Anomaly Detection for OTC Derivatives

Daniel Fogelberg

Master's thesis
2025:E54

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Master's Theses in Mathematical Sciences 2025:E54
ISSN 1404-6342

LUTFMS-3522-2025

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

http://www.maths.lu.se/

Abstract

This thesis explores the use of machine learning to detect anomalies in OTC derivatives. The purpose is to evaluate whether machine learning can be used in a new domain within the financial industry and to investigate whether anomaly detection can enhance the portfolio reconciliation process.

The study compares three different approaches to anomaly detection: logistic regression, isolation forest, and denoising autoencoders. The data used consists of real-world trades provided by OSTTRA. As there were no labeled data for anomalous trades, anomalies were created synthetically.

The results show that the isolation forest performs poorly in detecting anomalies in this context. Logistic regression shows some capability but has clear limitations. Denoising autoencoders, on the other hand, demonstrate the strongest performance in detecting anomalies, especially regarding the product classes of the OTC derivatives.

Furthermore, the study explores the use of denoising autoencoders to enhance the portfolio reconciliation process. The results indicate that the model has potential to be used for certain limited use cases, but further research is needed to fully evaluate the potential possibilities and challenges.

The thesis concludes that machine learning, specifically denoising autoencoders, shows potential in detecting anomalies in OTC derivatives. However, the synthetic creation of anomalies and some limitations in the data imply that the conclusions should be interpreted with caution.

Keywords Anomaly Detection, Denoising Autoencoder, Isolation Forest, Logistic Regression, Machine Learning, OTC Derivatives, Portfolio Reconciliation.

Acknowledgement

I would like to express my sincere gratitude to OSTTRA for the opportunity to write my Master's thesis here. A special thanks goes to my supervisor Love Eklund for excellent guidance, but also to the rest of the team for invaluable input and encouragement. This thesis would not have been possible without Filip Tronarp, my supervisor at Lund Institute of Technology, whom I will be forever grateful for guiding me through this process and providing excellent support. Last but not least, I would like to thank my family and friends for supporting me during my years at LTH. It has been a great pleasure.

Contents

Chapter 1

Introduction

Bilateral OTC derivatives are financial contracts traded directly between two parties, as opposed to derivatives traded through a central counterparty or stocks that are traded on a stock exchange. The terms of these trades are agreed between the two parties directly, which enables a more flexible contract that serves their exact needs but at the same time introduces more possible errors. In order to ensure the accuracy of a firm's trade population as compared to its counterpart, portfolio reconciliation plays a crucial role. Such reconciliation is offered by OSTTRA through their triResolve service, which sees a vast majority of all bilateral OTC derivatives globally [1]. Basically, the portfolio reconciliation matches trades from one part to the corresponding trades on their counterpart. As the number of submitted trades is huge, the occurrence of errors in the data is non-negligible. Such errors can arise due to, but not limited to, complex interdependencies between systems and manual handling of trades. This leads to difficulties in being able to match trades and some discrepancies between matched trades can occur. The cause of discrepancies is often hard to find, and an effective way to determine whether a submitted trade is correct or not would significantly streamline this process. This raises the question whether a new approach using machine learning to find erroneous trades would address these challenges, which will be explored in this thesis.

The objective of this thesis is to implement and evaluate machine learning anomaly detection algorithms to detect anomalies in the portfolio reconciliation data.

1.1 Financial Background

1.1.1 OTC Derivatives

Derivatives are financial instruments whose values derive from other, more basic, underlying variables [2]. Often these underlying variables are the prices of traded assets, for instance stocks, but it can be almost any variable, such as the amount of snowfall at a certain location. Derivative contracts can be traded on an exchange where the contracts are standardized. Non-standardized contracts can also be traded directly between two or more parties. The latter are referred to as *over-the-counter* (OTC) derivatives. Due to the flexible nature of these contracts, they can be tailored to match the exact needs of the parties involved in the trade. Derivatives can be used to hedge against risks, such as fluctuations in price for commodities or speculate on movements of the underlying asset, to name a few. Common derivatives include options, swaps, and forwards. A forward contract is an agreement to buy or sell an asset at a certain future time for a certain price. An option can be either a put or call option, where the put option gives the holder the right to sell the underlying asset by a certain date for a certain price and the call option gives the holder the right to buy the underlying asset by a certain date for a certain price. A swap is an agreement between two parties to exchange a series of cash flows over a specified period in the future, based on different underlying assets or rates. Although these contracts are defined differently, there are some common parameters for most contracts such as which currencies are exchanged. [2]

1.1.2 Portfolio Reconciliation

A key aspect post-trade for OTC derivatives is the process of reconciling portfolios. This is done to ensure an accurate and common view of the trades between counterparties. Portfolio reconciliation is a crucial tool to reduce certain risks and is considered good market practice [3].

1.2 Problem

1.2.1 Original Problem and Definition

Anomaly detection refers to the problem of finding patterns in the data that do not conform to the expected behavior [4]. Those non-conforming patterns are most often referred to as anomalies or outliers. Examples where anomaly detection is widely used are fraud detection for credit cards, intrusion detection for cyber security, insurance, or health care [4]. Many approaches to anomaly detection are domain-specific, while others are more general. The unique position of OSTTRA, as a central part of the OTC derivatives market, enables a unique possibility to explore anomaly detection within a new domain. In addition, the structure of the data is tabular with a mix of categorical and numerical values, further raising the question of which approach could be most effective.

1.2.2 Research Questions

The aim of this thesis is to evaluate three different approaches to anomaly detection for OTC derivatives. Furthermore, to explore whether the most suitable approach could detect such anomalies that cause discrepancies in the portfolio reconciliation process. In order to do so, the following questions are asked:

- Can logistic regression, isolation forest, or autoencoders be used to detect anomalies among OTC derivatives?
- Can anomaly detection be used to enhance the portfolio reconciliation process? In particular, can trades that cause discrepancies during portfolio reconciliation be found using anomaly detection?

1.3 Purpose

The purpose of this thesis is to evaluate anomaly detection in a new domain. By doing so, two purposes are served. The first is to contribute to general knowledge on how machine learning can be used in the financial industry to streamline certain processes. Secondly, to enable an anomaly detection framework for OTC derivatives, from which OSTTRA could further enhance

their portfolio reconciliation process. Such enhancements include detecting trades that would cause discrepancies in the matching and, where possible, presenting suitable corrections for such trades.

Chapter 2

Theoretical Background and Litterature Review

2.1 Anomaly Detection

Anomaly detection, as previously mentioned, refers to the problem of finding patterns in the data that do not conform to the expected behavior; see fig. 1. Those patterns, called anomalies, can be introduced into the data for a number of reasons. It can be either on purpose as for e.g. credit card fraud or unintended as for e.g. a system breakdown. In its most simple form, anomaly detection can be performed by defining a "normal region" and then declare data outside this region as anomalies. However, this seemingly simple approach comes with a number of challenges, such as defining what a "normal" region is, and the exact notion of an anomaly can be highly domain-specific. [4]

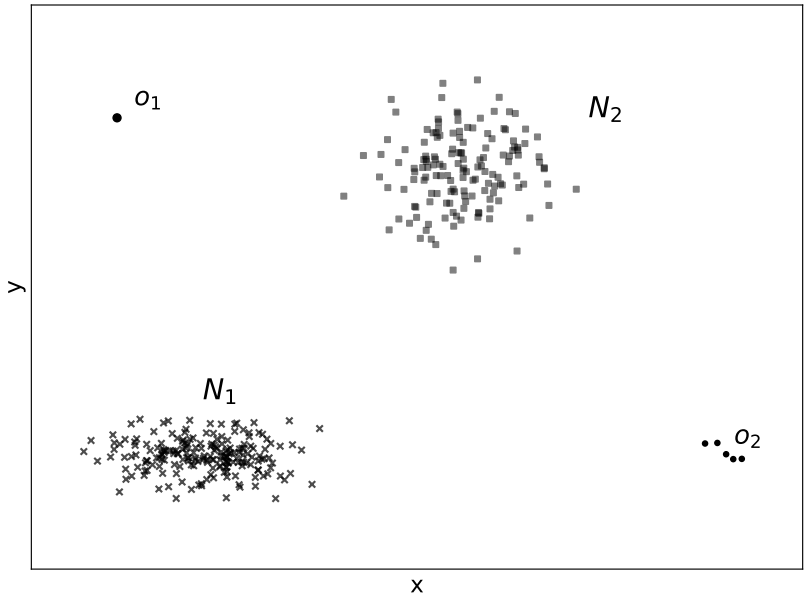


Figure 1: Example of anomalies in a 2-dimensional data. N_1 and N_2 are clusters of normal points whereas O_1 and O_2 are anomalies.

In general, anomalies can be classified into three different categories [4]. Point anomalies refer to individual data points that can be considered anomalous with respect to the rest of the data. Contextual anomalies refer to data points that are anomalous in a certain context but normal in others. Collective anomalies refer to a collection of data points that are anomalous with respect to the entire data set. If it is known whether a certain data point is an anomaly or not, the data point can be labeled either normal or anomalous. Obtaining a labeled data set can often be challenging. The labeling is often done manually by human experts, and the nature of the anomalies are often dynamic and change over time. Based on the availability of labeled data sets, anomaly detection techniques can operate in three different forms; supervised, semi-supervised or unsupervised anomaly detection. [4]

Supervised anomaly detection requires a labeled data set with both normal and anomalous data points. A typical approach would be to build a predictor model for normal vs. anomalous classes. However, often the normal points far outnumber the anomalous points, and obtaining an accurate and representative labeling of the classes can be challenging. To address this, there are

techniques to handle skewed data sets and synthetically create anomalies. [4]

Semi-supervised anomaly detection refers to cases where there are only labeled normal data points. A typical approach is to model the normal behavior and detect instances that do not conform to the expected behavior. [4]

Unsupervised anomaly detection does not require labeled data and is thus widely used. The techniques are based on the assumption that the normal data points are far more frequent than anomalous data points. [4]

In this thesis, both unsupervised and supervised anomaly detection will be explored. For the supervised approach, synthetic anomalies will be injected.

2.2 Machine Learning Approaches to Anomaly Detection

For the scope of this thesis, three machine learning approaches for anomaly detection are chosen. As a baseline, logistic regression is used, which is a simple but effective way to perform a linear combination of attributes to form a prediction of whether the data point is an anomaly or not [5]. As a more sophisticated approach, isolation forest is used. Isolation forest is an ensemble of decision trees, with the idea that anomalies can be extracted from normal data with fewer decisions compared to other normal points [6]. Lastly, an autoencoder is used which enables a deep learning approach based on neural networks that has the ability to learn the complex structures of the data [7].

2.2.1 Feedforward Neural Networks

A neural network is a machine learning model that is loosely inspired by the structure of the human brain. By learning parameters that best approximate some function $f*$, it can be used to perform a large number of tasks [8].

A feedforward neural network is the fundamental model architecture for autoencoders but can also be used to implement logistic regression. Such networks consists of nodes, also called neurons or perceptrons, which are stacked in connected layers. Each connection has a weight determining the

"strength" of the connection, which together with a bias for each neuron can be adjusted during training in order to improve performance. Each neuron has an activation function that determines the computation performed on the input to form the output from the neuron. There are different layers in a feedforward neural network. First, there is an input layer that receives the data, then optional hidden layers that enable complex computations, and finally an output layer that returns the final output. [8]

To update the weights, an optimizer is used. The exact optimizer can vary, but is based on backpropagation which leverages calculus and gradient descent, with the purpose of learning the network to produce the desired output [9]. To compare the predicted output with the desired output, a loss function is used. The loss function can be chosen based on the specific needs of the problem; however, a common one is mean square error. Thus, the task of the optimizer is to minimize the loss function.

To prevent the neural network from overfitting on the training data, that is, not generalizing well, regularization can be introduced. This can be done in several different ways, for example, by adjusting the loss function to penalize too large weights [9], or by randomly dropping out non-output neurons from the network [10]. The probability of a node to drop out during training can be set by a tunable parameter here called *dropout*. The penalty for large weights used in this thesis is L2 regularization, which achieves penalization by adding a term to the loss function that is proportional to the sum of all weights squared in the network. If the original loss function is denoted J_0, the loss function incorporating L2 regularization can be written as

$$J_{L2}(\mathbf{w}) = J_0(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||_2^2 = J_0(\mathbf{w}) + \frac{\lambda}{2}\sum_i w_i^2 \tag{2.1}$$

where $\mathbf{w}$ represents the weights, $||\mathbf{w}||_2^2$ is the squared L2 norm of the weight vector, and λ is a tunable parameter from now on called *weight decay*.

All parameter and configuration values that can affect network performance are grouped under the term hyperparameters. Those parameters are chosen and adjusted during training in order to obtain the best possible model. [8]

2.2.2 Logistic Regression

The logistic model is a statistical model that models the log-odds of an event through a linear combination of one or more independent variables. In terms of anomaly detection, the log-odds for a data point to be an anomaly can be modeled. To estimate the parameters, a logistic regression is performed. The logistic model is based on the logistic function

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

where $f(z)$ ranges from 0 to 1, see fig. 2, and thus can be interpreted as a probability $P(\mathbf{x})$. The logistic model is then formed by writing z as a linear combination of independent variables, x_i, generating the function.

$$f(z) = \frac{1}{1 + e^{-(b+\Sigma w_i x_i)}} \tag{2.3}$$

where b and w_i are unknown parameters that can be modeled [5].

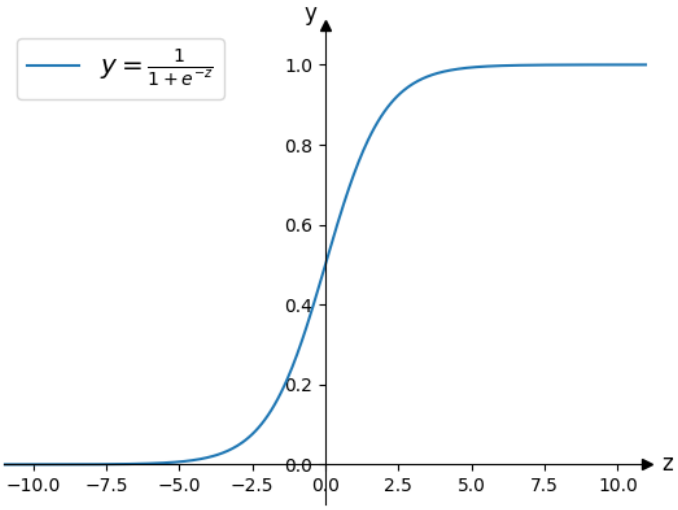


Figure 2: Logistic function

2.2.3 Isolation Forest

Isolation forest is an anomaly detection algorithm based on binary trees. A binary tree is a tree structure in which each node has at most two children; see fig. 3. This method explicitly isolates anomalies, instead of profiling normal points, as most other anomaly detection algorithms. The isolation forest method is based on the "few and different" characteristic of anomalies, which makes them more susceptible to isolation than normal points. Because of this, anomalies are isolated closer to the root of the tree, whereas normal points are isolated deeper down in the tree [6].

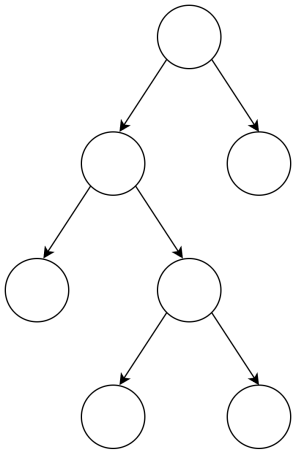


Figure 3: Binary tree

In order to detect anomalies, an ensemble of isolation trees is built, and those points with short average path lengths are anomalies. An isolation tree is made up of nodes, where each node T is either an external node without a child, or an internal node with one test and exactly two children nodes T_l and T_r. The test consists of an attribute q and a split value p such that the test $q < p$ divides the data points into T_l and T_r, see fig. 4. Given a dataset, an isolation tree is built by recursively dividing the data set by randomly selecting an attribute q and a split value p. The tree is finalized when either all data points in each node have the same value or the tree has reached a height limit. By counting the number of edges a data point x traverses from the root node until the traversal is terminated at an external node, a path length $h(x)$ is determined.

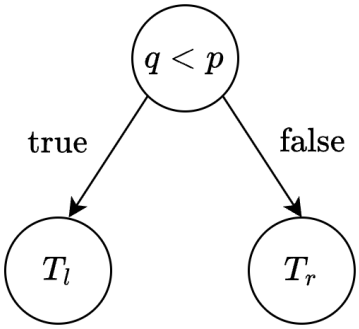


Figure 4: Node structure for isolation trees

Using the path length $h(x)$, an anomaly score can be calculated. Given a data set of n instances, an anomaly score s of an instance x is defined as:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{2.4}$$

where $E(h(x))$ is the average of $h(x)$ from a collection of isolation trees and $c(n)$ is the average path length of unsuccessful search in a binary search tree, given as:

$$c(n) = 2H(n-1) - 2(n-1)/n \tag{2.5}$$

where $H(i)$ is the harmonic number and can be estimated by $\ln(i) + 0.5772156649$ (Euler's constant). Using the anomaly score s, the following conclusions can be drawn.

- When s is close to one, the instance is probably an anomaly.
- When s is much smaller than 0.5, the instance is probably a normal instance.
- If all instances return $s \approx 0.5$, then the entire data set does not really have any distinct anomaly.

[6]

2.2.4 Autoencoders

There are different types of neural networks specifically designed for different tasks. One is called autoencoder which is a network trained to reconstruct its input, first introduced in [11]. The structure is a neural network with an equally sized input and output, with a characteristic bottleneck layer in the middle; see fig. 5. Such networks can be used for a number of things, one of which is anomaly detection. Formally, the problem for autoencoders is to learn two functions $g : \mathbb{R}^n \to \mathbb{R}^p$ and $f : \mathbb{R}^p \to \mathbb{R}^n$ that minimize

$$L(\mathbf{x}, g(f(\mathbf{x}))) \tag{2.6}$$

where L is a loss function that penalizes $g(f(\mathbf{x}))$ for being dissimilar to $\mathbf{x}$, e.g., the mean square error [8]. There are some variants of autoencoders, and for the purpose of this paper, denoising autoencoders will be evaluated further.

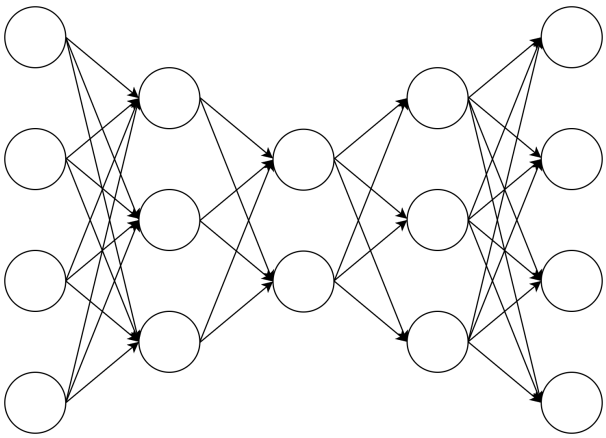


Figure 5: Autoencoder

Denoising Autoencoders

In order to introduce regularization of a normal autoencoder, to prevent it from being just an identity operator, a bottleneck layer is usually created in the middle of the autoencoder. However, the network can still be overfitted and further regularization can be introduced by adding noise to the input data. After doing so, denoising autoencoders can be trained to reconstruct

the clean version of the input. This process can thus be seen as both a regularization option, or as an autoencoder that can be used for error correction. A denoising autoencoder is structurally similar to regular autoencoders, but the training and inference are different. [7]

The process of training a denoising autoencoder consists of first corrupting the input $\mathbf{x}$ with some random noise and then passing the corrupted input $\tilde{\mathbf{x}}$ through the autoencoder. Instead of minimizing (2.6), a denoising autoencoder is trained to minimize

$$L(\mathbf{x}, g(f(\tilde{\mathbf{x}}))) \tag{2.7}$$

where $\mathbf{x}$ is the clean data and $\tilde{\mathbf{x}}$ its corrupted equivalent. The denoising forces the autoencoder to learn the structure of the data, which can then be used for different purposes, such as anomaly detection. [8]

2.3 Related Work

In the paper *Explaining Anomalies using Denoising Autoencoders for Financial Tabular Data* by T. Sattarov et al. [12], anomaly detection for financial tabular data was explored. The purpose of the study was to investigate a framework for explaining anomalies using denoising autoencoders, which was proven to be successful.

The nature of the data used was tabular with numerical and categorical attributes, which required the loss function to be able to evaluate both at the same time. After one hot encoding of the categorical features, the proposed loss function is

$$\mathcal{L}_{\theta,\psi}(x_n^d; \hat{x}_n^d) = \sum_{d=1}^{D_{cat}} \mathcal{L}_{\theta,\psi}^{\text{NLL}}(x_n^d; \hat{x}_n^d) + \sum_{d=1}^{D_{num}} \mathcal{L}_{\theta,\psi}^{\text{MSE}}(x_n^d; \hat{x}_n^d) \tag{2.8}$$

where $\hat{x}_n^d$ denotes the $n-th$ reconstructed sample and its attribute d. Such loss functions capture both categorical and numerical attributes. For numerical attributes, the standard mean square error [8]

$$\mathcal{L}_{\theta,\psi}^{\text{MSE}}(x_n^d; \hat{x}_n^d) = \frac{(x_n^d - \hat{x}_n^d)^2}{D_{num}} \tag{2.9}$$

is used where D_{num} is the number of numerical attributes. For categorical the negative log likelihood [13]

$$\mathcal{L}_{\theta,\psi}^{\text{NLL}}(x_n^d; \hat{x}_n^d) = -\sum_{c=1}^{C_d} x_{n,c}^d \log(\hat{x}_{n,c}^d) \tag{2.10}$$

is used, where C_d is the number of categories for the d-th categorical attribute, $x_{n,c}^d$ is the binary indicator (0 or 1) if the n-th sample's d-th attribute belongs to category c, and $\hat{x}_{n,c}^d$ is the predicted probability that the n-th reconstructed sample's d-th attribute belongs to category c.

However, in order to balance the network between recreating correct data and denoising incorrect data, further enhancements to the loss function were needed. A parameter $\alpha = [0, 1]$ was introduced that is sampled from a *Beta(0.5, 0.5)* distribution. The final proposed loss

$$\mathcal{L}_{\theta,\psi}(x_n^d; \hat{x}_n^d) = \alpha \mathbf{m} \odot \mathcal{L}_{\theta,\psi}(x_n^d; \hat{x}_n^d) + (1-\alpha)\bar{\mathbf{m}} \odot \mathcal{L}_{\theta,\psi}(x_n^d; \hat{x}_n^d) \tag{2.11}$$

where $\mathbf{m}$ is binary mask vector $\mathbf{m} \in \{0, 1\}^d$ with 1 at the entry with noise and 0 otherwise, with complement $\bar{\mathbf{m}}$ and $\odot$ is the element-wise multiplication, effectively handles both categorical and numerical attributes and maintains a balance between the two objectives of the autoencoder.

To explain the cause of an anomaly, the properties of the reconstruction error for each separate attribute are used. For categorical cells, an anomaly score $\pi_n^{d_{cat}}$ is computed as

$$\pi_n^{d_{cat}} = 1 - a_n^{dc} \tag{2.12}$$

where $a(\cdot)$ is the softmax function, which for a vector of logits $z = (z_1, z_2, ..., z_K)$ produces a probability distribution $a(z) = (\sigma(z)_1, \sigma(z)_2, ..., \sigma(z)_K)$ where each probability is given by:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.13}$$

In the context of the anomaly score, $a_n^{dc} = \text{softmax}(\hat{x}_n^d)^c$ refers to the c-th component of the softmax output applied to the reconstructed categorical attribute $\hat{x}_n^d$. This component represents the predicted probability of the d-th categorical attribute of the n-th sample belonging to category c.

For numerical attributes, the cell anomaly score $\pi_n^{d_{num}}$ is computed as

$$\pi_n^{d_{num}} = 1 - e^{-(x_n^d - \hat{x}_n^d)^2} \tag{2.14}$$

Subsequently, a row anomaly score is computed as the sum of all cell anomaly scores π_n^d

$$\pi_n = \sum_{d=1}^{D} \pi_d^d \tag{2.15}$$

Using these anomaly scores, both row anomalies can be found (high row anomaly score), but also which cells contain anomalies (high cell anomaly score). In addition, expected values for numerical values are obtained by the reconstructed value $\hat{x}_n^d$. For categorical, the highest probability category is $\arg\max_c a_n^{dc}$ of the softmax transformation a_n^d.

Chapter 3

Method

3.1 Data Collection

As a central player in the financial industry, OSTTRA has a large unique quantity of data at its disposal. The data used for this project are sensitive and therefore stored in an SQL database only available on a secure server. In the database, each row represents one trade reported by an OSTTRA triResolve client.

The first part of the data collection was to decide which fields were suitable for describing the trades effectively. Each row consists of a large number of entries, where not all are necessary to describe the trade for the scope of this thesis. Some fields are for internal use only, and some are largely populated by missing values. The final selected fields can be seen in table 1

Table 1: Fields used to describe the trades. The first leg refers to the transaction agreed to by the submitting party and the second leg refers to the transaction of the counterparty.

Attribute	Value
Agreement type	The type of agreement, e.g. 'ISDA'
Asset class	The type of trade, e.g. FX
Product class	Detail breakdown of asset class, e.g. 'FX-forward'
Regulated by	The jurisdiction the trade falls under, e.g. 'ESMA'
Curr1	Notional currency for the first leg, e.g. 'USD'
Curr2	Notional currency for the second leg, e.g. 'EUR'
Notional1	The nominal value of the first leg, e.g 10000
Notional2	The Nominal value of the second leg, e.g 15000
MTM	Mark-to-market valuation of the trade, e.g. 20000
MTM Curr	Currency used in the MTM valuation, e.g. 'JPY'
Start date	The start date of the trade, e.g 2010-10-10
End Date	The maturity date of the trade, e.g. 2012-12-12
Trade Date	Date when the trade was created, e.g. 2010-09-20
MTM Date	Date of the MTM valuation, e.g. 2012-08-11

Below, a brief explanation of some aspects of the trade attributes is provided.

- The agreement is a document that specifies certain terms of the transaction; a common one is the "ISDA master agreement", which is published by the International Swaps and Derivatives Association (ISDA) [14].
- The asset class refers to the underlying asset the trade is based on, an example is can be currencies which is referred to as *foreign exchange* (FX).
- The product class refers to the specific type of OTC derivative. In section 1.1.1, options, swaps, and forwards were briefly introduced.
- The regulated by field refers to which jurisdiction the trade falls under. "ESMA" is the European Securities and Markets Authority [15].
- The MTM is the current value of the trade based on current market data.

With the decided fields in place, the match types of the trades were investigated. As triResolve consists of a match-engine that matches one party's trades to its counterpart, helpful labels are put on the trades. For the first purpose of this thesis, to detect anomalies, only trades matched without diffs are used. Other trades can be matched with diffs, or completely unmatched. A trade matched with diff refers to a trade that the match-engine matches to another trade, but some trade attributes differ between the matched trades. The reason for only using trades matched without diffs is to obtain a pool of trades which are probably correct, i.e. a pool of trades without anomalies. After obtaining the matched trades, only one of two matched trades is kept to avoid identical data points. Trades with product classes with a frequency below 1% in the data were not considered.

To collect a reasonable number of trades to conduct the study, 100,000 unique trades with MTM date between 2024-01-01 and 2024-12-31 that met the criteria were randomly selected. The data were split into a training, validation and test set with 70% training, 15% validation, and 15% testing.

For the second purpose of this thesis, to investigate whether anomaly detection can be used to enhance the portfolio reconciliation process, trades matched with diffs are also included in the data.

3.2 Data Corruption

As the data obtained from the database for the first purpose consist solely of correct trades, anomalies had to be created synthetically. In order to keep the synthetically created anomalies as accurate as possible, the domain experts at OSTTRA contributed with their knowledge in the process. The exact way certain fields tend to be anomalous is hard to determine; thus, it was necessary to corrupt these cells in a way which seemed most reasonable. After discussions with the OSTTRA team, it was decided that the cells presented in table 2 were reasonable to corrupt. The exact way these cells are corrupted is described in section 4.2.1

Table 2: Fields reasonable to corrupt in order to create synthetic anomalies

Attributes to Corrupt
Product Class
Curr1/Curr2
Notional1
MTM Curr
MTM
End Date

3.3 Evaluation Framework

The main goal of this thesis is to detect anomalies in OTC derivatives. To obtain an accurate result, training, validation, and tests are performed on different datasets. During training, one epoch of training is done on the training set followed by a validation on the validation set. Iteratively, this is done for a predefined number of epochs, and the model with the lowest loss on the validation data is kept. Finally, tests are performed on the test set. For logistic regression, the tests are based on the log-odds described in section 2.2.2. For the isolation forest, the tests are based on an anomaly score obtained from the implementation used, see section 4.1.2, and finally for the denoising autoencoder, the row anomaly score described in section 2.3 is used.

3.3.1 Anomaly Detection

To evaluate the results for anomaly detection, a number of metrics are used. These are precision, recall, F1-score, ROC and AUC as well as precision-recall curve. The threshold is chosen to maximize the F1-score. The following are the formulas for the different metrics used.

- precision $= \frac{\text{TP}}{\text{FP+TP}}$
- recall $= \frac{\text{TP}}{\text{TP+FN}}$
- F1 $= 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision+recall}}$

Where TP refers to *true positives*, FP refers to *false positives* and FN refers to *false negatives*.

ROC, or *receiver operating characteristics*, is a visual representation of the performance of a model for different thresholds. It is done by calculating the false positive rate, FPR, and the true positive rate, TPR, for different thresholds, then plotting TPR over FPR. The AUC, or *area under curve*, represents the probability that the model, if given a randomly chosen positive and negative example, will rank the positive higher than the negative.

The precision-recall curve is better suited for highly imbalanced data. It is done by plotting precision over recall for different thresholds; thus, it shows the trade-off between precision and recall.

3.3.2 Portfolio Reconciliation

For the second purpose of this thesis, to explore how anomaly detection can be used to enhance portfolio reconciliation, the best performing model on anomaly detection was used. It was explored whether it was possible for that model to detect certain discrepancies in the trades matched with diffs.

Chapter 4

Development and Implementation

4.1 Model Architectures

In this section, the architectures of the three different machine learning models employed for anomaly detection are presented. Model architecture refers to the design of the computational structure that learns patterns from the data. For neural networks, this encompasses the number of layers and the specific operations performed within each layer. For an isolation forest, architecture refers to the way in which the specific forest structure is composed.

4.1.1 Logistic Regression

Logistic regression can be implemented as a feedforward neural network with one input layer and a single output node; see fig. 6. The activation function for the input layer is linear and for the output node it is the logistic function, which together with the weights and bias generates the desired function (2.3) introduced in section 2.2.2.

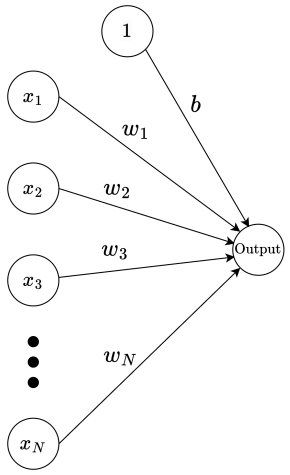


Figure 6: Logistic regression feedforward neural network. x_i is the input, w_i the weights and b the bias.

4.1.2 Isolation Forest

For the isolation forest, the `IsolationForest` class by Scikit-learn is used [16]. The class enables a number of hyperparameters to be set, including the number of trees to build the isolation forest, called *n_estimators*, and the number of samples to draw from the data to train the trees, called *max_samples*. The class can also be used to obtain both predicted classes and anomaly scores for each sample.

4.1.3 Denoising Autoencoder

The denoising autoencoder is implemented as a deep feedforward neural network; see fig. 7. As input layer, N neurons are used, where N is the number of features in the data after the preprocessing described in section 4.2 is done. The features are derived from the trade attributes; more on this later. Then, three hidden layers with 256, 128 and 256 neurons, respectively, are used. The chosen architecture is inspired by [12]. Finally, the output layer consists of N neurons. As activation functions, the hidden layers have ReLU,

$$\text{ReLU} = \max(0, x) \tag{4.1}$$

where x is the output of each node. The input and output layer has an

ordinary linear activation function (i.e., no explicit activation function is applied to its output).

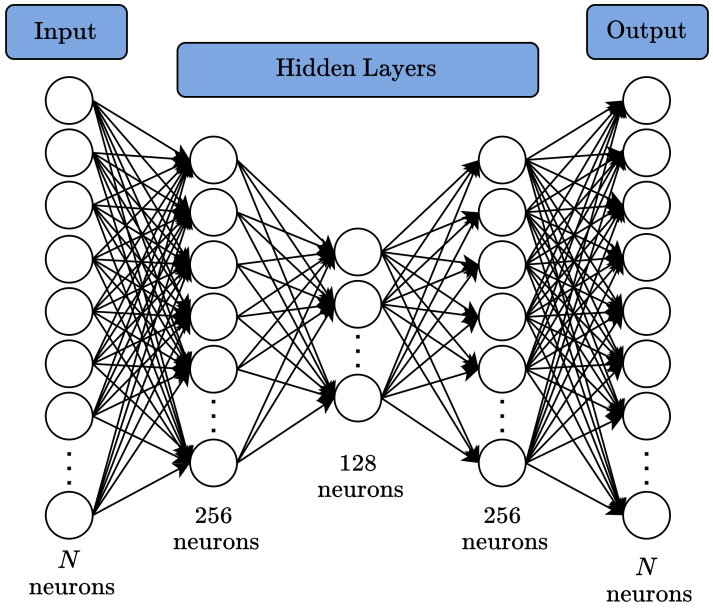


Figure 7: Architecture of the denoising autoencoder. N is the number of features after preprocessing.

4.2 Preprocessing

Before data could be passed on to models for training and evaluation, preprocessing was necessary. First, missing values were fixed. Empty categorical values were filled with a new category "Null", missing numerical values were filled with 0 and missing date values were filled with 1970-01-01.

The MTM currency was then replaced with the USD exchange rate on the specific MTM date. That is, the categorical value was replaced by a numerical value.

The dates were then converted to the number of days since the unix epoch, 1970-01-01.

Then, standard scalers were fitted to the numerical values, before the data could be corrupted to create anomalies. After the data was corrupted, the standard scalers were applied to the numerical values, and the categorical values were one-hot encoded, a process that transforms each categorical feature into a set of binary (0 or 1) columns. Specifically, for each unique category within a feature, a new binary column is created. If a particular data point belongs to that category, the corresponding binary column will have a value of 1, while all other binary columns for that feature will have a value of 0. This representation allows machine learning models to effectively process categorical data, as it avoids imposing any ordinal relationship between the categories and converts them into a numerical format that algorithms can readily understand and utilize for learning.

After preprocessing, the dimension of the feature vector was 210.

4.2.1 Data Corruption

In order to create labeled anomalies, such anomalies had to be created synthetically. After consultation with the OSTTRA team, described in section 3.2, the following was done.

First, a random 10% of the trades is selected to be corrupted. For each of these trades, a random selection of 3 to 6 attributes in table 2 is corrupted. The attribute corruption is then done as:

- *Product Class*: Change to another random product class within the same asset class
- *Curr1/Curr2*: Switch Curr1 and Curr2
- *Notional1*: Multiply by a random factor μ, described below
- *MTM Curr*: Switch to another random currency, which also occurs as MTM Curr
- *MTM*: Multiply by a random factor μ, described below
- *End Date*: Shift a random number of days backward or forward, uniformly chosen from [7,31]

Let the random factor be denoted by μ. Define μ as:

$$\mu = e^{\delta}$$

where δ is a random variable drawn from a modified uniform distribution. Let $U(a, b)$ denote a continuous uniform distribution between a and b (inclusive). The distribution of δ can be defined piecewise as:

$$\delta \sim \begin{cases} U(-4.6, -2.3) & \text{with probability } \frac{1}{2} \\ U(2.3, 4.6) & \text{with probability } \frac{1}{2} \end{cases}$$

This was done to create a distribution of corruptions for MTM and Notional1 that could reflect realistic errors. The corruption process is exemplified in fig. 8.

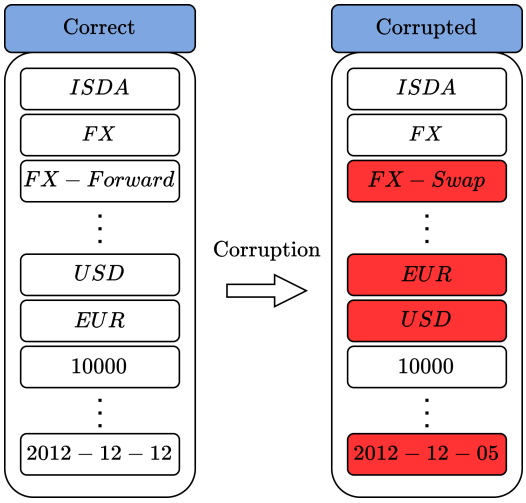


Figure 8: Example corruption, where *Product Class* is changed, *Curr1* and *Curr2* switched and *End Date* is shifted.

After the data are corrupted, the training process can take place.

4.3 Training Procedure

During training, the data set is used to learn the models to predict whether an unseen trade is anomalous or not. As the data are labeled after the corruption process, supervised anomaly detection is possible.

4.3.1 Logistic Regression

For logistic regression, the aim of training is to estimate the parameters in (2.3). This is done by first splitting the data into batches of 256 in size, to enable batch training. The loss function used is *binary cross entropy*, BCE;

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \tag{4.2}$$

where N is the number of observations, y_i is the true label (1 or 0) and p_i is the predicted probability of being class 1. This loss function is suitable for a binary classification task (anomaly or not). As optimizer, stochastic gradient descent is used, which has a learning rate hyperparameter.

The learning rate hyperparameter is tuned using Optuna [17], which is a framework to tune hyperparameters for machine learning pipelines. To find the best values, Optuna utilizes tree-structure optimization algorithms. The process is initialized by defining a search space for the hyperparameter, and then the optimization algorithm finds the best value based on a chosen objective. For the logistic regression learning rate hyperparameter, the search space is chosen to LogUniform[1e-5, 1e-1], with the objective of minimizing the BCE loss on the validation data.

After one epoch is done, the performance on the validation data set is tested and after 100 epochs, the model with the best performance on the validation data is kept to be evaluated on the test set.

4.3.2 Isolation Forest

The isolation forest training process involves fitting the isolation forest object to the training data using a built-in method in the `IsolationForest` class. During training, the algorithm constructs an ensemble of isolation trees. For

each tree, a subset of the training data (determined by the max_samples parameter) is randomly selected. Each tree is then built by randomly selecting a feature and a split value within the range of that feature for the current subset of data, recursively partitioning the data until isolation is achieved or a stopping criterion is met. After training, the model can be used to predict whether a data point is an anomaly or not.

The number of trees and the number of samples to train each tree is determined using Optuna. The search spaces for these hyperparameters are presented in table 3, and the objective is to maximize the F1-score on the validation set.

Table 3: Hyperparameter search space for isolation forest

Hyperparameter	**Search space**
n_estimators	Uniform[100, 1000]
max_samples	Uniform[100, 10000]

4.3.3 Denoising Autoencoder

For the denoising autoencoder, the training consists of both learning the autoencoder to recreate correct trades $\mathbf{x}$, but also "denoising" incorrect trades $\tilde{\mathbf{x}}$. This is done using both correct and anomalous trades as input, but only using correct trades as a comparison for the output $\hat{\mathbf{x}}$. That is, if the input is an anomalous trade, the output is compared with its corresponding correct version. If the input is a correct trade, the output is compared to the trade used as input. The training procedure is summarized in fig. 9.

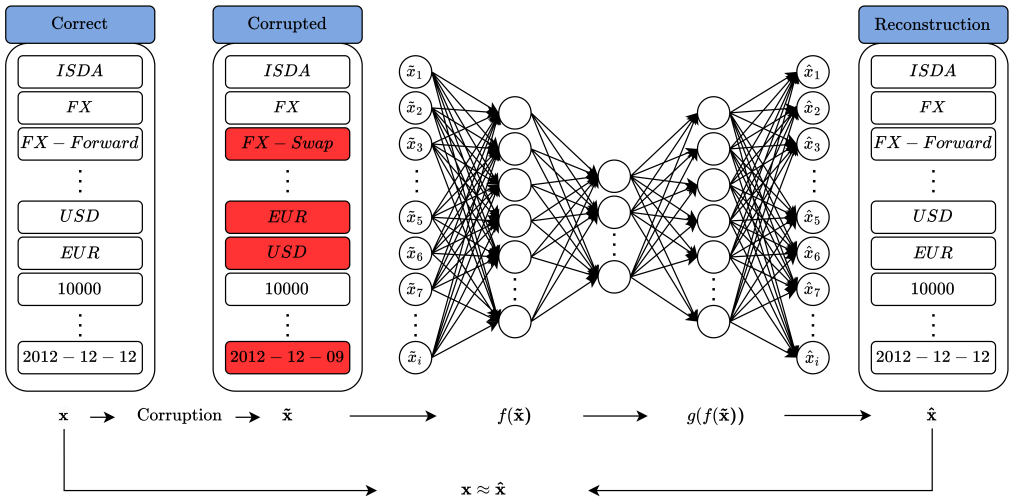


Figure 9: Schematic view of the training procedure for the denoising autoencoder

As loss function, the custom function (2.11), introduced in section 2.3, is used. The optimizer is Adam [18], an adaptive learning rate optimization algorithm that computes individual learning rates for different parameters by maintaining estimates of the first and second moments of the gradients. Complementing Adam is a cosine learning rate scheduler [19], which modulates the learning rate over the course of training following a cosine function. This schedule typically starts with a relatively high learning rate that gradually decreases to a minimum, potentially increasing again in subsequent cycles. The number of cycles hyperparameter dictates how many such full or partial cosine cycles occur during training, influencing the frequency of learning rate adjustments. Furthermore, the T_mult hyperparameter determines how the duration of each cycle evolves; if T_mult is greater than 1, each subsequent cycle becomes longer, allowing for finer-grained optimization in later stages.

The first part of training, after the corruption is done, is to split up the data into batches of size 256. Then the Adam optimizer, together with the learning rate scheduler and loss function, is used to update the weights to minimize loss. Dropout and weight decay is used in order to prevent overfitting on the training dataset. After each epoch, the model is evaluated on the validation data, and after 100 epochs, the model with lowest loss on the

validation data is kept to be tested on the test data.

The hyperparameters for the denoising autoencoder are chosen using Optuna. The different parameters are the dropout rate in the training phase, the initial learning rate, number of cycles and T_mult for the cosine learning rate scheduler, and finally the weight decay factor. The search spaces for these parameters are presented in table 4.

Table 4: Hyperparameter search space for the autoencoder

Hyperparameter	**Search space**
Dropout	Uniform[0.0, 0.5]
Learning rate	LogUniform[1e-5, 1e-2]
Number of cycles	Discrete[1, 2, 3, 4, 5]
T_mult	Discrete[1, 2, 3]
Weight decay	Uniform[0.0, 0.1]

4.4 Hardware and Software

The development and testing was carried out in a secure Linux environment running on a 32 GB RAM CPU. The software was Python 3.11.6, with PyTorch (2.2.1) and Jupyter (1.0.0). Scikit-learn (1.4.1.post1) was used for the isolation forest and the autoencoder and logistic regression were developed using PyTorch. Optuna (3.5.0) was used for the hyperparameter search. NumPy (1.26.4) and Pandas (2.2.1) were used for data handling, and Scikit-learn and Matplotlib (3.8.3) were used to analyze and evaluate the results.

Chapter 5

Results and Analysis

5.1 Anomaly Detection

The results for the first purpose of this thesis, anomaly detection, are presented here for the three different approaches. The optimal hyperparameters obtained from the hyperparameter tuning are also presented.

5.1.1 Logistic Regression

Hyperparameter Tuning

The only hyperparameter that needed to be tuned for logistic regression was the learning rate, with the resulting optimal learning rate presented in table 5.

Table 5: Tuned hyperparameter for logistic regression

Hyperparameter	Optimal value
Learning rate	9.917×10^{-2}

Anomaly Detection

For the baseline model, logistic regression, the ability to detect anomalies was evaluated with the metrics presented in 3.3.1, with the numerical metrics presented in table 6 and the ROC curve in fig. 10 and the precision-recall curve in fig. 11.

Table 6: Results for logistic regression

Metric	Value
Precision	0.3558
Recall	0.5040
F1	0.4171

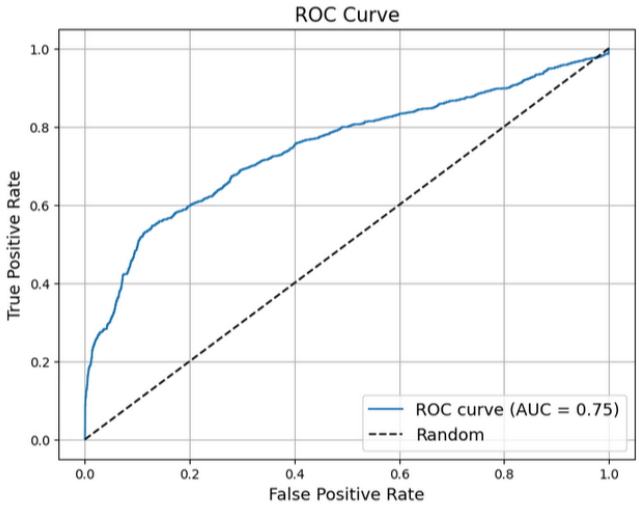


Figure 10: ROC curve for logistic regression, with an AUC of 0.75

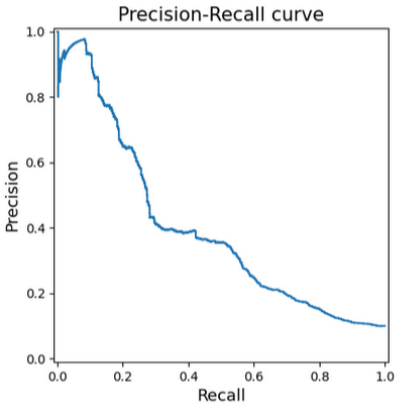


Figure 11: Precision-Recall curve for logistic regression

Analysis

The performance of the logistic regression model reveals a baseline capability in anomaly detection. The precision of 0.3558 indicates that when the model identifies a trade as anomalous, it is correct only about 35.58% of the time. This suggests a high rate of false positives, where normal trades are incorrectly flagged as anomalies. Recall, at 0.5040, shows the model's ability to capture actual anomalies, detecting approximately 50.40% of all anomalous trades. The F1 score, which balances precision and recall, is 0.4171.

The ROC curve (fig. 10) illustrates the performance of the model across different classification thresholds, with an AUC of 0.75. This indicates a moderate ability to distinguish between anomalous and normal trades, better than random chance, but leaves room for improvement. The precision-recall curve (fig. 11) further emphasizes the challenge of achieving high precision and recall simultaneously. As recall increases, precision drops significantly, highlighting the difficulty in accurately identifying anomalies without increasing the number of false positives.

In summary, while logistic regression offers a starting point for anomaly detection, its performance is limited by a high false-positive rate and a moderate ability to distinguish between anomalous and normal trades.

5.1.2 Isolation Forest

Hyperparameter Tuning

The hyperparameters to be tuned for the isolation forest were max_samples and n_estimators, with the optimal values presented in table 7.

Table 7: Tuned hyperparameters for isolation forest

Hyperparameter	Optimal value
max_samples	4440
n_estimators	139

Anomaly Detection

The numerical metrics for the ability of the isolation forest to detect anomalies are presented in table 8. The ROC curve is presented in fig. 12 and the precision-recall curve in fig. 13.

Table 8: Results for isolation forest

Metric	Value
Precision	0.00
Recall	0.00
F1	0.00

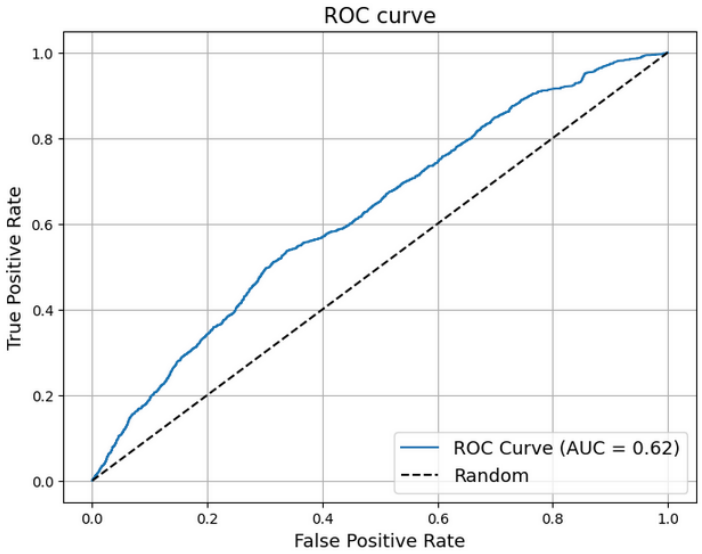


Figure 12: ROC curve for isolation forest, with an AUC of 0.62

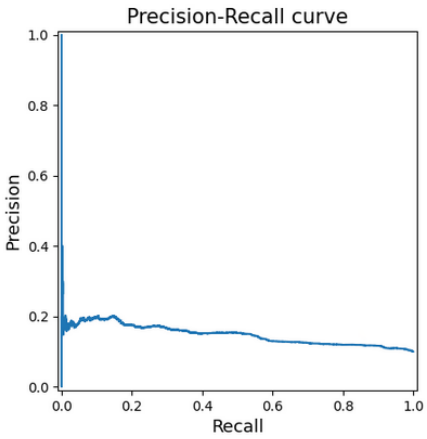


Figure 13: Precision-Recall curve for isolation forest

As the isolation forest computes the anomaly scores for each sample, the distribution of the anomaly scores is plotted in fig. 14.

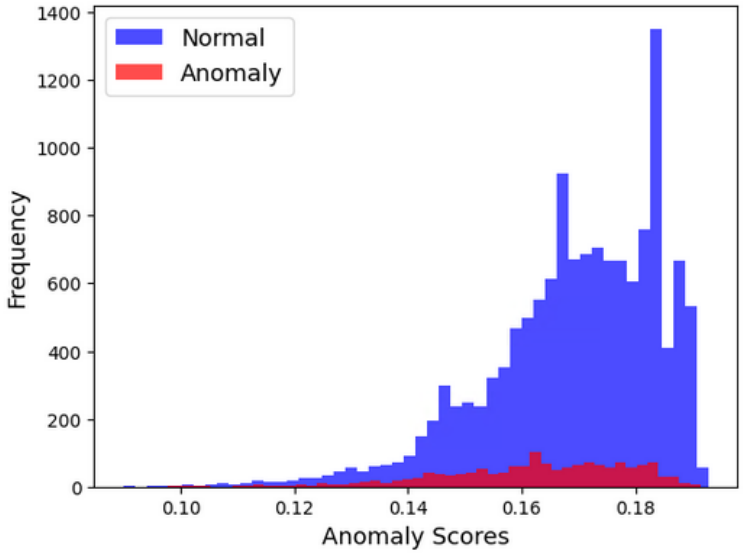


Figure 14: Anomaly scores for isolation forest. The lower score, the more abnormal.

Analysis

The isolation forest model demonstrates very poor performance in anomaly detection within this dataset. The precision, recall and F1 score are all 0.00, indicating that the model did not correctly identify any anomalies. This is further supported by the ROC curve (fig. 12), which shows an AUC of 0.62, suggesting that the model performs slightly better random chance in distinguishing between anomalous and normal trades. The precision-recall curve (fig. 13) also confirms the model's inability to effectively detect anomalies, since no meaningful recall and precision could be obtained simultaneously.

The distribution of anomaly scores (fig. 14) reveals that the scores for normal and anomalous data points are not separated, showing a complete overlap between the two classes. This lack of separation explains the model's failure to achieve meaningful precision and recall.

In summary, the isolation forest model, with its tuned hyperparameters, is not suitable for anomaly detection in this particular data set. The model's inability to distinguish between anomalous and normal trades, as evidenced by the near-zero performance metrics and poor ROC and precision-recall curves, suggests that it consistently labels trades as normal.

5.1.3 Denoising Autoencoder

Hyperparameter Tuning

The hyperparameters to be tuned for the denoising autoencoder were the dropout rate, learning rate, number of cycles, T_mult and weight decay, with the optimal values presented in table 9.

Table 9: Tuned hyperparameters for the denoising autoencoder

Hyperparameter	Optimal value
dropout rate	8.771×10^{-2}
learning rate	1.357×10^{-3}
number of cycles	1
T_mult	1
weight decay	3.283×10^{-4}

Anomaly Detection

The numerical metrics for the denoising autoencoder are presented in table 10. The ROC curve is plotted in fig. 15 and the precision-recall curve in fig. 16.

Table 10: Results for the denoising autoencoder

Metric	Value
Precision	0.6078
Recall	0.6313
F1	0.6194

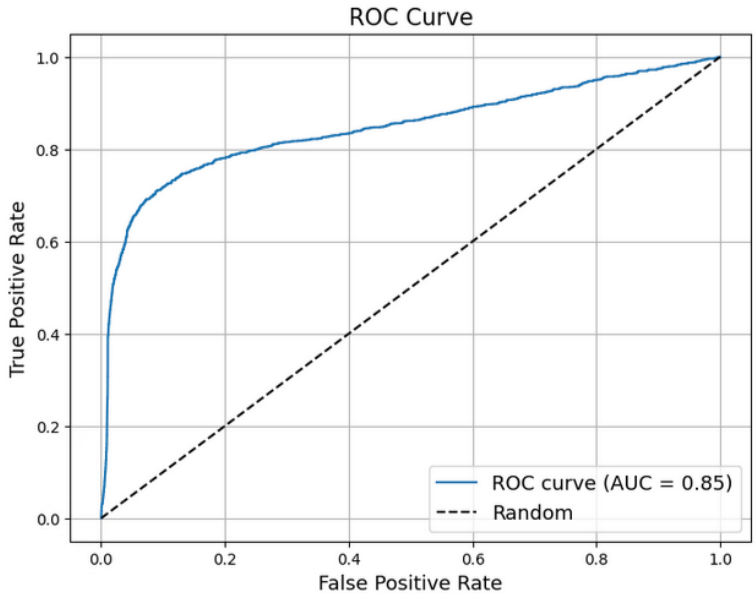


Figure 15: ROC curve for the denoising autoencoder, with an AUC of 0.85

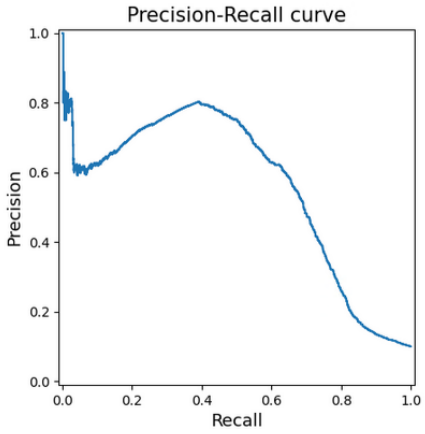


Figure 16: Precision-Recall curve for the denoising autoencoder

The distribution of anomaly scores for the denoising autoencoder is presented in fig. 17. As a random number of 3 to 6 attributes is corrupted for the synthetically created anomalies, the distribution of anomaly scores for trades with 3, 4, 5 and 6 corrupted attributes, respectively, is plotted in fig. 18.

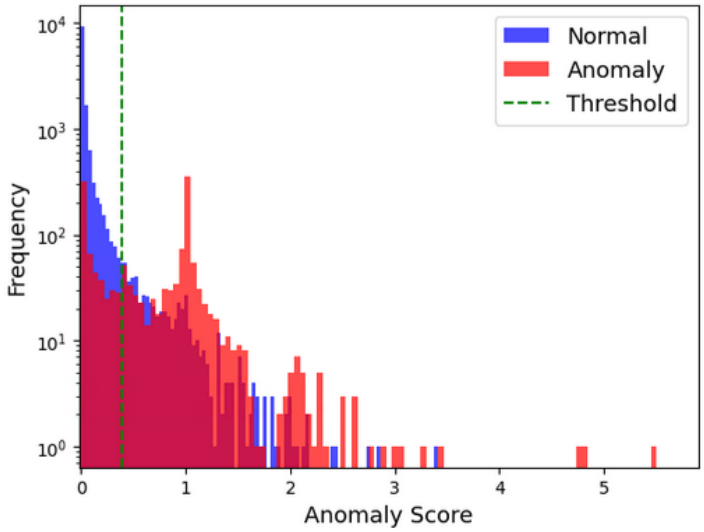
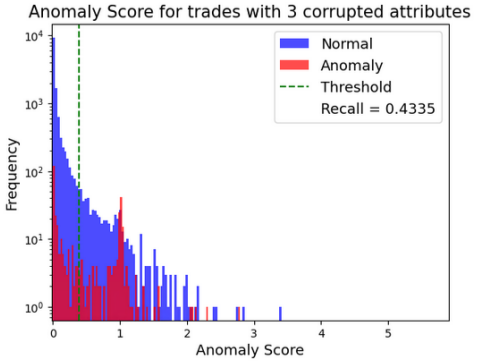
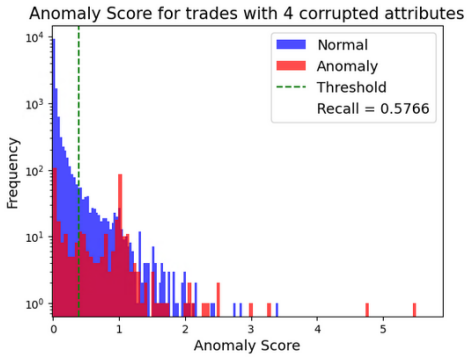


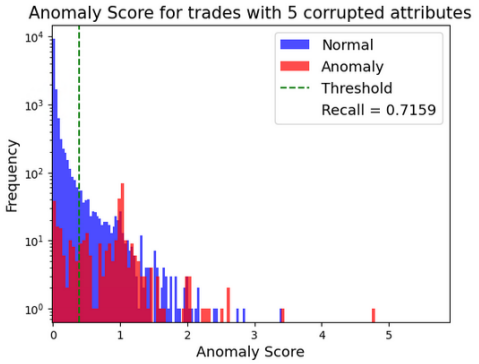
Figure 17: Anomaly scores for the denoising autoencoder, with a log scale on the y-axis. The higher score, the more abnormal.



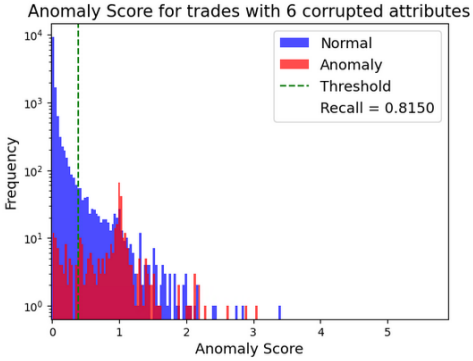
(a) 3 corrupted attributes



(b) 4 corrupted attributes



(c) 5 corrupted attributes



(d) 6 corrupted attributes

Figure 18: Anomaly scores for the denoising autoencoder for trades with 3-6 corrupted attributes. Log scale on the y-axis.

To gain further insights on the ability of the denoising autoencoder to detect anomalies, the different combinations of possible corruptions were investigated. The mean and standard deviation of the anomaly score provide a general view of the performance. In fig. 19, fig. 20 and fig. 21 these values are presented for all the different combinations of corruptions for trades with 3, 4 and 5 corrupted attributes, respectively.

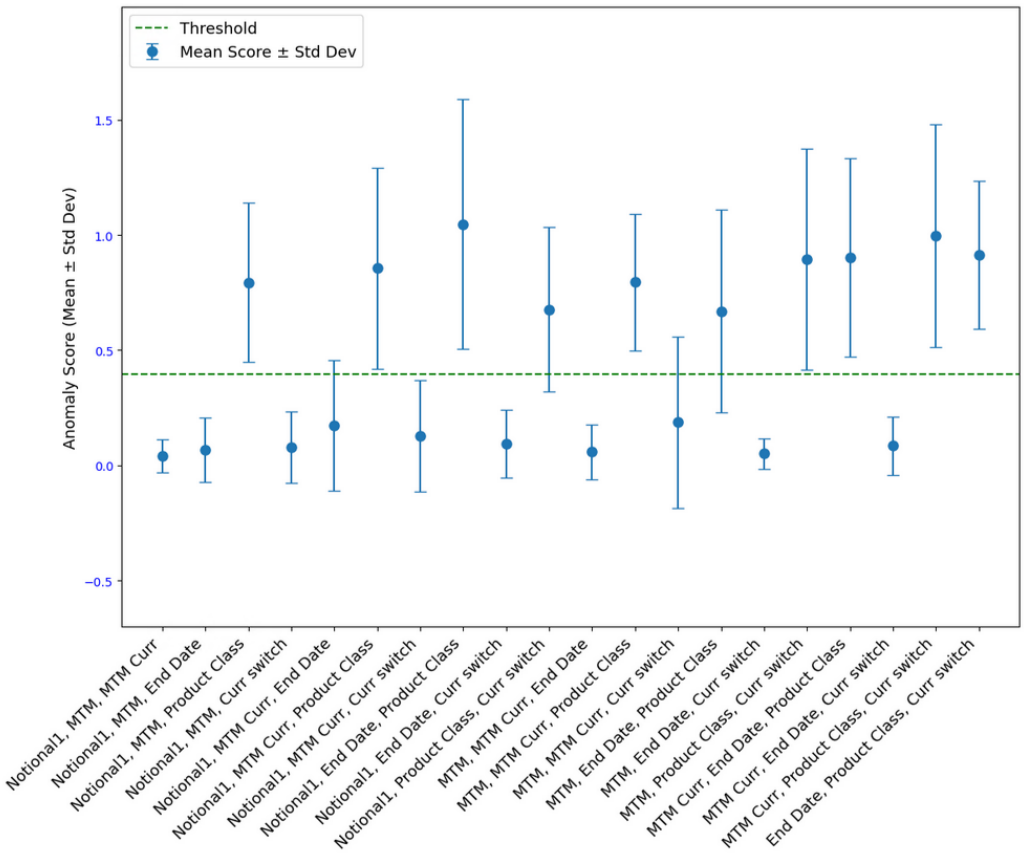


Figure 19: Anomaly score mean with standard deviation, and threshold for denoising autoencoder for all corruption combinations for trades with 3 corrupted attributes.

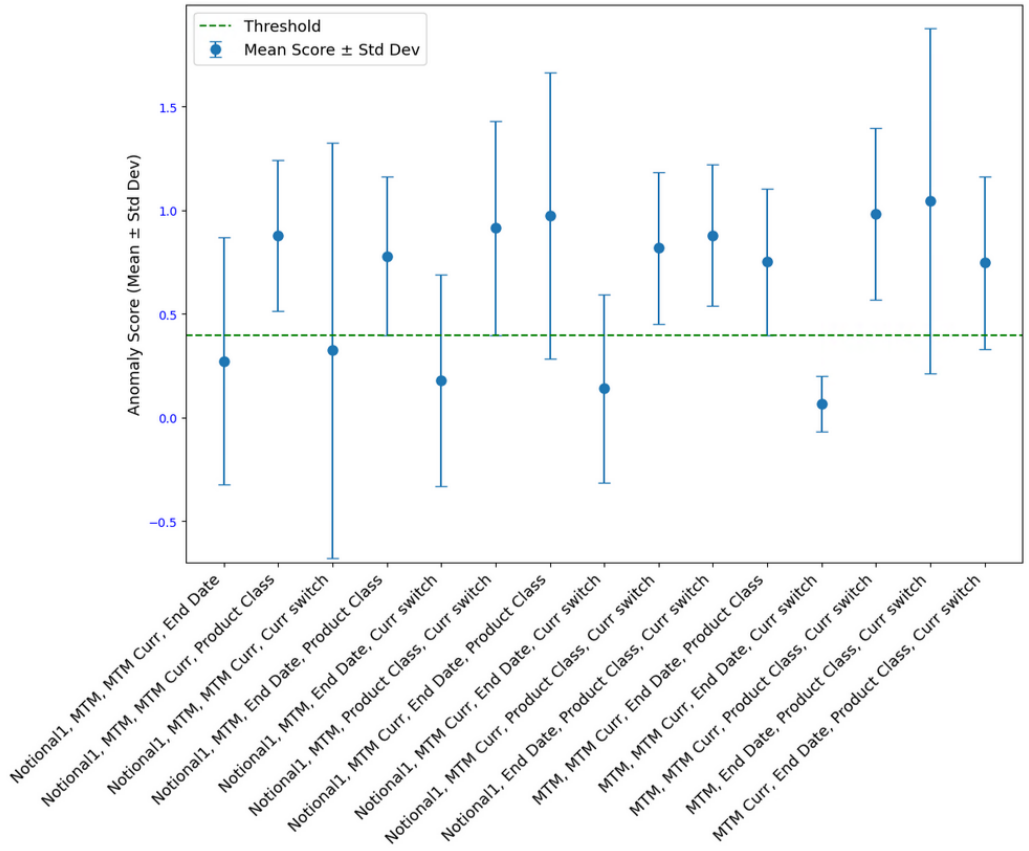


Figure 20: Anomaly score mean with standard deviation, and threshold for denoising autoencoder for all corruption combinations for trades with 4 corrupted attributes.

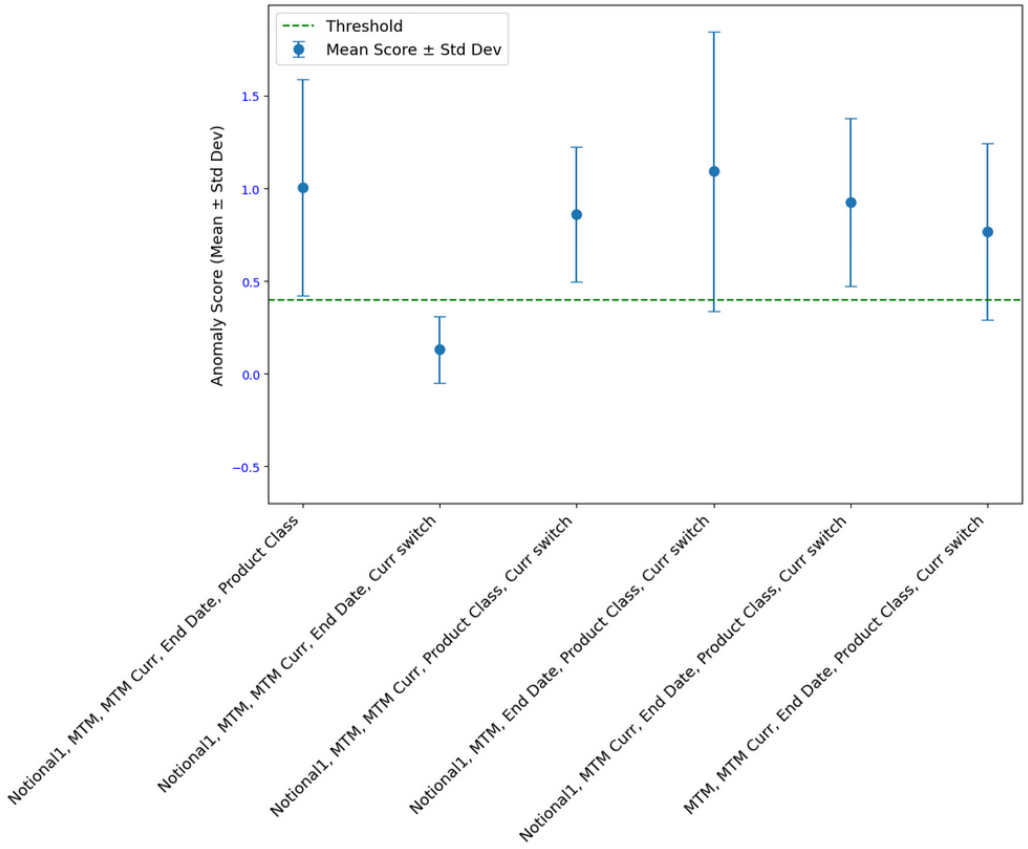


Figure 21: Anomaly score mean with standard deviation, and threshold for denoising autoencoder for all corruption combinations for trades with 5 corrupted attributes.

Finally, the distribution of anomaly scores for trades with the product class *not* corrupted is presented in fig. 22, to be compared with the distribution of anomaly scores for when the product class is corrupted, see fig. 23.

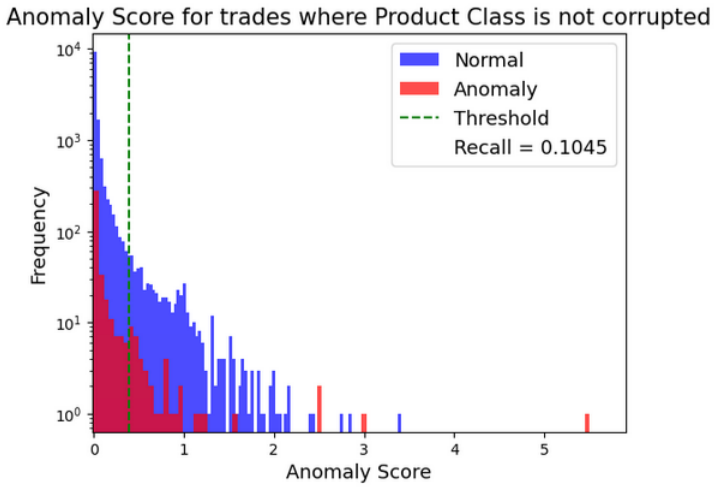


Figure 22: Anomaly scores for denoising autoencoder without product class as a corrupted attribute. Log scale on the y-axis.

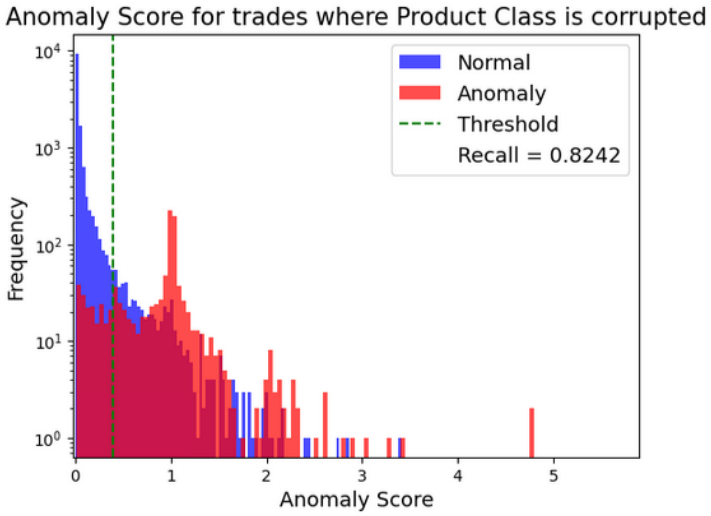


Figure 23: Anomaly scores for denoising autoencoder with product class as a corrupted attribute. Log scale on the y-axis.

Analysis

The denoising autoencoder demonstrates the strongest performance in anomaly detection among the models evaluated. With a precision of 0.6078, the model shows a good ability to correctly identify anomalous trades. A recall of 0.6313 indicates that the model captures a significant portion of the actual anomalies. The balance between recall and precision is reflected in the F1 score of 0.6194.

The ROC curve (fig. 15) shows an AUC of 0.85, indicating a strong ability to distinguish between anomalous and normal trades. This is further supported by the precision-recall curve (fig. 16), which maintains a relatively high precision as the recall increases, demonstrating the model's effectiveness in anomaly detection.

The distribution of anomaly scores (fig. 17) reveals a clearer separation between normal and anomalous trades compared to the isolation forest. As the number of corrupted attributes increases from 3 to 6 (fig. 18), the separation between the anomaly scores of the normal and anomalous trades becomes more distinct, and the recall improves from 0.4335 (fig. 18a) to 0.8150 (fig. 18d). This indicates that the autoencoder is more sensitive to trades with more corrupted attributes.

Further analysis of the anomaly scores for different combinations of corrupted attributes provides insights into the model's performance under various conditions. The plots show how the mean anomaly score and its standard deviation vary between different combinations of corrupted attributes, highlighting which types of corruption are more easily detectable by the model. In fig. 19, fig. 20 and fig. 21 it becomes clear that trades that have a corrupted product class have a significantly higher anomaly score than trades that do not. This is further shown in fig. 22 and fig. 23 where it is even more clear that the model only seems to be able to detect product class anomalies as the recall for corrupted trades without a corrupted product class is 0.1045, while for corrupted trades with a corrupted product class the recall is 0.8242. This behavior of the model will be further discussed in chapter 6.

In summary, the denoising autoencoder outperforms the other models, demonstrating a capability to detect anomalies, particularly those with a higher

number of corrupted attributes. The model distinguishes between normal and anomalous trades, as evidenced by the high AUC and precision-recall performance. However, the only corruption to which the model seems sensitive is product class corruptions.

5.2 Portfolio Reconciliation

For the second purpose of this thesis, to explore whether anomaly detection can be used to enhance the portfolio reconciliation process, the denoising encoder was used as it produced the best results in anomaly detection. The ability of the denoising autoencoder to enhance the portfolio reconciliation process is explored by letting both trades in a match with product class diff pass through the network. The output product class is then compared to the input product class, and specifically, whether it was changed or not. If it is not changed, the network interprets the input product class as normal, and if it is changed, the product class is interpreted as erroneous. Both sides of a match are then investigated, and as the match is with a product class diff, at least one of the sides should have a changed product class, as at least one of the sides is likely wrong about their reported product class. To compare the results for trades matched with diff, both trades in a match without diff are also passed through the network, and changes in product classes are tracked similarly. Several matches were passed through the network, with an almost equal amount of matches without diff and matches with product class diff. The exact number of trades can not be disclosed due to data privacy issues.

5.2.1 Results

The results are evaluated by creating a confusion matrix, see table 11, where a match with diff is labeled as anomaly, and a match without diff is labeled as normal. If at least one of the trades in a match changes the product class, then it is interpreted as the network predicts the match as an anomaly. The trades passed through the network are obtained from the same time period as the prior data, but the trades included in the prior training, validation, and test data are not considered.

Table 11: Confusion matrix where an actual anomaly is a match with diff and actual normal is a match without diff. A predicted normal is a match where none of the trades product class is changed, and a predicted anomaly is a match where at least one of the trades product class changes.

	Predicted anomaly	**Predicted normal**
Actual anomaly	36%	13%
Actual normal	3%	48%

Further evaluation is done by creating a 2×2 matrix; see table 12, where the two trades in a match are represented by counting the number of times the product class is changed or not. The results for matches with diffs and matches without diff are then presented separately in table 13 for matches with diffs and table 14 for matches without diff.

Table 12: Table representing whether the product class is changed or not for both parts in a match

	Trade 2	
Trade 1	**Not changed**	**Changed**
Not changed	No trade changed	Only trade 2 changed
Changed	Only trade 1 changed	Both trades changed

Table 13: Table representing whether the product class is changed or not for both parts in a match with product class diff

	Trade 2	
Trade 1	**Not changed**	**Changed**
Not changed	26%	8%
Changed	16%	50%

Table 14: Table representing whether the product class is changed or not for both parts in a match without diff

	Trade 2	
Trade 1	**Not changed**	**Changed**
Not changed	93%	3%
Changed	2%	2%

5.2.2 Analysis

The results for the possibility of portfolio reconciliation enhancement are a bit more unreliable. The main reason for this is the fact that the matches are now evaluated, not individual trades, which will be discussed further in chapter 6. That being said, table 11 suggests that the denoising network performs well especially in not changing product classes for trades that are in a match without diff. Only 3% of the trades in the total number of trades are false positives (actual normal predicted as anomaly). For the actual anomalies, the performance is also fairly good, but with a non-negligible occurrence of false negatives (actual anomalies predicted as normal).

Looking at the changes in product class for the separate trades in the matches, further insights can be obtained. In table 13, where the matches with diff are evaluated, it can be seen that when the product class is changed, it is often for both trades in the match. This suggests that in a match with product class diff, the autoencoder is prone to identify both product classes as incorrect. For matches without diff, 93% of the matches had both trades without changed product class, this further suggests that the denoising autoencoder correctly identifies most of the matches without diff as normal.

Chapter 6

Discussion

6.1 Synthetic Anomaly Creation

A fundamental challenge in this thesis lies in the synthetic creation of anomalies. As noted in section 3.2, the absence of a fully labeled dataset of real-world anomalies required the generation of artificial anomalies. Although this approach allowed for a supervised learning framework, it inherently introduces a degree of insecurity. The corruption process, guided by domain expertise, aimed to mimic potential errors in OTC derivatives. However, it is crucial to acknowledge that these synthetic anomalies are, at best, an approximation of reality. The complexity of real-world errors, which may arise from unforeseen reasons, cannot be perfectly replicated.

Despite these limitations, the synthetic approach was essential to evaluate anomaly detection models. It provided a controlled environment where the model's ability to identify anomalies could be quantitatively assessed. Furthermore, the collaboration with OSTTRA's domain experts added a layer of realism to the corruption process, ensuring that the generated anomalies were at least plausible within the context of OTC derivatives.

6.2 Isolation Forest Shortcomings

The isolation forest algorithm demonstrated a clear inability to detect anomalies in the data set. The results presented in section 5.1.2, indicate a performance close to random chance. This outcome can probably be attributed to

the nature of the synthetic anomalies and the characteristics of the isolation forest algorithm.

The isolation forest relies on the identification of anomalies that are distinct and isolated within the data space. In this study, anomalies were created by corrupting a combination of attributes within a trade. Although the combination of these corruptions made the trade anomalous, each individual attribute value might still fall within a normal range. For instance, changing the product class from "FX-Forward" to "FX-Swap" is an error, but both "FX-Forward" and "FX-Swap" are valid entries for the product class attribute.

The isolation forest, by design, isolates data points based on the number of partitions required in a random tree structure. If each attribute value appears normal, the isolation forest algorithm struggles to identify the trade as an anomaly. The algorithm does not capture the complex relationships and dependencies between attributes that define an anomalous trade in this context.

6.3 Denoising Autoencoder and Product Class Anomalies

The denoising autoencoder demonstrated superior performance compared to logistic regression and isolation forest. However, a critical observation is its apparent focus on detecting product class anomalies. As shown in section 5.1.3, the model's ability to detect anomalies is significantly higher when the product class attribute is corrupted.

Several factors might contribute to this behavior. One possibility is the categorical nature of the product class attribute. If the training data distribution among different product classes is very skewed, the denoising autoencoder could possibly inherently interpret data points with nonconforming product class as erroneous. As the corruption process includes changing the product class, there is a possibility that a vast majority of these changes consists of going from a highly frequently occurring product class, to a much less frequently occurring product class. This new, less frequent, product class

could then almost always be interpreted as erroneous, correctly so for corrupted data points. However, also incorrectly interpreted as erroneous for non-corrupted data points with a less frequent product class.

Another possible explanation lies in the nature of the corruption. Product class corruption involves changing the value to another valid product class within the same asset class. This type of corruption might lead to a more substantial change in the data representation compared to, for example, a numerical change in the MTM value. The autoencoder could then find the possible more substantial change, which consequently produces a higher anomaly score.

It is also important to consider the potential interaction between the product class and other attributes. When representing a trade, the product class could possibly be the most important attribute among the attributes that could be corrupted. This would lead to changes in the product class easier to find, and thus give such instances higher anomaly scores.

6.4 Portfolio Reconciliation Application Possibilities and Challenges

The application of the denoising autoencoder to the portfolio reconciliation process yielded mixed results. Although the model demonstrated a strong ability to correctly classify matches without discrepancies, the results for matches with product class differences were less definitive.

A key challenge in this analysis is the change from individual trades to trade matches. In the anomaly detection task, the model assessed the likelihood that a single trade is anomalous. In the portfolio reconciliation context, the model output is used to classify the relationship between two trades. This introduces a layer of complexity.

Specifically, it cannot be definitively concluded how product class differences manifest within the data. The data indicates that a difference exists, but not which trade (or both) contains the error. The tendency of the denoising autoencoder to flag both trades in a match with a product class difference as

potentially erroneous, as shown section 5.2.1, raises questions. It is possible that the autoencoder is indeed identifying the inconsistency, but it might also be oversensitive, flagging both trades even if only one is incorrect.

Furthermore, the evaluation metric used in the portfolio reconciliation analysis differs from the anomaly detection metrics. In the anomaly detection evaluation, a certain threshold is used to distinguish between anomalies and non-anomalies. In the context of the portfolio reconciliation process, only a change in the product class is used. As the autoencoder outputs probabilities for different product classes, the outputted product class with the highest probability is interpreted as the solely outputted product class. Although several product classes could have almost as high probability, only one is used in the evaluation. Thus, some threshold could also be used here to further enhance the analysis and results.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

This thesis explored the use of machine learning techniques for anomaly detection within OTC derivative data, with the particular goal of evaluating the possibilities to enhance the portfolio reconciliation process at OSTTRA. The study compared the performance of logistic regression, isolation forest and denoising autoencoders in identifying synthetically generated anomalies.

Regarding the first research question, if logistic regression, isolation forest, or autoencoders could be used to detect anomalies among OTC derivatives, some conclusions could be drawn. Firstly, isolation forest could not be used for this particular use case, while logistic regression showed some limited performance. On the other hand, a denoising autoencoder had more promising results with clear capabilities to detect certain anomalies among the data. However, the research process comes with some limitations. Basically, the synthetic creation of anomalies limits the possibility of drawing conclusions about how the models can perform on real-world data.

For the other research question, if anomaly detection could be used to enhance the portfolio reconciliation process, the conclusion is that in some limited use cases there may be a possibility. Specifically, the detection of incorrectly reported product classes showed some promising results. With that being said, to fully investigate the possibilities and limitations of this process, further research is needed.

7.2 Further Work

In order to further investigate the research questions in this thesis, some important improvements can be made. As discussed above, the synthetic creation of anomalies implies certain amounts of uncertainty. A real-world labeled dataset that could be used to perform supervised anomaly detection would further improve the quality of the research. Furthermore, the data obtained for this study could possibly be skewed in certain ways, potentially affecting the results. To account for this, more insights in the data is needed. Another aspect of data collection is the time span from which data are collected. In this study, the data are from 2024, which implies that the autoencoder is trained solely on trades from that period of time and thus is only applicable to find anomalies from the same period of time. To further enhance the anomaly detection process by being able to find anomalies in current data, additional adjustments to current market conditions need to be built into the model. Regarding the portfolio reconciliation process, improvements can be made by specifically training a model to detect certain anomalies, instead of a general approach as in this thesis. Furthermore, the metrics and evaluation framework for the portfolio reconciliation part can also be improved to obtain more insightful results.

Future research on this topic also includes exploring the possibility for the autoencoder to explain why certain trades are flagged as anomalies, as well as providing suggestions for correct alternatives. Related to that, it could also be worth exploring whether an autoencoder could be used to predict certain fields in the trades based on the other fields. As the denoising autoencoder showed best performance on the product class field, that field could be a suitable starting point. More sophisticated models, with larger amounts of training data, could also be explored as a potential area of improvement.

References

[1] OSTTRA, "Portfolio Reconciliation by OSTTRA triResolve." Available at: `https://osttra.com/services/trade-lifecycle-services/portfolio-reconciliation/` [Accessed: May. 15, 2025].

[2] J. C. Hull, *Options, Futures, and Other Derivatives*. Edinburgh Gate: Pearson Education Limited, 9 ed., 2018.

[3] ISDA, "Portfolio Reconciliation in Practice," tech. rep., Internation Swaps and Derivatives Association, Inc., 2008.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection : A Survey," tech. rep., 2009.

[5] D. G. Kleinbaum and M. Klein, *Logistic Regression A Self-Learning Text Third Edition*. 2010.

[6] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 413–422, 2008.

[7] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 3 2020.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 10 1986.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," tech. rep., 2014.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, p. 318–362. Cambridge, MA, USA: MIT Press, 1986.

[12] T. Sattarov, D. Herurkar, and J. Hees, "Explaining Anomalies using Denoising Autoencoders for Financial Tabular Data," 9 2022.

[13] "NLLLoss - PyTorch Documentation," 2025. Available at: `https://docs.pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html#torch.nn.NLLLoss` [Accessed: May. 15, 2025].

[14] ISDA, "Understanding the ISDA Master Agreement." Available at `https://www.isda.org/2022/05/11/video-understanding-the-isda-master-agreement/` [Accessed: May. 15, 2025].

[15] ESMA, "About ESMA." Available at `https://www.esma.europa.eu/about-esma` [Accessed: May. 15, 2025].

[16] "IsolationForest - scikit-learn documentation," 2024. Available at: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html` [Accessed: Apr. 16, 2025].

[17] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *KDD'19: Proc. 25rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, p. 2623–2631, Association for Computing Machinery, 2019.

[18] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 12 2014.

[19] "CosineAnnealingLR - PyTorch Documentation," 2025. Available at: `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html` [Accessed: Apr. 16, 2025].