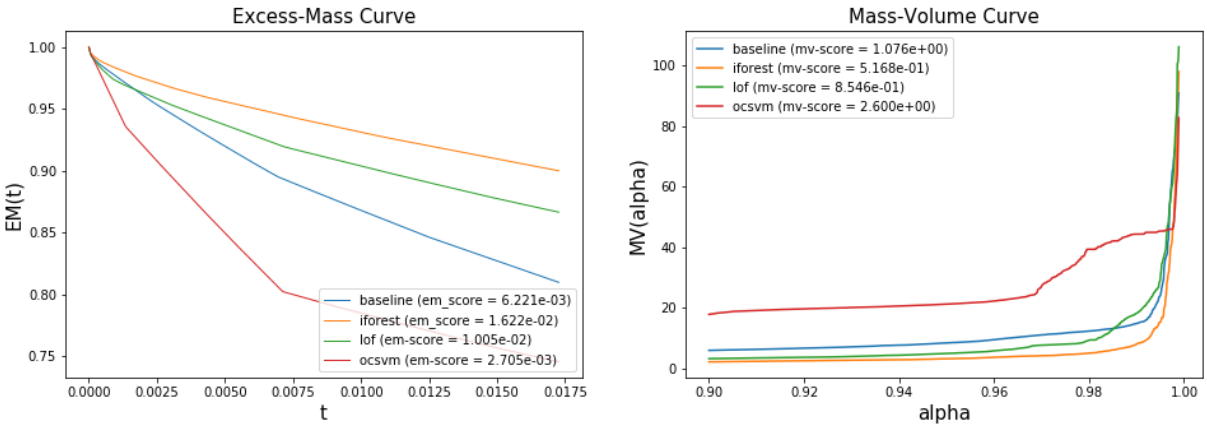


Figure 4.16: The EM and MV curves for different anomaly detection algorithms on the latent space of the CL-VAE. Clearly, Isolation Forest performs the best on this data and is reached using modest settings, with contamination below 0.01. The OCSVM, using the `gamma='auto'` setting performs the worst. It gets a strange tilt in the EM-curve and an even stranger bump near the elbow of the MV-curve. Luckily, this doesn't optimize the AUC scores of either curve so it wouldn't be confusing to that metric. However, this bump in the MV-curve wasn't produced with any other setting or data that was tried and is possibly due to underfitting.

In Figure 4.18 we see the results of using a close to optimal score for the OCSVM that was finally found in `gamma`=20. LOF performs good at minPts=200, but on this data it was quite robust to hyper parameter change. Isolation Forest performs the best of all algorithms at a contamination rate, again, around 0.01 and was very robust to hyper parameter change. At this level of `gamma`, OCSVM performs well on the EM and MV metrics, but taking a look in the latent space and we again see some strange signs, as seen in Figure 4.19. It is giving very low scores to the anomalies and many parts of smaller, less dense clusters like the center and right clusters are largely labeled as anomalous. Labeling that many points as anomalies is not going to be helpful, so regardless of the metrics, the OCSVM will be disqualified for use on the data from the CL-VAE.

The results of these runs are summarized in table 4.2. Isolation Forest is the best algorithm for our data in this case. The AUC scores for all the algorithms are really close for the CL-VAE, but as discussed earlier, Isolation Forest has some other advantages over the other algorithms, which is giving further merit to our choice of best algorithm. All of the tried algorithms perform better on the CL-VAE dataset than the VAE-dataset. Which is another argument of using CL-VAE over the regular VAE. The data simply seems easier

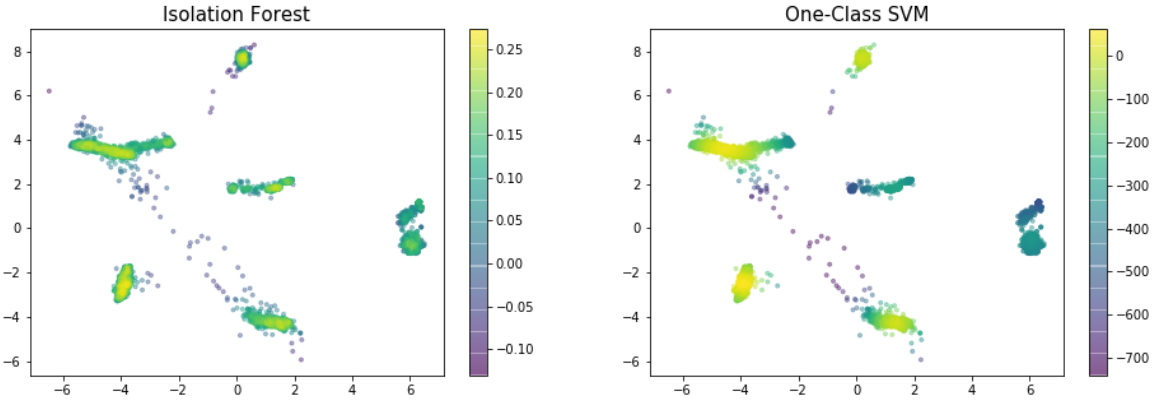


Figure 4.17: A comparison with an optimized Isolation Forest prediction and the OCSVM with `gamma='auto'`. While Isolation Forest gives every point a score in the range [-0.10, 0.25], the OCSVM gives scores as low as -700. Even entire clusters are being classified as anomalous. All the points of the center cluster and the right cluster gets scores in range of [-400, -200]. This is quite strange as this data should be more easy to fit for an anomaly detection algorithm, but the OCSVM might simply underfit with this setting.

to fit for anomaly detection algorithms. However, OCSVM seemed to struggle a bit with this dataset, likely because it was either over or underfitting.

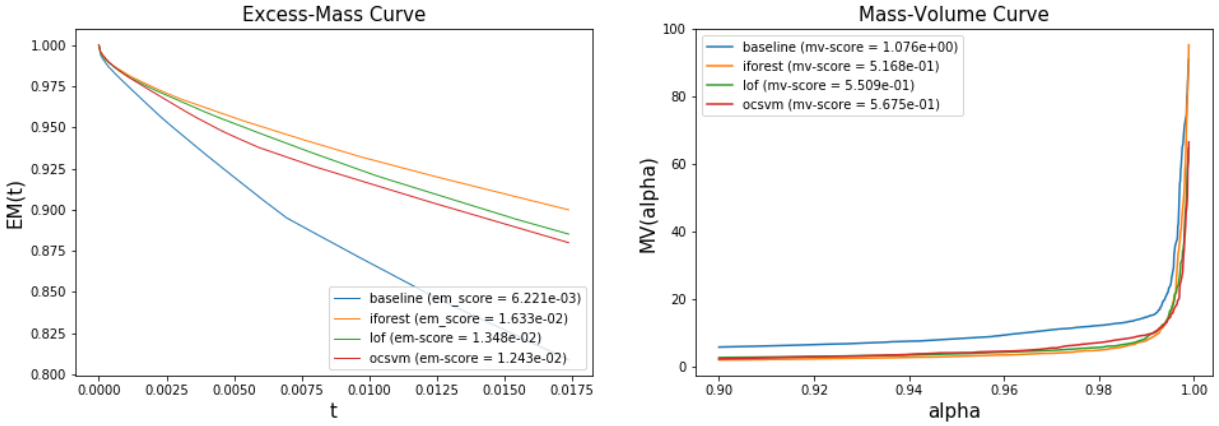


Figure 4.18: The EM and MV curves for different anomaly detection algorithms on the latent space of the CL-VAE, now using (close to) optimal parameters. This was found at `gamma`=20 for the OCSVM, at minPts=200 for LOF and close to 0.01 for Isolation Forest. Isolation Forest is the strongest candidate here, not only because it get's the best scores with very smooth curves, but because it is also very robust to hyper parameter change.

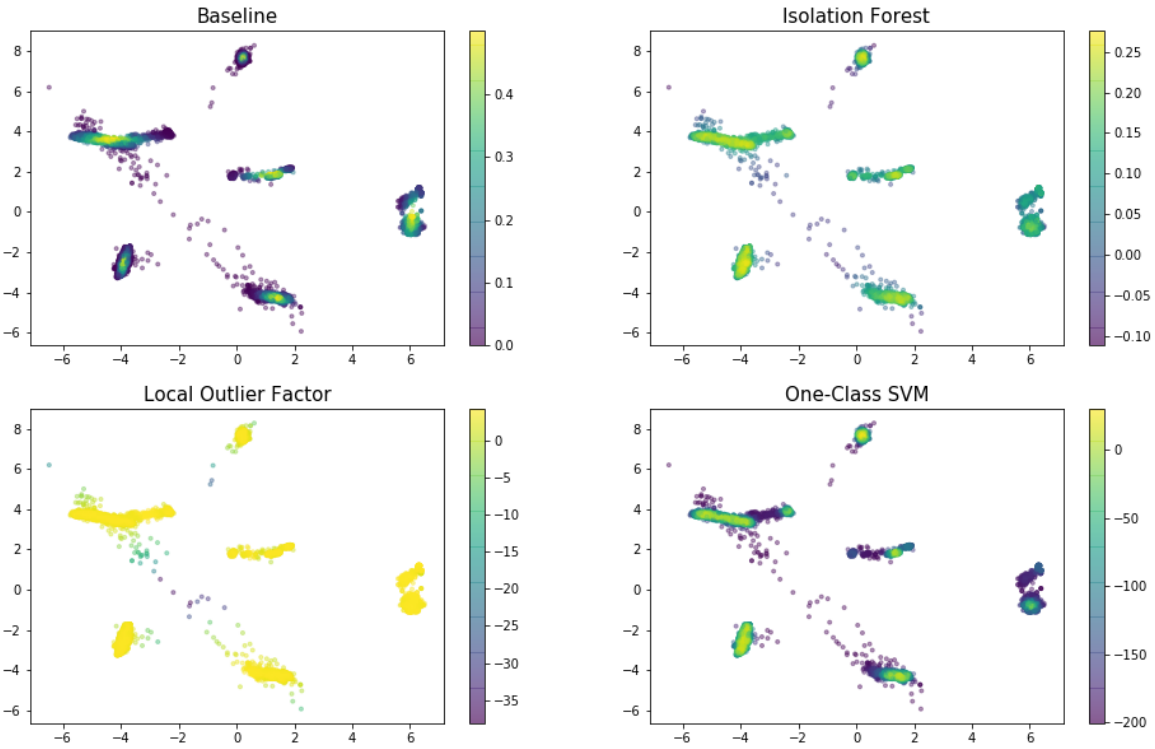


Figure 4.19: The anomaly prediction on the latent space of the CL-VAE comparing all the different algorithms with hyper parameters as specified in Figure 4.18. Even though all of the algorithms have good metrics with these settings, we get a clearer picture of what's going on by looking at this figure. Clearly, the OCSVM is scoring anomalies too extreme and half of some clusters also get bad scores. Having so many anomalies is not going to be helpful, so we can't use it. LOF is perhaps on the other end of the spectrum, too generous with labeling points as normal. While points that are clearly outliers are scored very low in comparison with surrounding points. Also, little or no points in some clusters are being categorized as anomalous. Isolation Forest seems to be the best candidate as it has a reasonable range in where it classifies anomalies vs normal points and also doesn't classify any entire cluster as either anomalous or normal.

Latent Space	Baseline		Isolation Forest		LOF		OCSVM	
VAE	3.592e-03	1.924	4.648e-03	1.362	3.587e-03	1.736	**5.903e-03**	**1.053**
CL-VAE	6.221e-03	1.076	**1.633e-02**	**0.5168**	1.348e-02	0.5509	1.243e-03	0.567

Table 4.2: Summary on the results of using different anomaly detection algorithms on our two latent spaces using the best settings found in Figures 4.15 and 4.19. The left column corresponds to the AUC of EM and the right to the AUC of MV. For the VAE it was the OCVSM but it tended to overfit or underfit the data of the CL-VAE. On this latent space, Isolation Forest performed the best and will be used going forward. All algorithms perform better on the latent space of the CL-VAE, giving merit to our proposed method.

Going forward, we will use the latent space of the CL-VAE and an Isolation Forest with contamination rate 0.01 as a default anomaly detection algorithm. Points that it classifies as anomalous will be considered as known anomalies in the next section and therefore removed in order to focus on finding anomalies inside clusters.

Supervised

When we have fitted the EM-clustering on the training data and want to monitor what goes on in the latent space we can use the V-score again. This is because we can predict the label of a new trade that comes in using the fitted EM-clustering model (a Gaussian Mixture Model). Then, simply feeding the V-score algorithm your predicted cluster labels and their classes (traders) will give us a V-score that we can monitor. If that prediction is correct in the sense that the trader that did this new trade end up in a cluster such that the V-score is retained, nothing will happen. It will just stay at approximately the same level as the training score.

However, if we randomly select trades in the test data and label them as other traders than actually did them, let's call these "strange trades", the V-score of the test data goes down fast. Figure 4.20 shows the collapse of the V-score as the number of strange trades increase. In this data (test data) we have 20 000+ trades and even with 200 "strange" ones we get a decrease in V-score from 0.81 to 0.71. This means that this curve can be used to monitor if trades that would otherwise be considered normal, but made by the wrong trader, would start to appear.

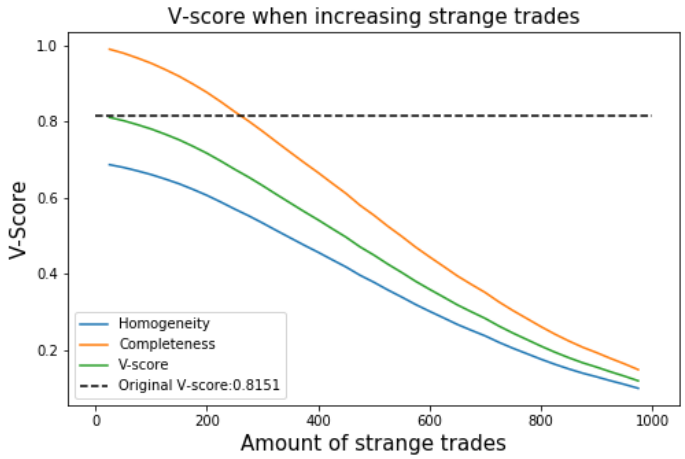


Figure 4.20: The V-score curve. When trades are being categorized as part of an improbable cluster, the completeness and homogeneity will drop simultaneously. This was achieved by selecting a trader in a cluster, then randomly drawing points made by other traders in other clusters and labeling them as being done by the first trader. Doing this for each trader, this simulates anomalies that can occur if a trader suddenly starts trading like someone else.

Because of the shown stability of EM-clustering on our data we can be more confident that the V-score is not going to be changed randomly when normal trading is occurring. Therefore, when it suddenly drops it would give an indication that some anomalous be-

havior is going on in the latent space and could be seen as an "early warning". In that case it is necessary to be able to catch this anomalous behavior.

	z1	z2	traderName	anomaly	cluster
0	1.427273	1.823972	8.0	0.0	2.0
1	1.818110	-4.207020	9.0	0.0	4.0
2	-3.869929	-2.893602	1.0	0.0	1.0
3	-3.922679	3.477970	4.0	0.0	3.0

Table 4.3: The first 4 lines of the test data set that we will use to validate the anomalies that are found. Categorical columns will of course we one hot encoded if they are used right away in the Isolation Forest. The anomaly-column is our target, but as Isolation Forest is an unsupervised algorithm it will only be used for validation.

Since we chose these strange trades myself (randomly) we can label them as anomalous (0 or 1) and train an algorithm on finding them. The case of 600 strange trades was picked (100 per cluster) and with running the regular Isolation Forest on it 10 were immediately selected so there's 590 left within clusters. Yet, this is only 2.891% of the total dataset of 20 406 trades. Meaning that the dataset is quite imbalanced, just like most other supervised anomaly detection problems. Because of this imbalance we will use the precision-recall metric rather than the ROC.

In this case we have a dataset of 5 parameters as presented in Table 4.3. We will use an Isolation Forest to try and capture these anomalies which is an unsupervised algorithm, so this will only be supervised in the sense that we can confirm its predictions. To simulate an actual situation, we want to try and can catch the anomalies by training on the "normal" training data and then predict on the data with the miss-labeled trades. The 2 categorical features `traderName` and `cluster` of the training data was one-hot encoded for improved performance.

Trying to fit one Isolation Forest on the training data, predicting on the test data and then validating with the target we get a rather disappointing F1-score of 0.4078 as seen in Table 4.4. However, the F1-score is not the most important thing, what really matters is the dynamics of precision and recall. Unfortunately precision is rather bad at 0.3796. This means that 37.96% of the classified anomalies were correctly categorized as such; we have more false positives than true positives. This is achieved with a rather bad recall as well of 0.4407, meaning that only 44.07% of the real anomalies were correctly classified.

Thinking of other ways to approach this we came up with the idea that we can "condition" on the trader like we did for the CL-VAE. In this case, this just means to train one Isolation Forest per trader so every single Isolation Forest just has to learn the dynamics of one trader. It might seem like an inefficient solution, but it actually has a slightly shorter run time than a single Isolation Forest as seen in table 4.4. This isn't so surprising for anyone that has worked with decision tree based algorithms before, as they tend to scale poorly with growing amounts of data. Dividing the work-load up like this will decrease training and prediction times as more data becomes available too. Also, we don't have to one-hot encode as the conditioning solves this for us. We are not feeding it any categorical information, only the two numeric columns z1 and z2, meaning less complexity.

Algorithm	F1-score	Recall	Precision	Anomaly rate	Run time (s)
Single IF	0.4078	0.4407	0.3796	3.357%	10.76
Multiple IF	0.5982	0.8051	0.4760	4.891 %	7.443

Table 4.4: Results of running a single Isolation Forest on the dataset compared with using multiple ones (one per trader). All Isolation Forests are run with a contamination rate of 0.0289, as it is the rate of anomalies in the test data. The performance is quite bad with a single Isolation Forest both in recall and precision. However the high recall when using multiple Isolation Forests is promising. This is achieved with a slightly shorter run time and a slightly higher rate of data classified as anomalies at 4.491%. Of course, we would like this number to be closer to the actual level of 2.891%.

Using multiple Isolation Forests didn't just bring down the run time, it achieved a much better recall of 0.8051 and a slightly better precision and F1-score at 0.4760 and 0.5982 respectively. Although, it did so at a slight cost of anomaly rate. Remember, the real rate of anomalies in this data is 2.891% and with the multiple IF we get 4.891% and 3.357% with the single IF. This is not so good because we want to report anomalous behavior to someone with domain knowledge so that person can investigate further. It is not going to be a very efficient system if we predict that there are almost double the amount of anomalies than there actually are. This is also a possible reason for the low precision scores in both cases, as too many trades are being labeled anomalous.

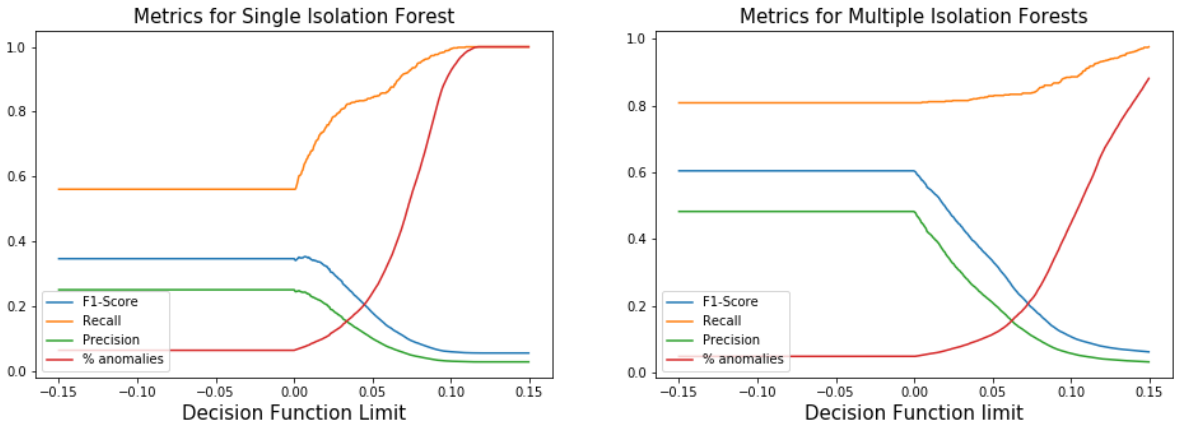


Figure 4.21: Plots describing how our metrics change with the threshold of the decision function for both a single Isolation Forest and one Isolation Forest per trader. The recall get's better in both cases at an enormous cost in precision as the anomaly rate goes through the roof. Clearly, changing the limit is not going to help us here. Especially because no measurable difference was found in decreasing the rate either.

A natural way of improving on this is to change the threshold. In a regular classifier this would work by varying the probability score given to each class label. Perhaps the algorithm takes all points above 50% and labels them as anomalies, then changing this would clearly affect precision and recall. Isolation Forest uses instead a decision function that gives a score in the range of [-1,1], the smaller the more likely a point is an anomaly and in its standard setting it will label everything below 0 as an anomaly. In Figure 4.21 we see what happens when we change this parameter for both a single IF and the multiple IF. As is clear, when the threshold goes over 0 the recall starts to increase, but

the precision goes down as the total anomaly rate goes up fast. No gain was seen in decreasing the threshold and increasing it will just mean more miss-labeled trades.

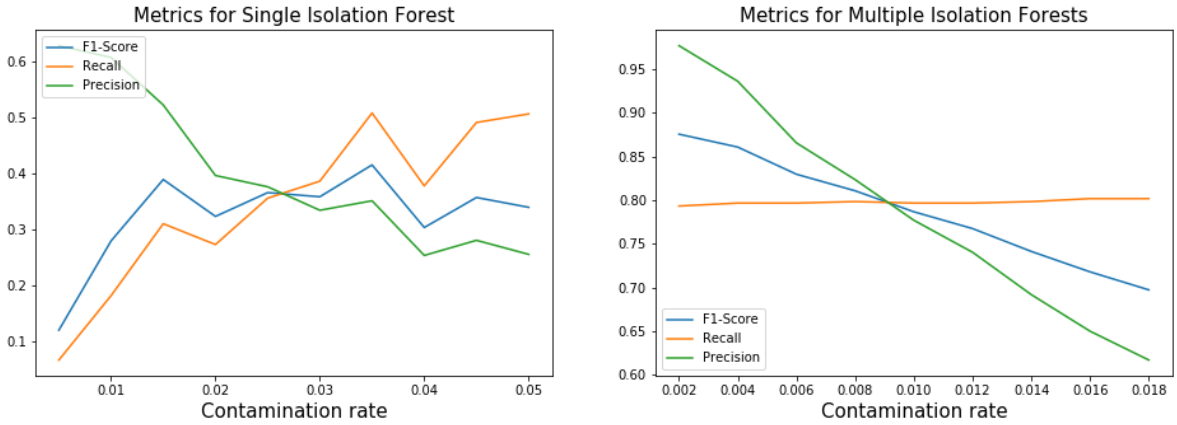


Figure 4.22: Plots describing how our metrics change with contamination rate for both a single Isolation Forest and one per trader. A single Isolation Forest is not performing so well on this data, likely because the anomalies are inside other clusters, which not really what the algorithm was designed to do. Letting each trader have their own Isolation Forest we get a much better performance as the problem setting is more akin to what an Isolation Forest was designed for. At the intersection between the precision and recall curves in these two cases the anomaly rate is very close to the actual rate of 2.891%.

Contamination	F1-score	Recall	Precision	Anomaly rate (%)
0.002	0.8756	0.7932	0.977	0.0235
0.004	0.8608	0.7966	0.9363	0.0246
0.006	0.8297	0.7966	0.8656	0.0266
0.008	0.8107	0.7983	0.8234	0.028
0.01	0.7866	0.7966	0.7769	0.0296
0.012	0.7673	0.7966	0.7402	0.0311
0.014	0.7411	0.7983	0.6916	0.0334
0.016	0.7178	0.8017	0.6497	0.0357
0.018	0.6971	0.8017	0.6167	0.0376

Table 4.5: Metrics when using multiple Isolation Forests (one per trader). Note that a low contamination rate will give very high precision without losing much recall. Here, the contamination rate is set globally, i.e. the same for all the Isolation Forests.

Instead, let's investigate what happens when we change the hyper parameter *contamination rate*. Letting this parameter vary, we can study how it affects the precision and recall in Figure 4.22. Remember that in the case of the multiple Isolation Forests, we're setting the contamination rate to the same value for all Isolation Forests. Interestingly, we find that the real anomaly rate is very close to the rate of classified anomalies in the intersection between the precision and recall curves. After the intersection point, recall continues to rise for the single Isolation Forest but not to a high level before the precision starts dropping.

Using multiple Isolation Forests we can get a slightly higher recall after the intersection point, but at a high cost of precision. What is interesting is that the recall is still quite high, even for low values of contamination rate. This means that we can have a very high precision without losing that much recall using low contamination rates as seen in Table 4.5. However, like many things in life this is a trade off. Do we want to find all anomalies or do we want the ones that are found to be true? Because we lose so little recall for a huge increase in precision by lowering the contamination rate we would suggest such a setting.

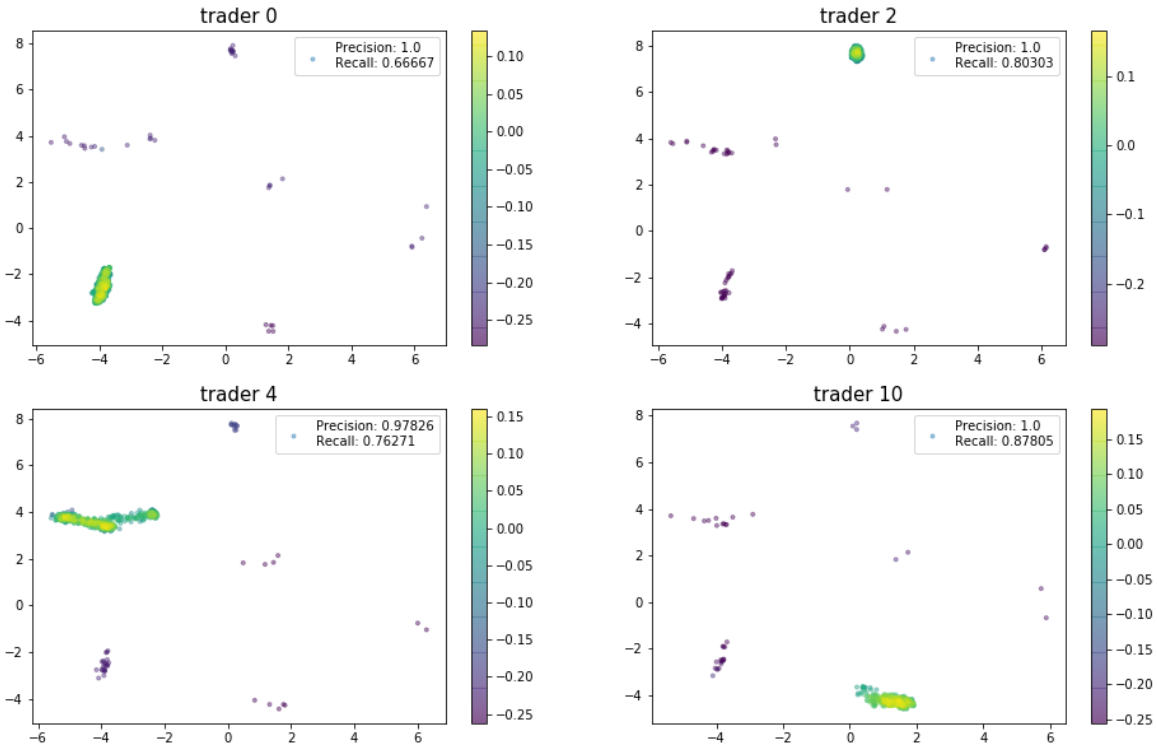


Figure 4.23: Some examples of different traders and their precision and recall rates using a contamination rate of 0.002. Some traders get perfect precision but low recall, like the case of trader 0. Trader 2 has a little better recall score with still perfect precision but trader 10 has an even better recall score at perfect precision. One contributing factor to low recall rates might be because some anomalies form clusters of themselves. It might be hard to see but the top 'anomaly-cluster' for trader 4 has a slight blue shade, suggesting it is close to being considered normal.

Studying some different traders and their precision and recall rates found by using multiple Isolation Forests we see that they actually are quite different from trader to trader as seen in Figure 4.23. This makes sense because we set the contamination rate globally. We do not consider the individual precision and recall rate of each Isolation Forest but instead try and optimize the overall rates. Optimizing these rates individually would likely improve performance.

Reconsidering this experiment it's clear that it is not optimal as the anomalies will likely cluster because we pick them out of existing clusters. If points are too well grouped together, Isolation Forest might think that they are forming their own cluster even though it has previously only seen the cluster formed by the trader we want to model. This is clearly a limitation in the experiment. However, it does show that we can catch *most* trades that would be considered normal if they came from another trader than you're trying to model.

4.3 Reconstruction loss

Obviously there are some points that have lower reconstructive power than others. According to this approach then, we'd take some top percentile of reconstruction losses to be considered anomalous. For the purpose of demonstration the reconstruction loss was computed for every point in the test data which was used to color the points in Figure 4.24. As is clear from this figure, the points with the worst reconstruction error are mostly centered in the middle of latent space while the best reconstruction errors are on the fringes.

At first, this might seem counter intuitive. Closer to the center means closer to the mean of the underlying Gaussian we're trying to force the points towards, therefore those points should be considered "normal" right? Likewise, a point out on the tail of the distribution should be considered anomalous if anything. Remembering Figure 2.7 we know that greater separation leads to better reconstruction. Therefore, a small, tightly knit cluster of points far away from anything else should have good reconstructive power. Likewise, a point close to the center with many overlapping classes should have bad reconstructive power. Therefore, this actually makes sense from the models perspective.

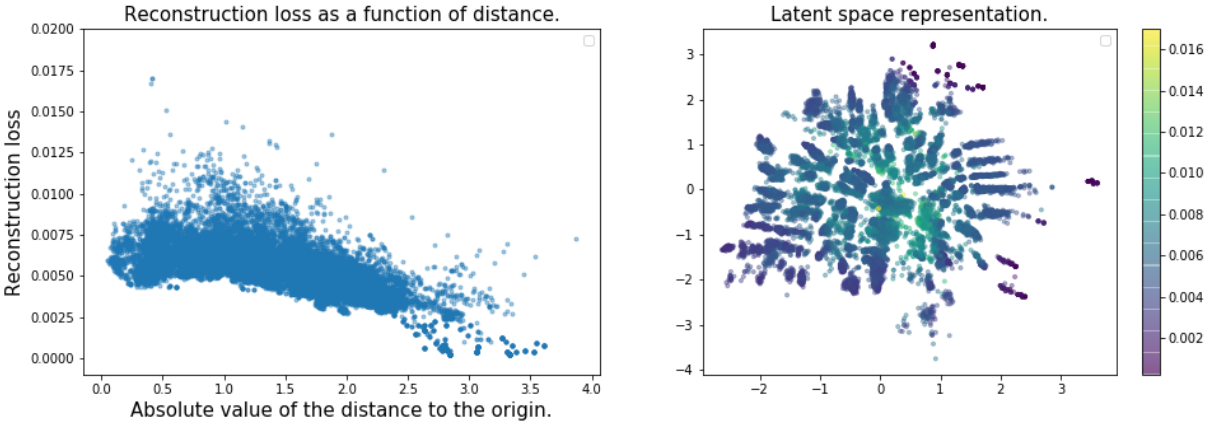


Figure 4.24: An example of a latent space of the VAE. These two plots describe how the reconstruction loss changes as a function of distance from the origin. The coloring in the right plot is based on each points reconstruction loss. Generally, the further from the origin a point is, the smaller its reconstruction loss. This makes sense considering Figure 2.7 as class overlap leads to bad reconstruction. Near the origin there will be much more class overlap than on the fringes.

Although this raises some doubt on how well reconstruction loss actually measures outliers

for our purposes with a normal VAE. If we simply have many overlapping classes, with bad reconstruction loss, it could simply mean that it's a common type of trade that many traders partake in. On the other hand a less dense region, far away from other regions would make more sense to be anomalous but it will have a better reconstruction loss. What we would like is a modification to the VAE such the negative dependence on distance of the reconstruction loss is reversed. Doing so will make the anomalies that are found make sense.

It is not clear that the VAE is particularly good at finding clusters either. Considering Figure 4.1 we find separation between classes but it is unclear what this separation mean. As the KL-divergence is trying to force the points to be close to the Gaussian normal prior, it would be reasonable to assume that points on the fringes seen from the origin are improbable. It is unclear however how we can distinguish improbable points between clusters.

Using the same dataset but instead applying the CL-VAE, we get Figure 4.25. Now, the absolute distance is computed in comparison to the mean of the cluster a point is belonging to. The data suggests a slighter dependency on the reconstruction loss on the absolute value to the origin. However it is not clear that the phenomenon is completely gone. There's some local dependence that could tend negative, likely due to some remaining class overlap. Overall though, there does not appear to be such a clear dependence anymore.

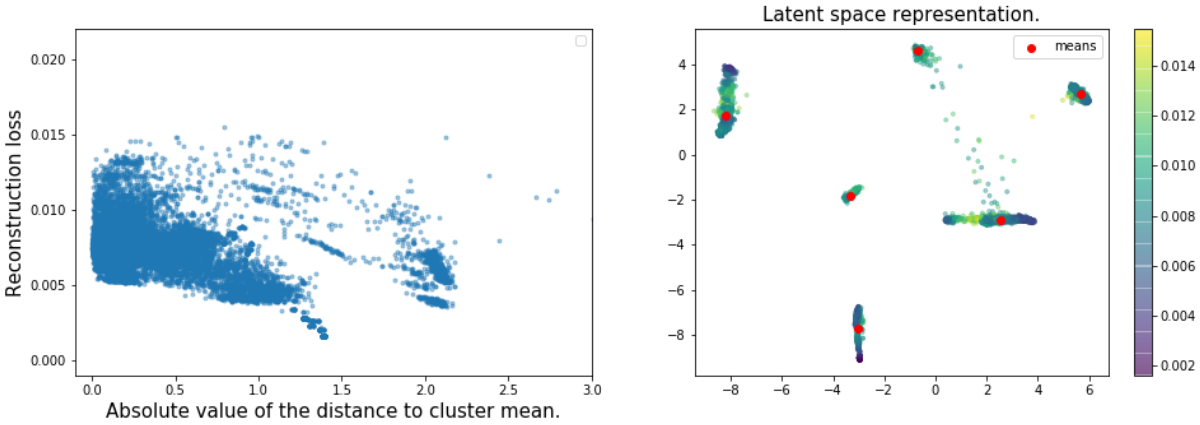


Figure 4.25: An example of a latent space of the CL-VAE. These two plots describe how the reconstruction loss changes as a function of distance from the mean of the clusters formed by using a Gaussian Mixture Model. We do not have such a strong negative dependence between reconstruction loss distance from the mean of a cluster as we did for the VAE. Likely, there's still some class overlap, but the problem is reduced compared with Figure 4.24.

What does this mean for the tug-of-war effect? Recalling Figure 2.7, instead of having one large blue blob representing the normal distribution, all the different traders have their own blue blob. What this means is that there is no longer necessarily a tug-of-war effect on the loss function or at least, it is reduced.

If we want to use the reconstruction loss as a measure for anomaly detection we should try and isolate each class as much as possible. Because remember, overlap is the enemy of reconstruction. If we simply have one Gaussian as a prior, the reconstruction loss gets

worse as we come close to the origin. This is because points on the tail of the distribution are more well defined and therefore have less ambiguity in what they represent. Corollary, closer to the origin, there's severe class overlap, which makes the position of the points more ambiguous, which therefore leads to higher reconstruction loss.

Using a mixture of Gaussians as your priors and letting some known labels condition them subverts this problem somewhat. Already by simply conditioning on the traders name we see the reverse start happening. Points on the tails of the distribution are no longer the best and points in the densest regions are no longer the worst. However, there's still class overlap in this case. This is because each trader can have many classes themselves. If we were to continue conditioning on labels such as portfolio, currency, instrument etc. we would at some point reach even better clusters that actually represent the data in the latent space. This would create many Gaussians, as each combination of trader-portfolio-currency etc would get their own Gaussian. Doing so would require tedious evaluation of each label and is outside the scope of this thesis.

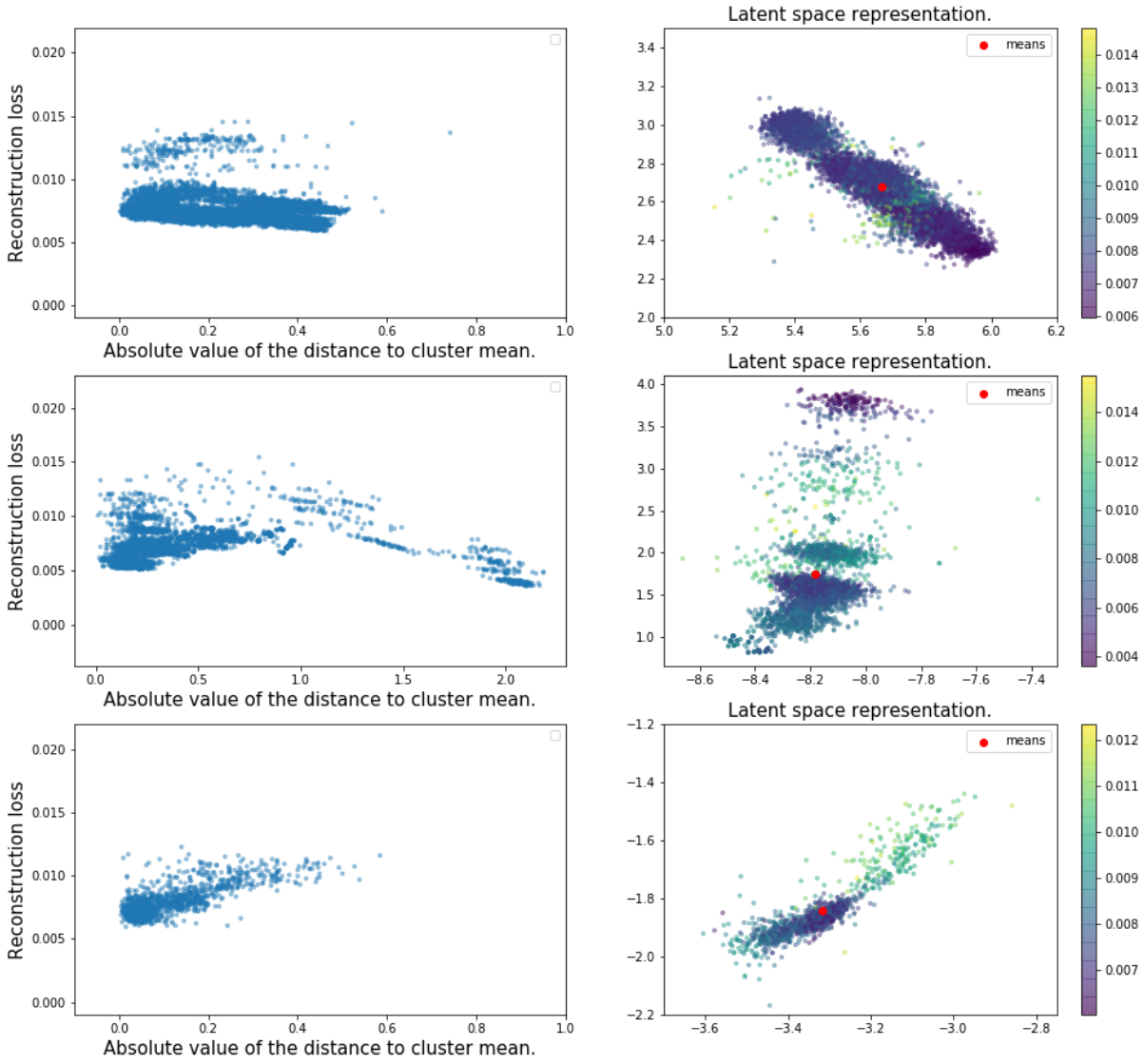


Figure 4.26: In this figure we see 3 examples of latent space representations of clusters and their corresponding reconstruction losses. None of these have a *clear* negative dependence of reconstruction loss on absolute distance to the mean of the cluster. The top cluster seem to be neutral on this point. Perhaps because it represents cluster 0 that has the two traders with the most trades in them. Likely there are more classes among them that we do not study yet. The middle plots describe a cluster with three traders in it so the dependence is not very clear. Locally in the left part of the plot there's a positive dependence for reconstruction loss on absolute distance to the mean but the other way around on the right side. The bottom two plots describe a cluster with only 1 trader in it, here there's a clear positive dependence! In an ideal world, all clusters would look like this, but class overlap is a still common in our data.

Chapter 5

Discussion

Fundamentally what the CL-VAE does is transforming some multi-dimensional data to a Gaussian space where it is easier to work with. We are not aware of another model that does this quite as simply and straightforward as CL-VAE so we think that this is a contribution. This kind of pre-processing could be applied to any type of categorical or numerical data that have some pre-determined label. In some such cases clustering and anomaly detection can prove really challenging so this approach could help out a lot.

Reliability of the results

As our exploration showed of the different clusters that was formed, similar traders did end up in the same clusters using the CL-VAE. Also the latent space picks out some feature that can be varied in order to find traders of different kinds. This was shown by noting that traders in bonds ended up on one side of the latent space and traders in swaps and future/forwards on the other side. However, what exactly is learned, i.e. what the axes in the latent space really represent is unclear. While it would make the model even more understandable by knowing this, it is not necessary to find outliers in a statistical sense.

Because we were not able to get some dataset that could confirm our anomalies we had to make two workarounds. The first being the EM-MV measure. This measure and its associated curves are of course quite difficult to interpret even though (and maybe because) they have a solid theoretical backing. Although they have been shown to agree with the ROC and PR measures on some famous datasets [35] it doesn't mean that they will do so for our dataset. However, seeing them as an optimization target rather than some ground truth can still be useful.

As we showed in the section on unsupervised anomaly detection some strange behaviors can be found using the EM-MV curves. We haven't found any other papers that showed clear signs of over or underfitting in the curves so this might be a new find. As we discussed in that section, one should not blindly optimize the area under the curve without taking the shape of the curves or your predictions on the latent space into consideration.

The second workaround was the fabricated experiment in the supervised setting. These results could at least be evaluated using regular measures but whether or not the setup

makes sense is a different question. We did try inputting noise instead to see if it was detectable, which it clearly was. However, in discussion with Richard Henricsson, we decided to go this way instead. Mainly because it's difficult to interpret what noise in the latent space actually means. Whether or not some noteworthy anomalous behavior would accurately be modeled by using real trades we do not know. But we did show that using the V-score curve in Figure 4.20 one can monitor if it starts occurring.

The key issue of anomaly detection is that we have no idea if the trades that the algorithm classifies as anomalous really is an anomaly or not. It is just finding mathematically anomalous behavior, which is really what this thesis is about. Therefore, this work should not be seen as some "perfect anomaly detector" but rather a contributing, alternative way of doing anomaly detection. Obviously if this were to be tested at the bank it would have to be evaluated in comparison with existing systems.

Interpretability of the latent space

Comparing the latent spaces of a regular VAE and the CL-VAE it's quite clear that the CL-VAE is easier to understand, simply but not exclusively because of the shapes of the clusters. As shown in the section on the reconstruction loss, the resulting latent space will have a *more* natural way of determining where the reconstruction loss is better and where it is worse. As denser regions, close to the mean of the cluster tend to have better reconstructive power - although with many exceptions. At least, it's not the reverse, as it is for the VAE. As stated earlier, conditioning on more features would likely separate the classes further to get an even more interpretable latent space.

Alternative Application

One promising application of this could be in data quality. Being able to create distributions in an understandable space of multi-dimensional data you can evaluate when improbable measurements are coming in to your data pipelines. In one sense this is close to another type of autoencoder called a denoising autoencoder [44], but the CL-VAE would have a more interpretable latent space where you can apply clustering and anomaly detection. Also, we have not used any of the generative aspects of this model in this thesis but that would be very interesting to explore further.

The CL-VAE (and the regular VAE) could be used to simplify multi dimensional time-series prediction problems. Because you're fundamentally estimating some stochastic process and making it easier to understand, you can do your time-series prediction on the data coming from the latent space and then transform it back to the original space.

Improvements and Future Research

For ease of visualization the latent space was chosen to a simple 2D-space. However, the model is not limited to just 2 dimensions. The latent space can be any dimension of your choosing, and since both EM-clustering and Isolation Forest works for higher dimensional data, you would be able to do the same types of analysis even in higher dimensions. Likely, the problem of class overlap would be reduced as you introduce more dimensions.

Using other prior distributions than Gaussians would be interesting as it may capture the structure better for some data. Although the performance of EM-clustering and Isolation Forest may decline as they both rely heavily on data that is close to Gaussian. This was one of the major reasons that we tried to make the latent space "more Gaussian" in the first place, as these algorithms are very strong on the right kind of data.

Including some time axis in order to see the probability of a *sequence* of trades would also be interesting. That would be possible with the CL-VAE and perhaps you could do clustering and anomaly detection there too.

5.1 Concluding Remarks

As a conclusion, let's revisit the questions that we posed after the introduction and see if we answered them fully.

1. ***Can Variational Autoencoders be used to pre-process the data for clustering and anomaly detection?***
   - Yes, we have shown that Variational Autoencoders in two different forms can indeed be used to prepare our dataset of trades for clustering and anomaly detection. The latent space that was created with both the VAE and the CL-VAE was continuous and showed structures that could be analyzed using these methods in measurable ways.
2. ***What kind of Variational Autoencoder is most successful at this job?***
   - The CL-VAE clearly performs the best on the tests that we presented. An argument can be made that the VAE showed more structure but the interpretability of the clusters in the CL-VAE and the ability to customize the latent space are major advantages.
3. ***Can a system be developed using this method to perform clustering and anomaly detection automatically on financial trading data?***
   - Yes with caution. We showed that you can detect anomalies and categories in trading using both supervised and unsupervised methods. However we can not claim that the trades that were considered anomalous by our algorithms are true before cross referencing with some known anomalies. That being said, there are many results in this thesis that suggests that such a system can be developed.

Bibliography

[1] *On Lines and Planes of Closest Fit to Systems of Points in Space, K. Pearson, (1901)*

[2] *Auto-Encoding Variational Bayes, P. Kingma, M. Welling, (2014)*

[3] *A Logical Calculus of Ideas Immanent in Nervous Activity, W. McCulloch and W. Pitts, (1943)*

[4] *Handson Machine Learning with Scikit-Learn and Tensorflow, Aurélien Géron, (2017)*

[5] *The Perceptrion: A Probabilistic Model For Information Storage And Organization In The Brain, F. Rosenblatt, (1958)*

[6] *Learning representations by back-propagating errors, David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams, (1986)*

[7] *Deep Learning With Python, F. Chollet, (2017)*

[8] *Variational auto-encoders do not train complex generative models, D. Train, (2016)*

`http://dustintran.com/blog/variational-auto-encoders-do-not-train-complex-generative-models`

[9] *Lecture 13, Generative Models, Serena Yeoung et al, (2017*

`https://www.youtube.com/watch?v=5WoItGTWV54`

[10] *Derivations For Linear Algebra and Optimization, J. Duchi*

[11] *A Tutorial on Variational Autoencoders with a Concise Keras Implementation, Louis Tiao, (2018)*
`https://tiao.io/post/tutorial-on-variational-autoencoders-with-a-concise-keras-implementation/`

[12] *The wake-sleep algorithm for unsupervised neural networks, Hinton et al, (1995)*

[13] *Lecture 22 Variational Autoencoders, Russ Salakhutdinov, (2017)*

[14] *Learning Hierarchical Features from Generative Models, S. Zhao, J. Song, S. Ermon, (2017)*

[15] *Density Estimation: Variational Autoencoders, R. Shu, (2018)*

`http://ruishu.io/2018/03/14/vae/`

[16] *Generative Adverserial Nets, Ian J. Goodfellow, (2014)*

[17] *Variational autoencoders do not train complex generative models, Dustin Tran, (2016)*

[18] *Autoencoding beyond pixels using a learned similarity metric, A. B. L. Larsen, S. K. Sønderby, H. Larochelle, O. Winther (2016)*

[19] *Adverserial Autoencoder, I. Goodfellow et al. (2015)*

[20] *GAN-Why it is so hard to train Generative Adversarial Networks, Jonathan Hui, (2018)*

`https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b`

[21] *Least squares quantization in PCM, S. P. Loyd (1982)*

[22] *Maximum Likelihood from Incomplete Data via the EM Algorithm, A. Dempster, N. Laird, and D. Rubin (1977)*

[23] *Gaussian Mixture Model, John McGonagle, Geoff Pilling, Vincent Tembo*

[24] *Isolation Forest, F. Liu, K. Ting, and Z. Zhou,(2008)*

[25] *Classification and Regression by randomForest. R News 2(3), A. Liaw and M. Wiener (2002)*

[26] *Anomaly Detection: Algorithms, Explanations, Applications, T. Dietterich (2018)*

`https://www.youtube.com/watch?v=12Xq9OLdQwQ&t=290s`

[27] *Unsupervised Anomaly Detection with Isolation Forest, E. Sharova, (2018)*

`https://www.youtube.com/watch?v=5p8B2Ikcw-k&t=476s`

[28] *LOF: Identifying Density-Based Local Outliers, M. Breunig et al, (2000)*

[29] *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, M. Ester, H. Kriegel, J. Sander, X. Xu*

[30] *Support Vector Method for Novelty Detection, B. Schölkopf et al. (2000)*

[31] *https://scikit-learn.org/stable/modules/svm.html*

[32] *V-Measure: A conditional entropy-based external cluster evaluation measure A. Rosenberg and J. Hirschberg, (2007)*

[33] *Anomaly Detection using Autoencoders in High Performance Computing Systems, A. Borghesi et al.,(2018)*

[34] *GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection Q. Nguyen et al., (2019)*

[35] *How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms?, Nicolas Goix, (2016)*

[36] *How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms? (presentation), Nicolas Goix, (2016)*
`https://github.com/ngoix/EMMV_benchmarks`

[37] *On Anomaly Ranking and Excess-Mass Curves, Nicolas Goix, et al., (2015)*

[38] *Su Jianlin, Variational Self-Encoder: One-Step Clustering Scheme, (2018)*

`https://kexue.fm/archives/5887` (Chinese)

[39] *Unsupervised Deep Embedding for Clustering Analysis, J. Xie, R. Girshick, A. Farhad, (2015)*

[40] *Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering, Z. Jiang. Y. Zheng, H. Tan, B. Tang, H. Zhou, (2016)*

[41] *Gaussian Mixture VAE: Lessons In Variational Inference, Generative Models and Deep Nets, R. Shu, (2016)*

`http://ruishu.io/2016/12/25/gmvae/`

[42] *Learning Structured Output Representation using Deep Conditional Generative Models, K. Sohn, X. Yan, H. Lee, (2015)*

[43] *Tutorial on Variational Autoencoders, C. Doersch, (2016)*

[44] *Extracting and Composing Robust Features with Denoising Autoencoders, P. Vincent et al., (2008)*

.1 Appendix A

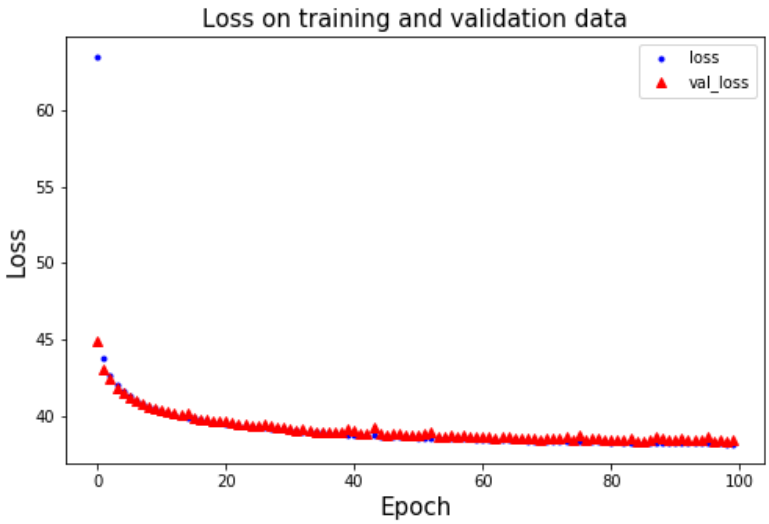


Figure 1: Loss on training and validation data using the VAE. In this case it was trained with 100 epochs and a batch size of 100. Clearly, it's not overfitting.

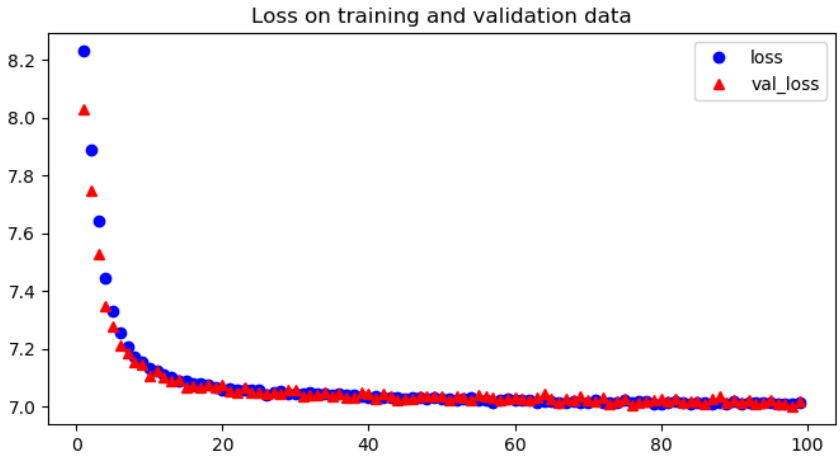


Figure 2: Loss on training and validation data using the CL-VAE. In this case it was trained with 100 epochs and a batch size of 100. Clearly, it's not overfitting.

Master's Theses in Mathematical Sciences 2019:E35
ISSN 1404-6342

LUTFMA-3370-2019

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

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