TECHNISCHE UNIVERSITÄT
DORTMUND

MASTER'S THESIS

Test-time training for anomaly detection in Time Series

Supervisors:
Rrof. Dr. Emmanuel Mueller
Simon Kluettermann

Author: Vikas Kumar

April 22, 2025

Contents

Chapter 1

Introduction

In today's era, detecting outliers in time series data has a wide range of real-world applications, including stock market manipulation (Ahmed et al., 2016), credit card fraud detection (Sahin and Duman, 2011), predictive maintenance (Kamat and Sugandhi, 2020), and more. One of the applications in machine maintenance is the detection of defects in microchip/device production. This helps prevent financial losses caused by fraud in financial sector or defects in the manufacturing sector, highlighting the importance of high-performing outlier detection methods.

Anomalies can be categorized into two parts: point anomaly and contextual anomaly. Point anomaly denotes an abrupt change in the pattern at a given timestamp. It can be a sudden spike in a positive or negative direction, or no change compared to the previous timestamp. Contextual anomaly occurs when an anomaly persists for more than one timestamp and relies on the neighboring context to identify it as an anomaly.

Identifying outliers in time series data is challenging due to the presence of contextual or time-varying information, including seasonality, patterns, trends, amplitudes, frequencies, as well as noise and irregularities. Therefore,

knowledge discovery in data helps to better capture the underlying data distribution.

Traditional statistical methods for anomaly detection (AD), such as the z-score or visual techniques like the box plot, are often insufficient when dealing with complex data. This has led to a demand for high-performing algorithms, where clustering, isolation methods, and other approaches have shown better results. Newer methods leverage the power of deep learning in unsupervised settings to improve performance, such as ensemble models with deep neural networks (DEAN) (Klüttermann and Müller, 2022).

Additionally, self-supervised algorithms such as test-time training (TTT) use the test dataset to learn and enhance the model, leading to improved performance (Sun et al., 2020). This method has been gaining popularity for addressing distributional shifts in data, making algorithms more robust. It has also been adapted for anomaly detection, as demonstrated in DOUST(Klüttermann and Müller, 2024). In this thesis, we have attempted to implement test-time training on time series data, aiming for a method that is more generalized, robust, and scalable.

1.1 Thesis Structure

This report contains 10 chapters in total, explaining the methodologies and our work in detail. Chapter 2 reviews the literature on anomaly detection methods from traditional to recent advancements. This chapter provides an overview of two datasets, simulated and real-world, used for testing and experimentation, which is described in Chapter 3.

Chapter 4 explores the necessary modifications for sequential data in DOUST, analyzing the fundamental problem and challenges through the analysis of various plots and graphs. Now that the problem has been identified in time series data during test-time training, the next step is to find a solution.

Chapter 5 provides a detailed explanation of the first solution to the problem, which involves modifying the loss function to address distributional shifts in temporal data. In the chapter 6, we experiment with and analyze all the loss functions on a few datasets, narrowing down the list to the most promising ones. After selecting a few promising loss functions that have the potential to perform well on the given datasets, we will proceed with further analysis.

Chapter 7 of this thesis provides a description of all the features extracted from time series data and adds value to the proposed methodology. Chapter 8 presents the analysis of the results obtained by applying our methods to simulated datasets. This section also explains the performance of the methods based on base oscillations and the number of dimensions. Furthermore, Chapter 9 explains the performance by analyzing the anomaly scores on real-world datasets.

Finally, the last chapter summarizes the thesis by concluding the results of the study. The thesis concludes with a brief note on potential future work and research extensions.

Chapter 2

Related Works

This chapter will provide a brief overview of the relevant methods that form the basis of the experiments in the thesis. The task of outlier detection can be divided into the behavior of the algorithm, such as the usage of labels and the type of algorithm.

Every algorithm can be divided into different categories on the basis of learning or usage of labels, which are as follows:

1. Supervised learning:

   Using labels during training refers to the classification of instances as normal or abnormal. Several traditional ML algorithms have been good at classification, such as decision tree, and random forest. There has been gained popularity for the past decades in developing more powerful deep learning algorithms such as CNN (O'shea and Nash, 2015) - imagenet, alexnet for images; RNN (Medsker and Jain, 1999) such as GRU (Dey and Salem, 2017), LSTM (Hochreiter and Schmidhuber, 1997), or even attention based transformers (Vaswani et al., 2017) such as BERT (Devlin et al., 2019) for sequential data.

2. Unsupervised learning:

When the algorithm tries to categorize these instances without the use of instance labels, then this is called unsupervised learning. This type of learning can be generalized as a grouping problem. This clustering has several other sub-families, such as Clustering based, distance based, density based, forecasting based, and reconstruction based.

3. Semi-supervised learning:

   It is a hybrid of both above mentioned techniques in a way that labels are used during training. However, the unlabeled data is used to refine the already trained model. It can be used with a different set of assumptions on the data distribution as compared to other types of learning.

2.1 Unsupervised learning methods

Various methods of unsupervised learning can be divided into several categories based on the main working style.

2.1.1 Distance

KNN, and K-means are some methods where anomalies can be left out of the clusters (Ramaswamy et al., 2000), (Lloyd, 1982). Being the simplest of AD methods, they are traditional ML algorithms, with a faster runtime. However, they always suffer from the curse of dimensionality, leading to a failure e.g., when the dimensions are 100 or even more (Köppen, 2000).

2.1.2 Forecasting based

One approach to solving AD involves predicting the value at each time stamp, then identifying the location of anomalies by thresholding the difference be-

tween actual and predicted values. Statistical ways of time series forecasting include ARIMA (Box et al., 2015a), Exponential Smoothing (Gardner Jr, 1985), and Kalman-Filter (Maybeck, 1982), but they all have some limitations such as ARIMA's linear and stationary assumptions, sensitivity to outliers, and other assumptions of noise.

Machine learning can be used to forecast non-linear patterns using methods such as Random Forest and Support Vector regression. Other deep learning methods include RNN (GRU or LSTM) and transformers. One such algorithm developed mainly for anomaly detection is **Telemanom**, which involves training an LSTM model per feature to forecast time series and detecting outliers using a non-parametric dynamic thresholding technique (Hundman et al., 2018).

2.1.3 Encoding-decoding

AD can be viewed from a different point of view, in which a representation from the inputs (encoding) is learned, then the algorithm tries to decode and reconstruct the input from the latent representation, detecting outliers based on a threshold. There has been a lot of research on Autoencoders (Zhou and Paffenroth, 2017); further, it has been extended to variational autoencoders for anomaly detection (An and Cho, 2015).

2.1.4 Other methods

Apart from classical ML methods, various approaches have been proposed to tackle AD problems, such as using an ensemble of trees to isolate the anomalies in relatively fewer number of steps: Isolation Forest (Liu et al., 2008). AD can also be addressed by discriminating outliers based on density, where similar normal points typically have higher density and outliers are rare with lower density. One such method is LOF (Breunig et al., 2000) or

COF (Tang et al., 2002), but these methods also suffer from the curse of dimensionality.

2.2 One-class classification

Another approach to AD is classifying the normal data; it's possible because anomalies are rare and don't follow a specific pattern. One-class classification (Khan and Madden, 2014) methods, such as One-class SVM (Schölkopf et al., 2001) and DeepSVDD, try to classify and model the normal data by fitting the normal points to the smallest possible hypersphere, which is the feature representation of the normal points. The learning is done using deep Neural Networks, where anything outside this hypersphere is labeled as an anomaly (Ruff et al., 2018).

2.3 DEAN

The submodel's loss function of DEAN (Deep Ensemble Anomaly Detection), inspired by DeepSVDD, aims to learn a constant relation from train data distribution, which is kind of intuitive because all the normal data points should have a common constant relation (Klüttermann and Müller, 2022). The loss function of DEAN is typically defined as :

$$loss_{DEAN} = (f(x_{train}) - 1)^2.$$

As a result, the neural network will try to learn a function f, such that $f(x) \approx 1$. So, the intuition behind this type of loss function is that outliers should deviate much from 1, whereas for normal points, $f(x)$ is near to 1 (Klüttermann and Müller, 2022).

In other words, DEAN tries to learn a common relation for the training data, which is to learn $f(x)$ such that the mean of $f(x_{train})$ is 1. This is done for all the submodels of DEAN; however, each submodel doesn't learn the best MLP, but a collection of such submodels that can potentially perform really well when combined in an ensemble.

2.4 Test-time training

This novel technique can be categorized under self-supervised learning. TTT is typically used to address the distributional shifts that usually occur in data distribution (Sun et al., 2020). It can be used for both supervised and unsupervised algorithms. TTT adds an extra *auxiliary* task to the standard *main* task, for instance, prediction of numbers on CIFAR-10 data.

Let's suppose a test data set (X, y), where $X = (x_1, x_2, ..., x_n)$ is the dependent variable and $y = (y_1, y_2, ..., y_n)$ is the target variable. For the final optimization problem, loss function for main and auxiliary tasks can be added as follows (Sun et al., 2020):

$$\min_{\theta_m, \theta_s} \frac{1}{n} \sum_{i=1}^{n} \ell_m(x_i, y_i; \theta_m) + \ell_s(x_i; \theta_s).$$

Where, $\ell_m(x, y; \theta_m)$ is the loss function of *main* task with $\theta_m = (\theta_1, ..., \theta_K)$ being the learning weights of K layers of neural network. While, $\theta_s = (\theta'_1, ..., \theta'_K)$ are the parameters of *auxiliary* task, learned during test-time and it contains some of the shared parameters from θ_m, apart from them, they are called unshared parameters (Sun et al., 2020).

During the training phase, both θ_m and θ_s are optimized and learned using X and y. Now, the shared parameters θ_{shared} for auxiliary task can be optimized, because, y labels during test-time training are not used, therefore, ℓ_m wouldn't be optimized at that moment, further refining the model.

A paper on TTT shows a case when it will fail, and they explain how TTT breaks under huge distributional shifts or changes in the data. While they also tried to improve the drawbacks of test-time training using contrastive learning (Liu et al., 2021). Use of TTT for anomaly detection is possible due to the assumption that anomalies are rare.

2.5 DOUST

Originally, TTT was built for supervised algorithms, but it can be adapted for unsupervised learning since there is no need for labels in this specific case. Even the loss functions would have the same objective, unlike in TTT.

As a result, DOUST (Deep OUtlier Selection with Test-time training) can detect outliers in an unsupervised setting, meaning it does not require labels. The Figure 2.1 shows the fundamental concept behind the working of DOUST, and how it leverages the idea of TTT using deep neural network, aiming to maximize the difference between data distribution of normal and abnormal (Klüttermann and Müller, 2024).

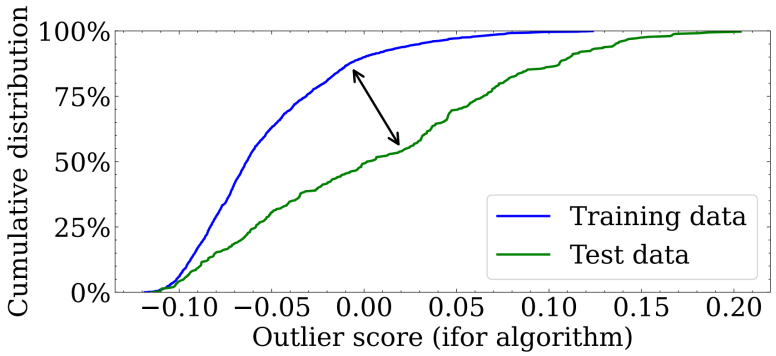


Figure 2.1: Example of cummulative distribution of train and test outlier scores, calculated using IForest (Klüttermann and Müller, 2024).

In the context of AD, **Positive Unlabeled Learning** is the type of learning when positive labeled and unlabeled data are used for learning a pattern (Bekker and Davis, 2020). DOUST has adapted PU learning according to the AD, normal (positive labeled) and a set of unlabeled data.

The authors attempts to create a 1- dimensional data representation of normal data points, ensuring a significant difference between normal and outlier data distribution. DOUST's working has 2 phases:

1. **Training phase**:

   A constant value of 1/2 for the normal data representation is learned. Mathematically, the first loss function for training on normal data is as follows:

   $$L_{train-phase} = (f(x) - \frac{1}{2})^2.$$

   The DOUST, similar to DEAN, also doesn't use any bias term in each of the NN layers and contains only 3 intermediate fully connected layers and a final output layer. The last layer contains a modified version of sigmoid activation function defined as follows:

   $$A^+ = \frac{1}{1 + \exp(1 - x)}.$$

2. **Refinement phase**: In this phase, the learned NN weights to achieve maximum performance, are refined. The loss function used in the phase is defined as follows:

   $$L = \frac{1}{||X_{Train}||} \sum_{x \in X_{Train}} (0 - f(x))^2 + \frac{1}{||X_{Test}||} \sum_{x \in X_{Test}} (1 - f(x))^2.$$

   This nudges the normal data to have low score and anomalies to have high score.

Authors have established a general rule for the adequate size of a dataset required for DOUST to achieve good performance: $N \gg 1/\nu^2$ observations, ν is percentage of anomalies and N is the size of dataset required for DOUST (Klüttermann and Müller, 2024). This algorithm uses an ensemble of models for robustness and high performance.

Chapter 3

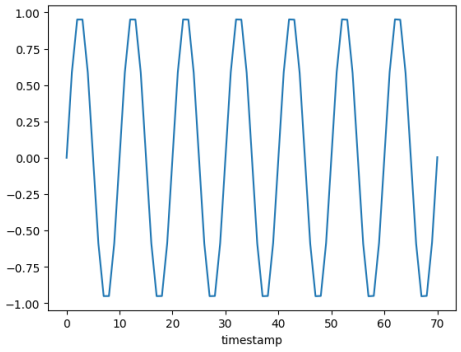
Dataset Overview

In this thesis, there are 2 datasets (simulated and real) for benchmarking our results and testing our experiments. The simulated data is sourced from GUTENTAG, contains various base oscillations and numerous anomaly types (Wenig et al., 2022). It is employed to conduct several experiments within a controlled environment. A real-world dataset, namely, NASA-SMAP and MSL, is used to evaluate the performance of our experiments (Hundman et al., 2018).

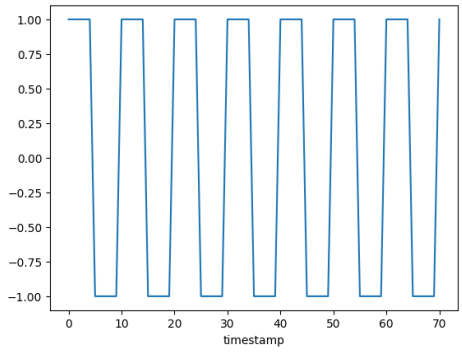
3.1 GUTENTAG (simulated)

3.1.1 Base Oscillations

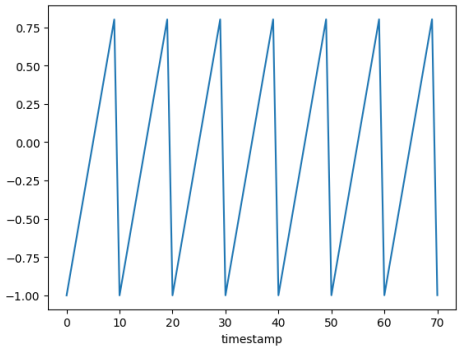
As shown in Figure 3.1, the dataset contains several base oscillations such as *Sine* and *Cosine Wave* for periodic time series, along with *Square* and *Sawtooth* waves for linear time series with changes. There are also *ECG* base oscillations present in the benchmark dataset of GUTENTAG, which introduce additional diversity. *Random Walk* and *Cylindrical Bell Funnel* are among the most complex time series in the entire GUTENTAG dataset, as



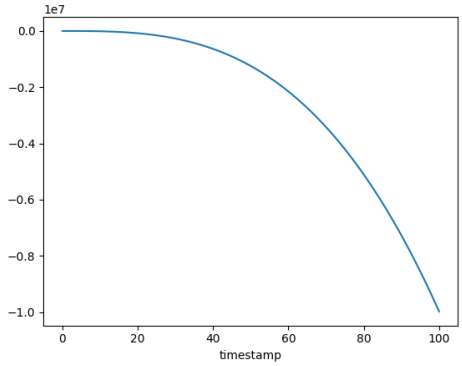
(a) Sine wave



(b) Square



(c) Sawtooth



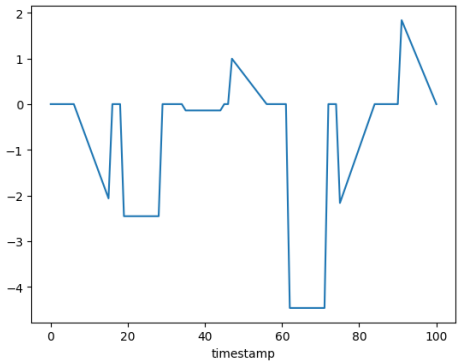
(d) Polynomial

Figure 3.1: Examples of time series from the GUTENTAG dataset (Part 1).

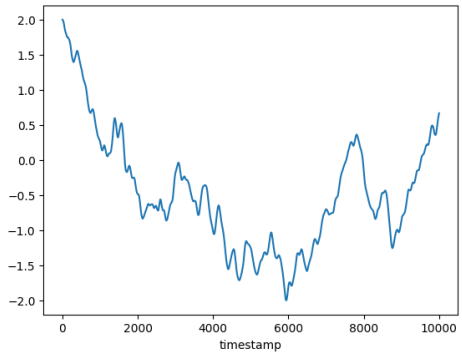
shown in Figure 3.2, further diversifying the dataset and good for training a robust algorithm. There is also *Polynomial*, in which anomalies are difficult to spot with the human eye when plotted (Wenig et al., 2022).

3.1.2 Anomaly options

Along with several base oscillations, the GUTENTAG framework provides a variety of anomaly options. Each anomaly type has its specific properties, which can be adjusted to gain more control over the contamination of the

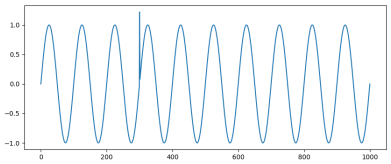


(a) CBF

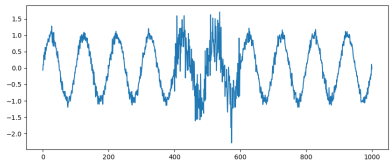


(b) Random Walk

Figure 3.2: Examples of time series from the GUTENTAG dataset (Part 2)



(a) Extremum anomaly



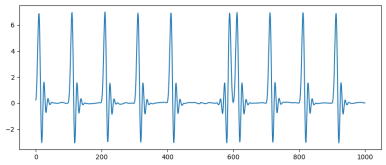
(b) Variance anomaly

Figure 3.3: Examples of anomaly types: Extremum and Variance from the GUTENTAG dataset (Wenig et al., 2022).

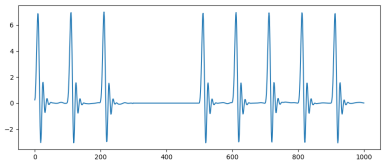
dataset. For example, *amplitude* and *frequency* can be altered for a few time stamps and injected at specific regions in sine and cosine waves.

Additionally, there are other ways to contaminate the dataset, such as introducing sudden extreme values. For instance, a point anomaly can be defined using the *extremum* option with a specific value, as shown in Figure 3.3(a). The variance of certain time stamps can also be manipulated using the *variance* option.

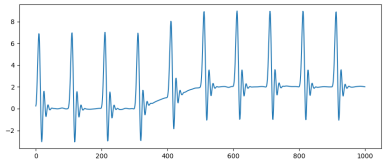
Other anomaly types include *pattern*, which introduces a specific pattern over a few time stamps, as illustrated in Figure 3.4(a). *Platform* is a contextual anomaly where the time series appears to remain constant for a short period as shown in Figure 3.4(b).



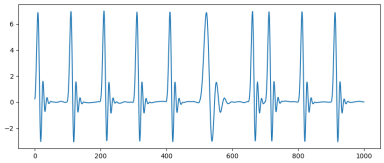
(a) Pattern anomaly



(b) Platform anomaly



(c) Trend anomaly



(d) Pattern-shift anomaly

Figure 3.4: Examples of anomaly types: Pattern, Platform, Trend, Pattern-shift from the GUTENTAG dataset (Wenig et al., 2022).

Some *trends*, such as increasing or decreasing slopes, can also be added to the existing time series. A shift in pattern is another form of abnormal behavior for example, the pattern shift in an ECG time series shown in Figure 3.4(d).

The benchmark datasets of GUTENTAG represent a collection of 193 diverse time series, each with a varying percentage of anomalies. Every dataset consists of 10,000 observations, with a varying number of columns, i.e., univariate or multivariate time series.

3.2 NASA SMAP & MSL dataset (real-world data)

This dataset has been taken from Kaggle, and uploaded by the authors of the paper titled: "Detecting Spacecraft Anomalies Using LSTMs and Non-parametric Dynamic Thresholding" (Hundman et al., 2018). SMAP stands for Soil Moisture Active Passive; its data was originally obtained from satel-

lite measurements. MSL refers to Mars Science Laboratory, with its data sourced from Mars rovers. Both datasets include anomaly labelled versions: train and test sets, which were labelled precisely by domain experts. They examined each dataset (channel) by manually inspecting the time series to label the anomalies.

Table 3.1: Summary statistics of datasets (simulated and real).

Name	**type**	**# dims**	**Avg. size**	**# Datasets**
GUTENTAG	simulated	1-20	10,000	193
NASA SMAP	real	25 or 55	7,813	55
NASA MSL	real	25 or 55	2470	27

Table 3.1 summarizes the contents of both types of datasets. SMAP consists of 55 datasets, with a total of 429,735 observations and 69 anomalies. This results in an average dataset size of 7,813 values and an overall anomaly ratio of less than 1% across all 55 datasets. In MSL, there are 27 datasets with only 36 anomalous points out of 66,709 total observations. On average, MSL datasets contain 2,470 observations, with fewer than 1% of them being anomalies. Each dataset contains a single time series.

Chapter 4

Adaptation of DOUST in time series

This chapter explores the possibility of DOUST adapting to time series data and presents the observations from a simple implementation of DOUST on several datasets after its application. It also discusses the research questions arising from the challenges in DOUST and possible solutions to them. The implementation code for this thesis is available at: `https://github.com/vk-vikaskumar/Test-Time-Training-Of-Anomaly-Detection`

4.1 Foundational Problem of DOUST

This section provides a detailed account of the setup and findings for the experiments. The experiments on time series data are divided into two key segments: identifying the problem in DOUST with time series data and exploring solutions. Since DOUST is initially built for tabular datasets, its time series adaptation needs some exploration. This adaptation can be challenging considering the foundation of the algorithm, which is "maximizing

the difference between normal and abnormal distribution". We first simulated a few datasets with high complexity and then analyzed the results.

4.2 Challenges in Time Series

4.2.1 Experimental Setup

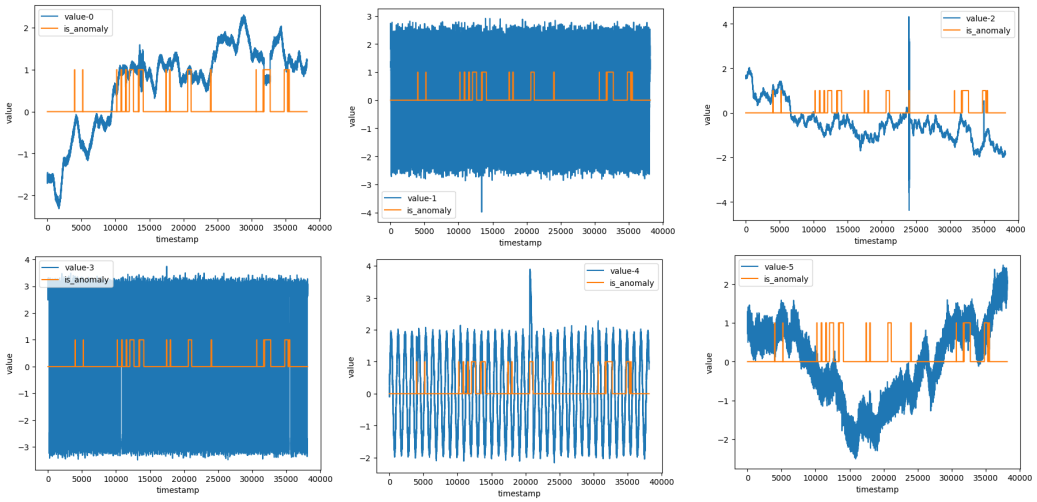


Figure 4.1: Visualization of the six-dimensional multivariate dataset D1, including labeled data points.

To analyze DOUST's adaptation to time series data, we designed a controlled experimental setup based on GUTENTAG. Synthetic datasets D1 and D2 are generated with anomalies constituting around 10% of the total data; their statistics are listed in Table 4.1. A visual representation of D1 can be seen in Figure 4.1. D2, in particular, consists of three independent random walks, one cosine and a sine wave, each with distinct properties such as frequency, amplitude, and variance.

To examine the effect of temporal order, an additional version of each dataset was created by randomly shuffling the timestamps, denoted as $\text{Dataset}_{\text{shuff}}$.

Table 4.1: Summary statistics of datasets D1 and D2.

Name	# dims	Size train	Size test	% anomaly
D1	6	56,000	10,000	12%
D2	5	70,000	38170	10%

This allowed us to investigate how removing temporal dependencies affects DOUST's anomaly detection capabilities. The shuffling process is illustrated in Figure 4.2.

For D1 and D2, a standard normalization of the dataset is applied in order to re-scale them. After that, DOUST is applied to these standardized D1 and D2. After the surprising results, D1 and D2 are shuffled and then, they are standardized for DOUST's application.

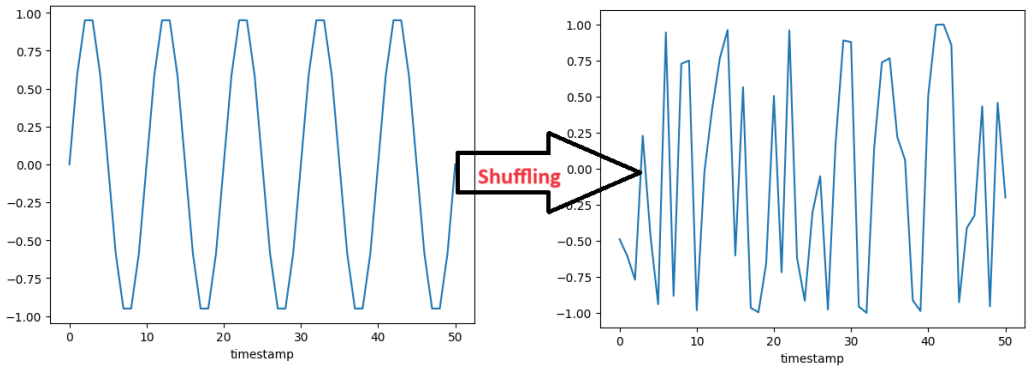


Figure 4.2: Illustration of the shuffling process applied to time stamps in the dataset.

4.2.2 Analysis of adaption to Time series

During experimenting with DOUST on D1 and D2, the algorithm with feature extraction showed good performance on other simulated datasets simpler than D1, for instance, one sine wave with 1 anomaly (*extremum*) for 100 time stamps. But then, as the datasets became more and more complex,

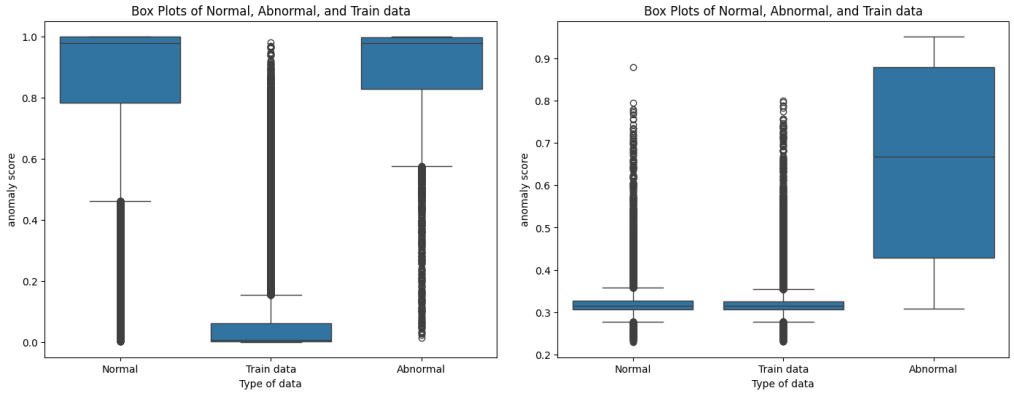


Figure 4.3: Box plots of anomaly scores for D2 dataset. The left plot represents scores before shuffling (D2), while the right plot corresponds to shuffled data ($D2_{shuff}$).

the performance started to deteriorate, indicating drawbacks in DOUST for sequential data.

From Table 4.2, it is evident that prior to shuffling, the anomaly detection performance was considerably lower (0.7261 for D1 and 0.4784 for D2). It can be seen that DOUST is performing badly for D1 and worse for D2. What could be the reason? Maybe DOUST is unable to perform well due to some issues with it. We need to examine every aspect of this. Therefore, distributions of train, test, and normal points need to be studied well.

Anomaly score might also give some information on why DOUST is performing badly on D2. We also know that DOUST's strength lies in its ability to learn from the difference between the distribution of normal and abnormal points. But, that was done for tabular datasets. How can we transform back to tabular data from time series? It can be done if temporal order is not respected. Shuffling inside a dataset is therefore necessary. Another possible reason is that the training and test data may contain a trend, such as a linear one, which creates a disparity between normal points in the training and test data. Random shuffling helps distribute the linear trend in the data,

breaking it down and spreading it evenly across both the training and test data, resulting in similar distributions of normal points in both.

To determine if DOUST performs well on a tabular dataset, the data is standardized, shuffled it, and then re-applied DOUST without extracting any features. After its application to the shuffled datasets, the AUC-ROC scores improved significantly, exceeding 0.9 for both datasets.

Table 4.2: Comparison of AUC-ROC scores for D1 and D2 with and without shuffling.

Dataset	**Before Shuffling**	**After Shuffling**
D1	0.7261	**0.9273**
D2	0.4784	**0.9702**

This reveals an inconsistency. To confirm the presence of the problem, we need to further analyze the distribution of train and test sets, as this is the core principle of DOUST. The distribution of anomaly scores can be seen in the Figure 4.3, where the box plots of anomaly scores before and after shuffling are compared.

The box plot of anomaly scores for D2 dataset on the left, reveals that the distribution of train and normal scores differs, whereas the distribution of normal and abnormal scores is similar, as reflected in the AUC-ROC score. The goal of DOUST is to maximize the difference between normal and abnormal. Therefore, D2 doesn't perform well. While after shuffling the distribution of normal and train data seems quite similar and way more dissimilar than abnormal data, and that also gives some explanation for the good performance of DOUST on $D2_{\text{shuff}}$.

More evidence can be collected by analyzing the anomaly score of the test dataset with and without shuffling the data. Therefore, we visualize the anomaly score trends for both training and test datasets in Figure 4.4. It shows that, in the unshuffled dataset (top 2 graphs), anomaly scores exhibit

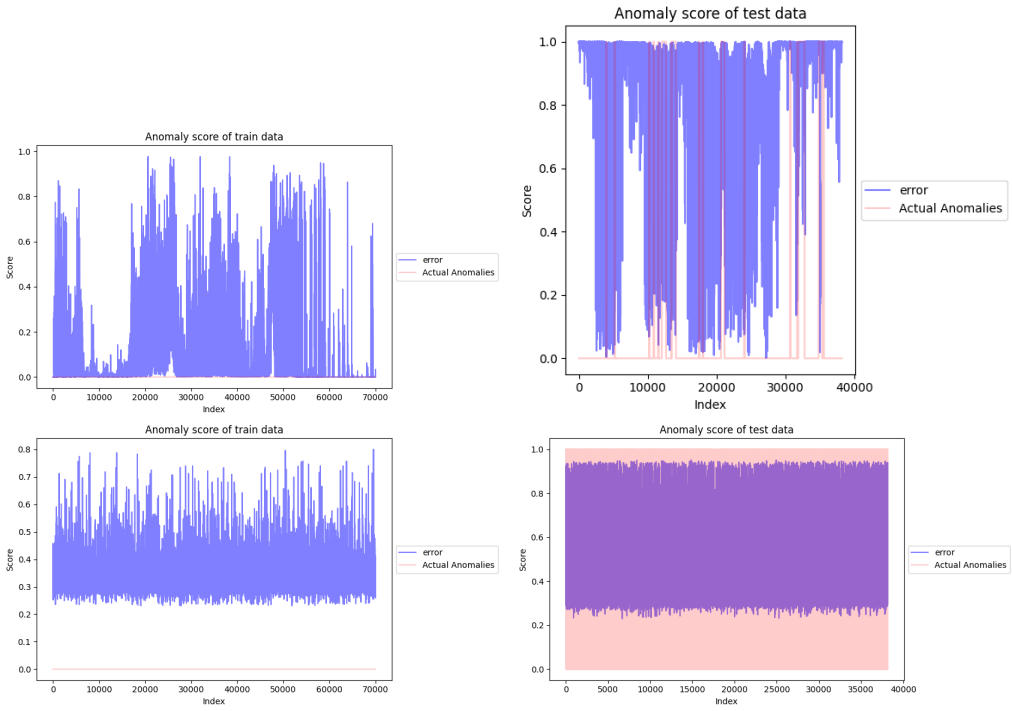


Figure 4.4: Anomaly scores for training and test datasets: Upper row represents D2, while the lower row corresponds to $\mathrm{D2}_{\mathrm{shuff}}$.

erratic behaviour, with some normal points being assigned high anomaly scores.

In contrast, after shuffling (bottom), the anomaly scores become more stable and consistent. The bottom left graph is the train set of D2, which shows relatively similar scores for the majority of data points, which is even better for the test dataset (bottom left). However, it is not so clearly visible.

Therefore, a small subset (first 1000 observations) of test $D2_{\mathrm{shuffled}}$ from the bottom right of Figure 4.3 can be seen in Figure 4.5. They also highlight a more stable pattern, with lower anomaly scores for all normal points and higher scores for most anomalies. However, a few anomalies still receive scores comparable to normal points; we can ignore them for now.

All of this points to a **Temporal Paradox** within DOUST—its underlying assumptions seem to falter when applied directly to sequential data. This issue must be addressed before proceeding with any further testing or experimentation on time series data.

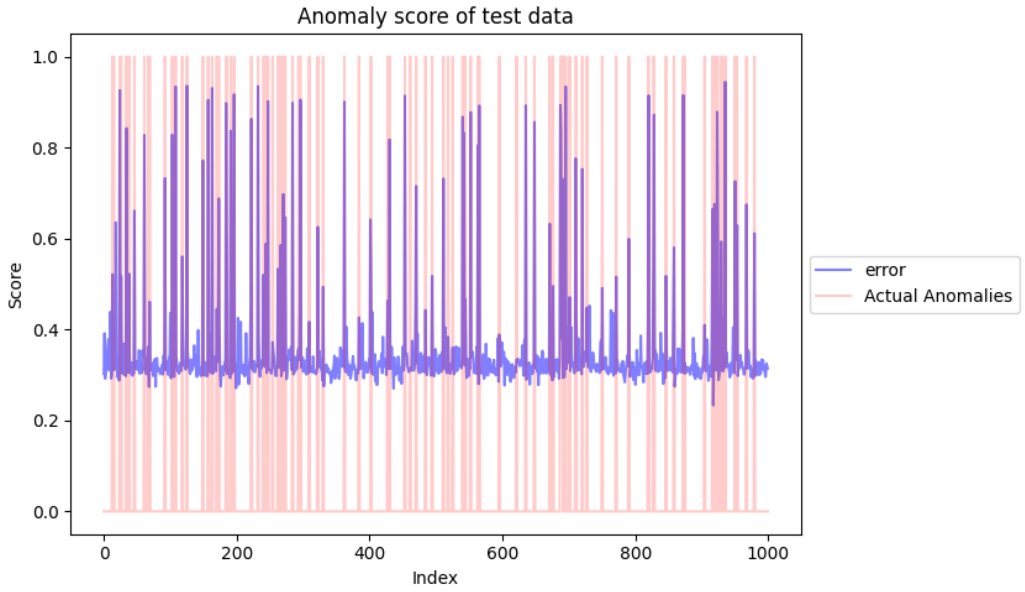


Figure 4.5: Anomaly scores for the first 1000 timestamps of $D2_{\text{shuff}}$.

4.2.3 Conclusion

The application of DOUST with feature extraction is analyzed on two simulated datasets. The results indicate low performance when it comes to time series, but after studying the distribution of AUC-ROC scores (anomaly score) on a box plot or against the labels, it shows that the distribution of anomaly scores of normal data is similar to that of abnormal data. The poor performance can be attributed to a flaw in the foundation of the DOUST, specifically when handling sequential data.

We know that DOUST claims to have good performance on tabular datasets, so time series data is converted into tabular by shuffling the timestamps. Surprisingly, shuffled (tabular) datasets provided much better performance as compared to original time series data without even extracting features. It can be concluded that order of timestamps in time series data significantly impacts DOUST's performance. The increase in anomaly detection accuracy after shuffling suggests that DOUST may not be effectively leveraging temporal dependencies.

This presents a crucial challenge: how can this temporal paradox be removed? Can we capitalize on this paradox?

Moving forward, further investigations will focus on the following aspects:

- Exploring modifications in loss functions of DOUST to either eliminated/minimize the effect of this paradox or take advantage from this.
- Developing hybrid feature extraction to enhance performance.

This study underscores the necessity for refining DOUST's adaptation to time series data. The findings highlight a gap that must be addressed to ensure meaningful anomaly detection in real-world time-dependent scenarios.

Chapter 5

Methodology: Losses

It is known that further research is needed for application of DOUST to time series. Since the DOUST's basic foundation is it's loss function, which is developed considering tabular data, there is a need to construct a new loss function for sequential data. This approach is expected to resolve the temporal paradox issue by targeting the foundation of DOUST. We need to first try several loss functions tailored mainly for sequential data. Constructing a robust loss function involves developing and studying them based on different properties of the time series data. This chapter describes all the loss function methodologies that are used to experiment on several datasets.

5.1 Custom Loss functions

Various loss functions are tested to determine which one would work best for adapting DOUST to time series data. It's quite difficult to custom build a good loss function while testing it against some complex datasets; therefore, we will try to build loss functions according to simpler toy example datasets. Afterwards, a couple of potentially good loss functions will then be tested against more complex datasets.

From DOUST, the newly updated loss function is defined as follows:

$$L' = \frac{1}{||X_{Train}||} \sum_{x \in X_{Train}} ||(0 - f'(x))||_{L_p}^p + \frac{1}{||X_{Test}||} \sum_{x \in X_{Test}} ||(1 - f'(x))||_{L_p}^p$$

Here $f'(x)$ should not be confused with the derivative with $f(x)$. It's just for notation purposes.

All of the modifications in loss functions can be expressed through new $f(x)$, denoted as $f'(x)$, and we are alos trying to use two variations of each loss function using the L_p norm, where $p = 1$ for L_1 and $p = 2$ for L_2. A stable loss function is required—one that yields low values for normal data points and high values for outliers. Therefore, modifying $f(x)$ makes sense, and also playing around with the objective such as inverting, changing the order of $f(x)$ might be useful. Different variations of the loss function are possible. For example, in the training set, $f(x)$ can be subtracted from 1 instead of 0, while in the test set, we can use only $f(x)$ inside the norm.

DOUST first pre-trains on the given data using only the training set with a mean squared loss. These learned weights are then redefined using a custom loss on the positive unlabeled data from both the training and test datasets.

It makes more sense to address the temporal paradox of correlation through the loss function. Therefore, modifying the loss function during re-training helps DOUST learn more complex structures while maintaining performance. The goal is to make the data distribution more learnable during the re-training of weights. Below is a list of the loss functions we experimented with:

1. **"l1_std"** and **"l2_std"**:

   Normalization of values typically results in a more learnable data distribution. Therefore, subtracting the mean value from $f(x)$ makes sense

for standardization. In the case of a simple linear line, this helps bring all the points closer to each other. Dividing by the standard deviation is a way to counter the variance in $f(x)$. The general loss can be defined as follows:

$$f'(x) = \frac{|f(x) - \overline{f(x)}|}{\sigma_{f(x)}}.$$

Then, replace $f(x)$ with this $f'(x)$ in the original DOUST loss function.

2. **l1_without_median** and **l2_without_median**:

   The mean is highly influenced by extreme values; therefore, subtracting the median from the data is highly intuitive. The median, which represents only the 50^{th} percentile, brings values closer to its presumed mean of 1/2 for normal data points.

$$f'(x) = f(x) - \text{median}(f(x))$$

3. **l1_min_max** and **l2_min_max**:

   Another way to standardize $f(x)$ is by using min-max reduction, where the minimum and maximum values of $f(x)$ are used to scale the data. Sometimes min-max standardization works better than standard normalization. But this might not be the best custom loss function; it was included to see if we observe any adverse results.

$$f'(x) = \frac{f(x) - \min(f(x))}{\max(f(x)) - \min(f(x))}$$

4. **l1_min_max_part1** and **l2_min_max_part1**:

   Another intuitive approach is to subtract some statistical values from the train data and apply them to the positive unlabeled dataset. The intuition is to remove a common additive factor from the dataset, which

might help in achieving uniform standardization. The new $f'(x)$ function is defined as:

$$f'(x) = \frac{f(x) - \min(f_{\text{train}}(x))}{\max(f_{\text{train}}(x)) - \min(f_{\text{train}}(x))}$$

5. **l1_std_inverted** and **l2_std_inverted**:

   The intuition behind the inversion of loss function is to test whether switching the target of normal and abnormal data points has any negative effect on anomaly score behaviour. Here, $f'(x)$ is the same as defined in the "l1_std" loss above, but the difference lies in how the loss function is structured. The updated loss function is as follows:

$$L' = \frac{1}{||X_{\text{Train}}||} \sum_{x \in X_{\text{Train}}} ||(1 - f'(x))||_{L_p} + \frac{1}{||X_{\text{Test}}||} \sum_{x \in X_{\text{Test}}} ||(0 - f'(x))||_{L_p}.$$

   The only change in this formula is the subtraction of 0 and 1, which was experimented to observe its effect on the loss function's behavior.

6. **"l2_min_max_inverted"** and **"l1_min_max_inverted"**:

   In this function, $f'(x)$ is the same as defined in "l1_min_max" loss. The only difference is that the subtracted 0 and 1 terms are replaced, similar to the method in "l1_std_inverted". Basically, it's the same "l1_min_max" loss function but with inverted objective.

7. **"l1_90_10"** and **"l2_90_10"**:

   Instead of using the minimum and maximum, we can also use the 10^{th} and 90^{th} quantiles of $f(x)$ for normalization. Min and max values might work well for normal points, but consider the min and max of abnormal points, which can be extreme. Normalizing based on those extremes could result in points that are less deviating receiving lower anomaly scores, making them more difficult to detect. The updated $f'(x)$ is

defined as:

$$f'(x) = \frac{f(x) - Q_{0.10}(f(x))}{Q_{0.90}(f(x)) - Q_{0.10}(f(x))}.$$

Consider $X = (x_1, x_2, \ldots, x_n)$, a univariate time series with n observations, i.e., the n^{th} time period. The formula for finding the quantile (denoted by Q_p, where p is the p-th quantile and is given in percentage) is as follows (Dodge, 2008):

$$Q_p = \begin{cases} \frac{x_{(j)} + x_{(j+1)}}{2}, & \text{if } x_{p(n+1)} \text{ is not an integer,} \\ x_{(j)}, & \text{if } x_{p(n+1)} \text{ is an integer.} \end{cases}$$

Here, $j = \lfloor p(n+1) \rfloor$, $x_{()}$ denotes the order statistics, and $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x.

8. **"l1__10__25__50__75__90"** and **"l2__10__25__50__75__90"**:

   The above loss function can further be extended to create a more symmetric statistics and thus, a distribution for better learning. Our goal was to find a more suitable *IQR*-based value that represents most of the data distribution. We only use a few quantiles, taking the average of every 10^{th} quantile would be too much. The goal is to stabilize the loss function such that normal points have a lower anomaly score. We took the mean of certain quantiles that represent the distribution well, such as 10^{th} and 90^{th}, 25^{th} and 75^{th}. The updated $f'(x)$ is defined as:

$$f'(x) = f(x) - Q_{mean},$$

   where Q_{mean} is the mean of $Q_{0.10}(f(x))$, $Q_{0.25}(f(x))$, $Q_{0.50}(f(x))$, $Q_{0.75}(f(x))$, and $Q_{0.90}(f(x))$.

9. **Wasserstein Distance / Earth Mover's Distance (L_2)**:

   This loss function is distinct from the others listed above. It leverages the **shuffling advantage**, also referred to as the **Temporal Paradox**,

which can be understood as the benefit gained by DOUST when the order doesn't matter. This distance metric is designed to find the optimal transport cost. The Wasserstein distance, W_p, can be defined as follows (Villani, 2009):

$$W_p(\mathbb{P}_x, \mathbb{P}_y) = \inf_{\gamma \in \Pi(\mathbb{P}_x, \mathbb{P}_y)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\|_{L_p} \right],$$

where $\Pi(\mathbb{P}_x, \mathbb{P}_y)$ denotes the joint distributions of x and y, and γ represents the quantity of "mass" required in order to transform distribution x into distribution y (Villani, 2009).

In this thesis, we use the L_2 norm for the difference between x and y. Additionally, since there was no specific function in TensorFlow for computing the Wasserstein distance, we developed a custom function to approximate it. We intuitively strive for a low score for *normal* case and a high score for *abnormal* cases. Therefore, the loss function L_w is defined as:

$$L_w = \frac{1}{\|X_{\text{Train}}\|} \sum_{x \in X_{\text{Train}}} W_2^{\text{app}}(f_w(x), O) + \frac{1}{\|X_{\text{Test}}\|} \sum_{x \in X_{\text{Test}}} W_2^{\text{app}}(f_w(x), \mathbb{I}),$$

where $W_2^{\text{app}}(f_w(x), O)$ is the Wasserstein distance approximation ().

$$W_2^{\text{app}}(f_w(x), O) = \frac{1}{T} \sum_{i=1}^{T} \|f_w(x_i) - 0\|_2,$$

Here, $f_w(x_i)$ denotes to the sorted values of $f_w(x)$ in ascending order for all $f(x)$ values in their respective set (whether the training or testing set). In other words, $f_w(x_i)$ represents the order statistics of $f(x)$. O and $\mathbb{I}$ are matrices where each element is 0 and 1, respectively, and they have the same dimensions as the corresponding $f(x)$. The intu-

ition behind this loss function is that $f(x)$ is sorted for train and test set which was previously done in this thesis to show the excellence of DOUST when used on Tabular dataset.

After defining the custom loss functions, we will analyze them using toy examples and then select some for further experimentation on benchmark datasets.

5.2 Evaluation metrics

There are several criteria that can be used to assess anomaly detection such as AUC-ROC, AUC PR, Recall, Precision, and so on. However, each is used for a specific reason.

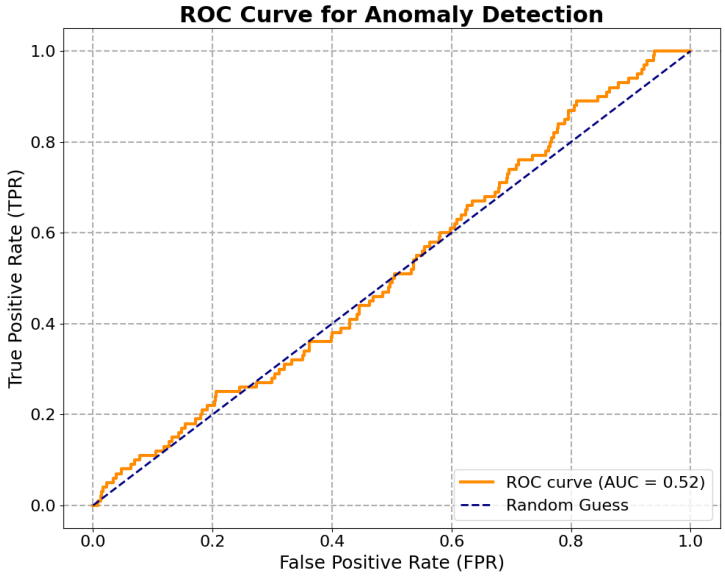


Figure 5.1: AUC-ROC curve plot.

For instance, precision would be a better metric for fraud transaction detection. Here, we are using AUC-ROC as a general purpose evaluation criterion

to assess the quality of the models. ROC is the Receiver Operating Characteristic Curve, and the area under this curve is used to check the quality of the model. ROC is the curve between True Positive Rate(TPR) on X-axis and False Positive Rate(FPR) on Y-axis, where they can be defined in terms of anomaly detection as follows (Fawcett, 2006):

$$TPR = \frac{\text{Number of actual anomalies predicted as anomaly}}{\text{Total anomalies}},$$

$$FPR = \frac{\text{Number of normal data points predicted as anomaly}}{\text{Total number of normal data points}}.$$

A visualization of AUC-ROC plot is shown in the Figure 5.1, where area under the orange ROC curve is 0.52. The highest value of this metric can be 1, which is considered to be the best, and 0.5 is the baseline value of a random guess.

Chapter 6

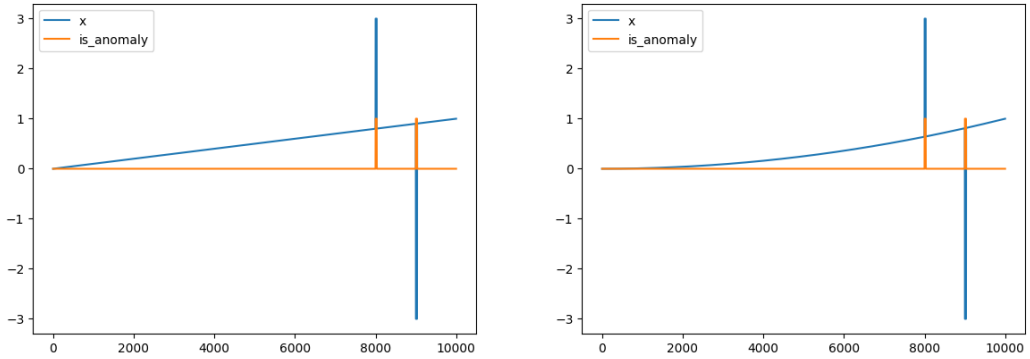
Experiments: Losses

This chapter provides a detailed analysis of the custom loss functions used for adapting DOUST to time series. These loss functions are tested, and an in-depth analysis of the anomaly score will also be performed on two toy example datasets.

6.1 Experimental Setup

To systematically assess the impact of various loss functions, we design a controlled experimental setup using two synthetic (toy) datasets. These datasets enable rigorous evaluation of the loss functions under controlled conditions before applying them to more complex datasets.

The first dataset, E1, consists of two columns: x and y. Here, x represents a simple linear progression with a slope of 1, while y is a binary variable containing a single, a well-defined contextual anomaly with an amplitude of 3 is doped in x. The second dataset, E2, builds on this idea by incorporating an additional transformation, replacing x^2 with x, which introduces a slightly more complex structure, while y retains the same anomalies as in E1.



(a) E1 dataset (b) E2 dataset

Figure 6.1: Visualization of toy datasets E1 and E2.

Since the loss function plays a crucial role in the DOUST framework, exploring different loss formulations is key to understanding and improving anomaly detection performance. To achieve this, we implement multiple variations of loss functions and evaluate them on datasets E1 and E2. The objective is to determine which loss functions maximize the contrast between the distribution of normal and anomalous data.

6.2 Analysis of Loss Functions

DOUST is applied to these two toy examples, which are E1 and E2, with numerous custom loss functions with no feature extraction; however, a simple standard normalization is applied to re-scale the datasets. It is expected that the AUC-ROC scores of the majority of loss functions will be 1. However, the behaviour of these losses needs to be studied and analysed critically.

A variety of loss function modifications are tested, and their anomaly detection performance is visualized through anomaly score line plots. These evaluations provide insights into how different loss functions behave. Our aim is to build a loss function perfect for these toy examples such that the

train anomaly score should be as low as zero and have a stable horizontal line. The score should be the same for *normal* data points and a very high score for doped anomalies.

For E1: The anomaly score for dataset E1 is visualized in Figure A.2 and Figure A.3. Notably, the loss function `l1_std` did not converge for any submodels; hence, no plots were generated for it.

Analyzing Figure A.2 reveal that the loss functions `l1_without_median` and its L2 counterpart successfully detect both anomalies. In contrast, the remaining loss functions fail to detect the first positive peak in dataset E1. However, a significant drawback is observed in `l1_without_median` and `l2_without_median`, where normal points exhibit a relatively high anomaly score (greater than 0.6). Furthermore, for some normal points towards the end of the orange line in Figure A.2, the score approaches 1.0, indicating a problematic tendency of overestimation in a simple simulated dataset.

Figure A.3 further highlights these observations. None of the loss functions detects both anomalies successfully, except for `l2_10_25_50_75_90`. Additionally, this loss function maintains relatively stable scores for normal points in both train and test datasets. Interestingly, the loss function `l1_10_25_50_75_90`, despite failing to highlight the first peak around 2000, achieves an AUC-ROC of 1.0. It may be due to the anomaly score being slightly higher than 0.3 for first anomaly. This implies strong performance in evaluation metrics, even though its anomaly visualization does not clearly depict the first anomaly. Notably, it also maintains the most stable anomaly scores for normal points, which is a highly desirable characteristic in a loss function.

Furthermore, the anomaly curve for `wasserstein` can be observed in the Figure A.1(a); the behvaiour seems to be against expectations and is definitely not ideal.

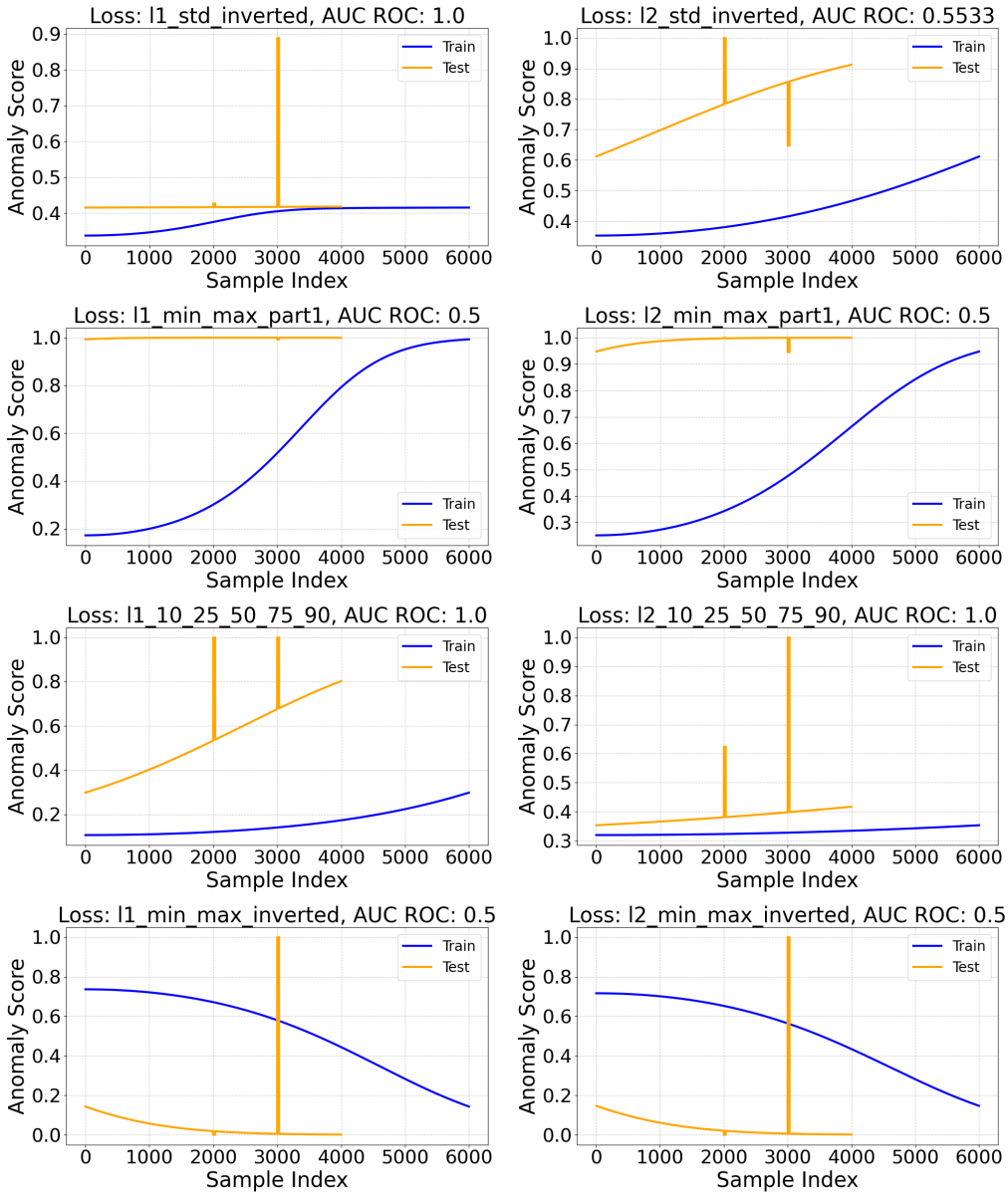


Figure 6.2: Anomaly scores of 8 loss functions on E2 dataset (Part 2).

For E2: The anomaly score outputs for eight different loss functions, each evaluated using both L1 and L2 norms on dataset E2—resulting in a total of 16 loss functions are depicted in Figure A.4 and Figure 6.2. Upon analyzing Figure A.4, we observe distinct performance variations across different loss functions. The loss functions `l1_std` and `l2_std` exhibit poor performance in terms of AUC-ROC scores. They fail to properly distinguish between anomalies and normal data, with the second peak displaying an unexpected downward deviation. Similarly, `l1_without_median` and `l2_without_median` yield an anomaly score of 1 at certain anomaly points. However, their overall behaviour suggests weak generalization to complex datasets, as they do not create a clear distinction between normal and anomalous points in the test data.

The second half of Figure A.4 shows that `l1_min_max` successfully captures the anomalies, whereas its L2 counterpart performs significantly worse. Despite its success in detecting anomalies, `l1_min_max` struggles to remove the underlying linear trend, limiting its applicability to more complex datasets. Similarly, the `l1_90_10` and `l2_90_10` loss functions fail to provide meaningful separation between normal and abnormal data, rendering them ineffective for our specific use case.

A secondary set of loss function variations is evaluated and visualized in Figure 6.2.

As shown in the Figure 6.2, `l1_std_inverted` appears stable but fails to detect the second anomaly. On the other hand, `l2_std_inverted` is both unstable and ineffective. The loss functions `l1_min_max_part1` and `l2_min_max_part1` perform the worst, as they misclassify almost all normal points with an anomaly score close to one, which is highly undesirable.

Finally, the anomaly score for `wasserstein` is graphed in Figure A.1(b), where, it can be observed that it retains the quadratic nature of E2, which shouldn't have happened.

6.3 Conclusion

Several loss functions were tested on the toy example datasets. For E1, `l1_std_inverted`, `l1_without_median` and `l1_10_25_50_75_90` alongwith their L_2 counterparts except `l1_std_inverted`, were able to detect both anomalies, however, every function behaved a little differently. All of the loss functions that were able to detect both anomalies for E1, also detected for E2 as well. `Wasserstein` was also able to detect both anomalies in E1 and E2.

However, among all tested loss functions, `l2_10_25_50_75_90` emerges as the most reliable. It maintains stability across normal points while effectively identifying both anomalies, making it the most promising candidate for further evaluation on more complex datasets. These results suggest that percentile-based loss functions incorporating multiple quantiles offer a balance between stability and anomaly detection capability, which is essential for reliable test-time training with time series data. However, other loss functions such as `l1_10_25_50_75_90` were also good, except for the part where it fails to assign a higher score to an anomaly in E2.

Chapter 7

Methodology: Feature extraction

Although applying DOUST to normalized data may yield a relatively good score for complex simulated datasets, it may struggle with simple real-world datasets that differ significantly from complex simulations, such as high-dimensional data with high sparsity or domain-specific data. Sequential data often contains a wealth of temporal information, including frequency, autocorrelation, and phase angle in the case of Fourier transform, which is typically useful for DOUST, as it involves learning from different properties. Feature extraction is generally a good practice for extracting knowledge, and in this case, it can be particularly beneficial.

In other words, our goal is to develop a universal set of features that can be extracted from any time series, converting it into tabular data. In this chapter, all the features that are extracted from the data are explained in detail.

The features extracted and formulated are based on uni-variate data and are defined as follows:

7.1 Moving Window Difference

Since time series data possess sequential behaviour, a sliding window function introduces a set of new features by capturing some local patterns. Many variations were tried, such as dividing each value by the standard deviation of the moving window, and so on. But, only 2 versions are retained: sliding window mean and sliding window median.

Consider a column feature X_1 and a sliding window of width w. The corresponding extracted feature is defined as:

$$X'_{1\ \text{median}} = X_{1i} - \texttt{median}(X_{1w}),$$

where X_{1w} denotes the current window containing X_{1i} data points. The same logic applies to the sliding window difference of the mean, except the mean of X_{1w} is subtracted from each value of X_1 instead of the median.

7.2 N-Order Difference

This feature is particularly useful when there is linearity or autocorrelation in the data, and it helps in eliminating their effects. The first-order difference is obtained by subtracting the value of the feature lagged by one index. In other words, first-order difference is:

$$X'_{1i} = X_{1i} - X_{1(i-1)}.$$

The value of N is chosen up to 5 for all features since initial-order differencing captures most of the important properties. For example, first-order differencing helps remove non-stationarity and reduces dependency on previous values (Box et al., 2015b). Furthermore, second-order differencing eliminates trends that are not captured by first-order differencing.

7.3 Upper and lower peak values

Determining amplitude can be a challenging part of time series data. This feature is designed to detect anomalies, including point extremum and long contextual anomalies with extreme values.

We have used the `scipy` library to estimate the amplitudes of several local peaks in the single-dimensional time series. `Scipy` determines these by identifying local maxima with the help of user-defined parameters such as `distance` and `height`. A good example is shown in Figure 7.1, where it can be seen that `scipy` is able to find all local maxima, which can then be used as features in time series. These parameters are used as a basis to estimate the available local peaks. The detection of upper local maxima, however, is highly dependent on the `distance` parameter. It essentially compares the past value, current value, and future value to confirm whether the current value is a peak.

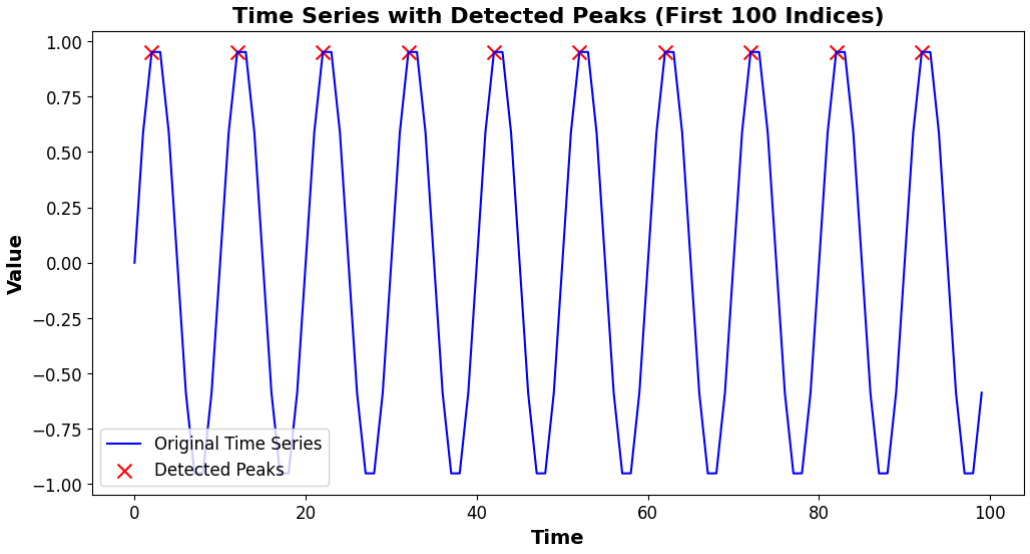


Figure 7.1: Illustration of the peak value extraction through maxima approximation.

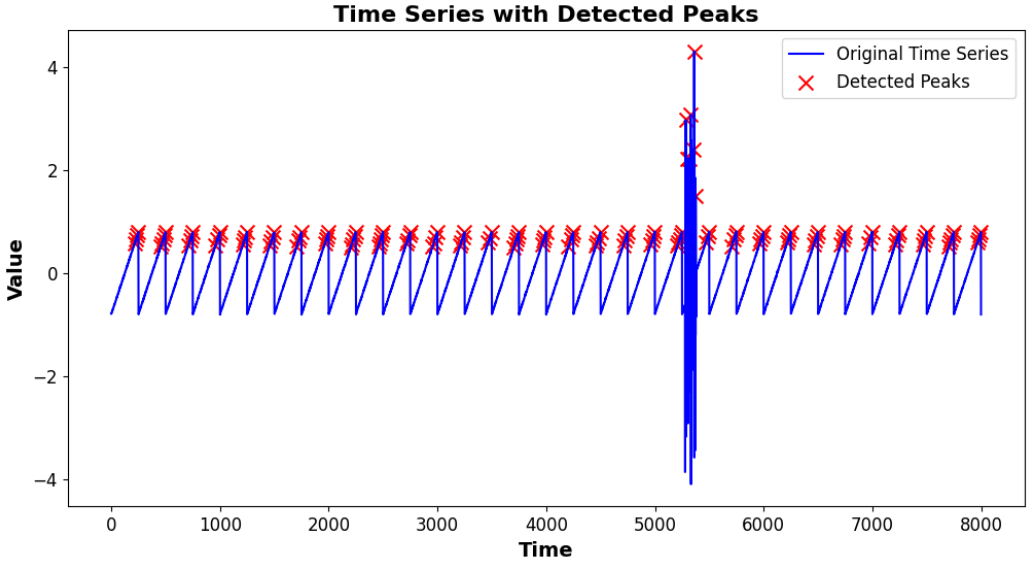


Figure 7.2: Illustration of the peak value extraction through maxima approximation alongwith anomalies in Sawtooth time series.

In Figure 7.2, local maxima with unexpectedly high values are also detected by `scipy`'s `find_peaks` function. This function can be used to create several other features, such as subtracting each value from its neighbouring local maxima.

Since a local maximum is a relative property that depends on surrounding points, the settings for each time series can vary. Here, the data is treated as a signal, which helps in generating more diverse features.

Similarly, local minima are usually present in a dataset due to oscillating behaviour of most time series. Thus, inverting the time series and multiplying it by -1 also helps identify the local minima. These two features are somewhat window-based, meaning their values remain the same until a new peak is found. Interestingly, another feature can also be created by taking the difference between each data point and the upper peak, and then adding

the same with the lower peak. This may help in detecting anomalies with sudden low or high values.

7.4 Fourier Transform

Spectrum analysis of a typical time series can detect several different types of anomalies, including those related to frequency or platform. Creating the DFT as a feature could be helpful, allowing DOUST to learn from the frequency domain representation.

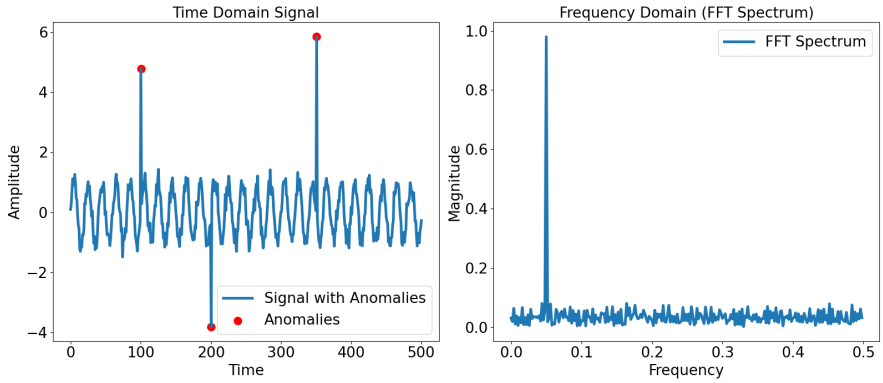


Figure 7.3: Discrete Fourier Transform of a simulated time series with anomalies.

The continuous Fourier expansion, $X(t)$, for a time series $x(t)$, can be expressed as:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t}\, dt,$$

where the angular frequency $\omega = 2\pi f t$, f is the frequency, and $t = 1, ..., T$. Since time is finite in our case, i.e., it starts from 0 and ends at T,the above equation can be rewritten as (Li, 2022):

$$X(\omega) = \int_0^T x(t)e^{-i\omega t}\, dt.$$

However, we usually don't have a continuous time series; therefore, the discrete Fourier transformation is considered. The DFT of $x(t)$ can be defined as (Li, 2022):

$$X_m = \sum_{k=1}^{M} x(t_k)e^{-i\omega_0 t_k},$$

where $\omega_0 = 2\pi m f_0$, the fundamental frequency $f_0 = 1/T$, and $m = 0, 1, ..., M-1$. Assuming a uniform interval Δt, such that $t_k = (k-1)\Delta t$, the frequency f can also be calculated as $f = mf_0$ (Li, 2022). The base function B is $e^{-i\omega_0 t_k}$, which has two parts: real and imaginary.

These can further be interpreted as the cosine (real) and sine (imaginary) components of complex numbers. Amplitude and phase shifts can be measured from this DFT. In this thesis, the amplitude of the DFT is used as a feature. Although phase angle information might also be useful, it is not used here. In Figure 7.3, a sine wave with a normally distributed noise and contamination can be seen on the left, and its Fourier transformation of amplitude is shown on the right. It can also be observed that at a certain frequency, i.e., approximately 0.05, the magnitude of the DFT is high.

7.5 Trend and seasonality

Consider a time series variable $y_t = (y_1, y_2, ..., y_n)$, where $n = 1$ to T, and T is the total time period. Based on the inclusion of trend and seasonality in the time series, it can be modeled in two ways: additive and multiplicative.

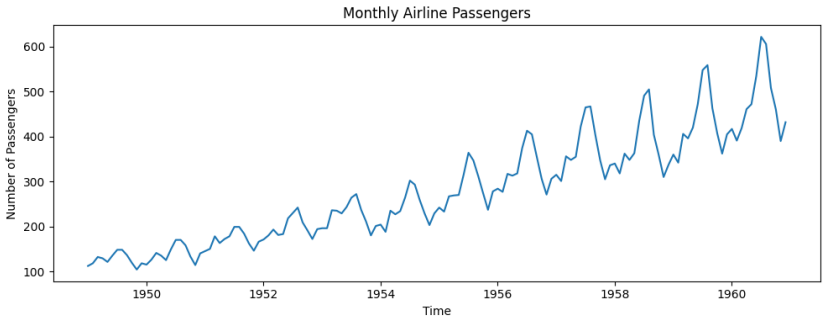


Figure 7.4: Flight passengers dataset from statsmodels.

Additive model: In this, y_t can be represented as a linear addition of the *trend*, *seasonality*, and *residual* (or noise) components (Hyndman and Athanasopoulos, 2018). It can be expressed as:

$$y_t = A_t + S_t + R_t.$$

A_t is the trend component, S_t is the seasonal component, and R_t is the residual. The trend component A_t can be increasing, decreasing, or following some other trend. For instance, in the Figure 7.4, the number of commercial flight passengers has been increasing each decade or every year due to technological advancements and a fast-paced lifestyle.

The seasonal component captures specific patterns at certain frequencies—for example, the number of passengers is typically high during the end of December due to Christmas and New Year. The frequency of the seasonal component can be estimated by plotting the time series. It can take values such as days, months, quarters, and so on. The remaining part in y_t is called the *residual* component, which usually consists of random data or noise, and typically it's modeling cannot be easily done.

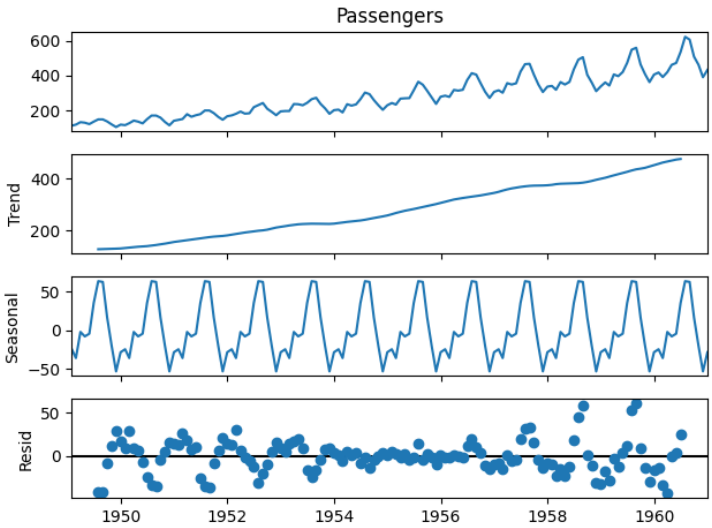


Figure 7.5: Time series decomposition of flight passengers data assuming it as an additive model.

Multiplicative model: In this case, y_t can be modeled as the product of the trend, seasonal, and residual components (Hyndman and Athanasopoulos, 2018). Mathematically, it is represented as:

$$y_t = A_t \cdot S_t \cdot R_t.$$

This model can be transformed into an additive model by taking the logarithm of y_t, i.e.,

$$log(y_t) = log(A_t) + log(S_t) + log(R_t).$$

The time series decomposition can then be performed similarly as in the additive model. In time series, trend and seasonality are often present in the data; hence, new features can be created based on these properties.

For instance, a classic example of time series decomposition is the dataset of passengers taking flights over a period of 11 years, from 1949 to 1960.

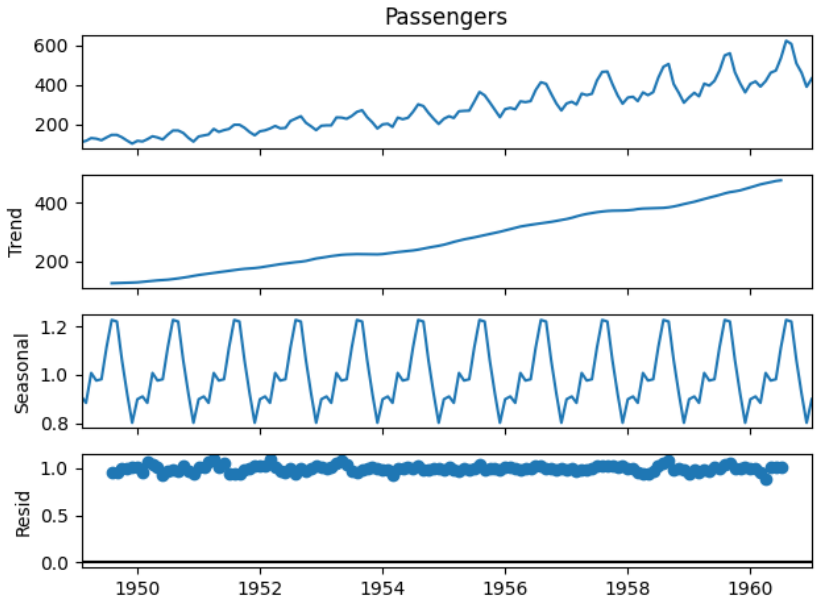


Figure 7.6: Time series decomposition of flight passengers data assuming it as an additive model.

This dataset consists of the number of passengers taking flights every month. It can be observed from Figure 7.5 that the trend component is increasing almost linearly. The frequency of the seasonal component in this time series is *year*, which can be extracted by assuming the model type. The TSD assuming an additive model is shown in Figure 7.5, while the multiplicative model can be observed in Figure 7.6. The type of model can be assumed by observing the seasonal effect, if it remains constant over a long period, the additive model is preferred. In this specific example, the multiplicative model is better suited as the seasonal effect appears to increase over time.

Chapter 8

Experiments: Losses + Feature Extraction (simulated data)

This chapter discusses and analyses some of the good loss functions alongwith several features extracted from the time series data and then, compared with traditional unsupervised algorithms. Further, these versions of DOUST will be tested on complex benchmark datasets to test the performance of the proposed solution. All the methods are implemented using Python, tensorflow (Abadi et al., 2015) and keras (Chollet et al., 2015) frameworks and using all the libraries such as scipy (Virtanen and Gommers, 2020), pandas(Reback et al., 2020) and numpy (Harris et al., 2020). For visualizations of several graphs, seaborn (Waskom, 2021) and matplotlib (Hunter, 2007) are used in Python.

8.1 Data preprocessing

The GUTENTAG dataset contains 193 datasets. DOUST is applied to time series with some custom loss functions and features extracted from each di-

mension of a dataset. For each dimension, 18 features are extracted, which is quite a lot, especially as the number of dimensions increases, the total number of features becomes extremely large. Therefore, bagging was introduced with a value of 0.5, i.e., only 50% of the features are used for training. This may reduce the performance of the models, as indicated by the authors of DOUST, who noted that bagging generally hurts the model's performance. However, when all features were used earlier, the submodels took significantly longer to run and included too much irrelevant information.

Error Handling: There were only very few errors while running our methodology on GUTENTAG benchmark datasets. All the failures on each dataset are handled with a re-try on the same data and parameters. This leads to no errors at all on any dataset.

8.2 Analysis of the results

Table 8.1: AUC-ROC Scores of DOUST versions, Isolation forest and KNN on GUTENTAG dataset.

Version	**Mean AUC-ROC**	**# errors**
DOUST_L2_10_25_50_75_90	0.527036	0
DOUST_L1_without_median	0.514855	0
DOUST_wasserstein	**0.586536**	0
IForest	0.5064715	0
KNN	0.571577	0

The Table 8.1 shows the average AUC-ROC scores of some of the custom loss functions along with features extracted from each time series in GUTENTAG. There are no errors (# errors) in any of the DOUST versions and the same holds for traditional clustering algorithms. It can be observed that `DOUST_wasserstein` performs the best overall, surpassing the mean score of KNN by just a few percentage. From Figure 8.1, it can be seen that the loss

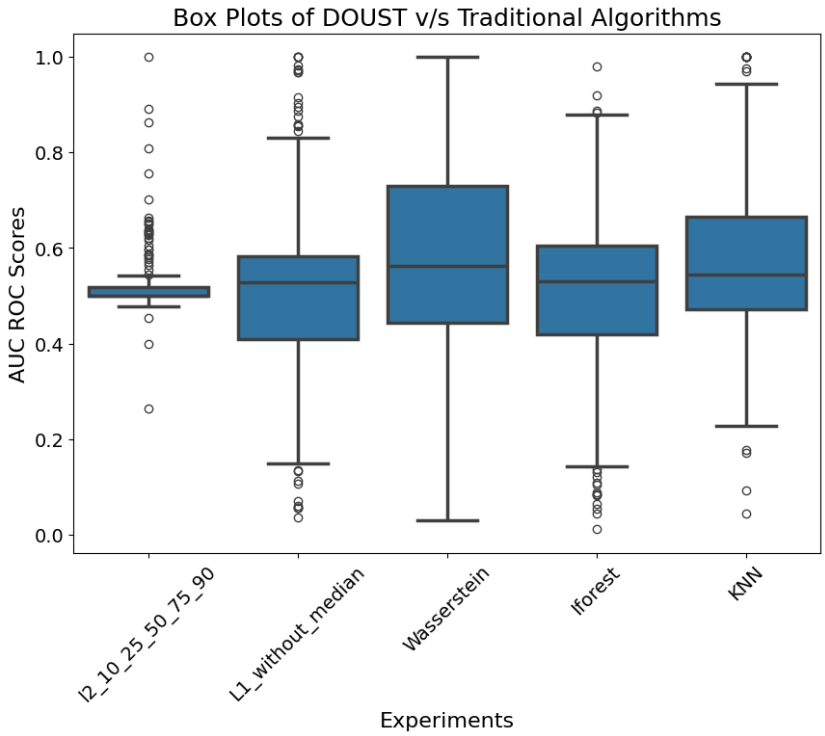


Figure 8.1: Box plots of AUC-ROC scores of DOUST with different versions and traditional algorithms such as Iforest and KNN on benchmark GUTENTAG.

function `DOUST_L1_without_median` seems to have a distribution of scores similar to IForest and KNN. On the other hand, `DOUST_L2_10_25_50_75_90` has more confined values around 0.5 for 50% of the datasets in GUTENTAG. In contrast, `DOUST_L2_10_25_50_75_90` has only a few poor performing datasets compared to the rest of the experiments. The DOUST variant with the loss function `L1_without_median` exhibits diverse performance.

However, the performance of `DOUST_wasserstein` is better than the others. Its 50% AUC scores lie between 0.4 and 0.8, which clearly indicates fairly better performance as compared to other methods in the figure. Overall, it

can be concluded that `DOUST_wasserstein` performs very well in terms of AUC-ROC score.

8.3 Analysis of results by Base Oscillations

The performance of the developed methods is also analysed on the basis of "base oscillations". Therefore, the average AUC-ROC scores corresponding to the base oscillations present in the GUTENTAG datasets are presented in Table 8.2 and Table 8.3.

Table 8.2: Mean AUC-ROC values for cbf and ecg base oscillations.

Method	**cbf**	**ecg**
Iforest	0.527461	0.518129
KNN	**0.528251**	0.592411
DOUST_L1	0.480742	0.486576
DOUST_L2	0.504038	0.518027
DOUST_Wasser	0.445893	**0.650085**

For *Cylindrical Bell Funnel* and *Sine* base oscillations, KNN demonstrates better performance than all versions of DOUST. The performance of all DOUST variants is notably lower in the case of CBF when compared to the other methods. *DOUST_Wasserstein* shows competitive performance against KNN for the *Sine* base in the Table 8.3. Moreover, *DOUST_Wasserstein* performs better than both IForest and KNN for *Polynomial*, *Random Walk*, and *Electrocardiogram*. This highlights the potential of DOUST to outperform traditional algorithms in anomaly detection tasks.

Table 8.3: Mean AUC-ROC values for poly, rw, and sine base oscillations.

Method	poly	rw	sine
Iforest	0.534038	0.436927	0.509234
KNN	0.572860	0.536094	**0.611172**
DOUST_L1	0.430716	**0.682379**	0.506784
DOUST_L2	0.574595	0.556294	0.502153
DOUST_Wasser	**0.604475**	0.632844	0.596410

8.4 Analysis of results by number of dimensions

To further analyze the performance of the results, the number of dimensions can also be considered. Figure 8.2 shows the average AUC-ROC results against the number of dimensions present in GUTENTAG.

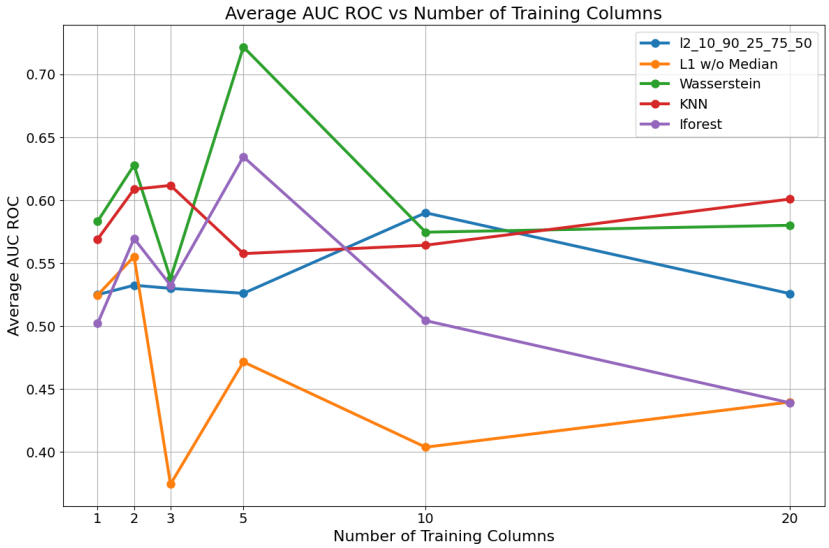


Figure 8.2: Line graph of average AUC-ROC scores on the basis of number of dimensions

The green line represents `DOUST_wasserstein`, which appears to exhibit the best performance among the other methods for 1, 2, and 5-dimensional

datasets. Although its performance slightly dips for 10 and 20 dimensions, it remains a close second, showcasing strong overall robustness. *KNN* (red line) is another reliable performer, maintaining stable and competitive scores across all dimensions, making it a strong alternative to *Wasserstein*.

The `DOUST_L2_10_25_50_75_90` method (blue line) delivers average results overall, with a noticeable spike at 10 dimensions where it temporarily leads. However, its performance at other dimensions remains relatively moderate as compared to `DOUST_L1_without_median`, which performs poor among other variants.

8.5 Conclusion

To summarize the results from application of several DOUST versions to the GUTENTAG benchmark datasets, it can be concluded that the performance of all DOUST versions was diverse. Overall, the results of the `DOUST_wasserstein` version were, by far, the best across most instances, demonstrating its strong ability to perform well on a diverse dataset.

Chapter 9

Experiments: Losses + Feature Extraction (real-world)

This chapter presents the analysis, results, and final assessment of the experiments conducted in this thesis using real datasets. While we have already evaluated the performance of several versions on various simulated datasets, it is now time to explore the performance of DOUST's versions on a real dataset.

9.1 Data preprocessing

For the NASA SMAP and MSL datasets, Kaggle is the only public source available. The data files were initially in `.npy` format, so each file was converted to `.csv`. Since some datasets contained missing values, these were filled with the median values of their respective columns. This step helps preserve the time series behavior, as removing the missing values could lead the model to misinterpret them as anomalies. Afterward, the feature extrac-

tion process is applied to the data in the same manner as with the simulated datasets, allowing us to extract relevant knowledge from the time series.

Error Handling: All the failures on each dataset is handled with a re-try once more on the same data and parameters. There were many errors while running our methodology, especially NASA-MSL datasets. Each anomaly score for a failure leave a missing value for that dataset, this value is dropped from the list. Therefore, the average score will be calculated only for the successful run.

9.2 Analysis of the findings

This section analyses the performance of DOUST versions and a state of the art model, namely, Telemanom, on NASA SMAP and MSL datasets.

Table 9.1: AUC-ROC Scores of DOUST versions and Telemanom on NASA SMAP dataset.

Version	**Mean AUC-ROC**	**# errors**
DOUST_L2_10_25_50_75_90	0.505860	18
DOUST_L1_without_median	0.61741024	7
DOUST_wasserstein	0.570528	3
Telemanom	**0.679620**	0

Table 9.1 shows the average AUC-ROC scores of 3 versions of DOUST, and a state-of-the-art method, Telemanom. `DOUST_L1_without_median` is performing well among other two variants, with the failure to run on 7 datasets. It seems that the loss function `L2_10_25_50_75_90` is buggy, the reason of failure of the variant is the inability to converge. Telemanom depicts the best performance on SMAP as compared to all DOUST variations, with zero errors on the datasets.

Figure A.5(a) shows the box plots of anomaly score for 3 variations of DOUST and Telemanom. `DOUST_wasserstein` have only a few low perform-

ing datasets, but that did't help in beating the average score of Telemanom. It can be also be observed that scores of all the DOUST versions are below the median score of Telemanom, which shows inability of DOUST to perform better in NASA SMAP.

Table 9.2: AUC-ROC Scores of DOUST versions and Telemanom on NASA MSL dataset.

Version	**Mean AUC-ROC**	**# errors**
DOUST_L2_10_25_50_75_90	0.5222486	17
DOUST_L1_without_median	0.470912	15
DOUST_wasserstein	0.572214	1
Telemanom	**0.708003**	1

Anomaly score for MSL datasets can be seen in Table 9.2. Here, only `DOUST_wasserstein` runs with only one failure to run on MSL, and two other versions of DOUST seem to be very buggy, as almost half of the MSL was an unsuccessful run. Once again, Telemanom outperformed every DOUST version. The loss functions were not able to converge in time for many datasets and that caused their average performance to be really bad. `DOUST_wasserstein` seems to be performing better than other two versions of DOUST, but unable to beat the average AUC-ROC score of Telemanom.

Figure A.5(b) shows the box plots of anomaly score of all the methods used to test on NASA MSL. Due to high number of errors in the two versions: `DOUST_L2_10_25_50_75_90` and `DOUST_L1_without_median`, the mean score is 0.5. `DOUST_wasserstein` also didn't perform very well. The reason might be the fact that DOUST needs a bigger dataset and here, the MSL datasets are really small. Telemanom has a diverse range of AUC-ROC scores, and its median is around 0.9, dominating the performance in NASA-MSL datasets.

9.3 Conclusion

To summarize the application of several versions of DOUST's on a real-world dataset, the DOUST's versions were able to beat the random guess performance, but not the state of the art Telemanom's average AUC-ROC score in both NASA SMAP and SMAP datasets. In case of MSL, two of the DOUST's versions failed on almost half of the datasets except `DOUST_wasserstein`, leading to an overall bad average score on MSL datasets. Overall, the performance of DOUST versions was not good enough on NASA SMAP and MSL datasets to beat the state of the art method, namely, Telemanom.

Chapter 10

Summary

The aim of this thesis was to study the adaptation of DOUST to time series data, and transfer the good performance ability of this algorithm. DOUST claims to be having performance similar to a supervised algorithm, which was tested and developed on Tabular datasets.

Initially, a DOUST with various features extracted were executed on manually simulated complex datasets (D1 and D2), which showed a fairly average AUC-ROC score on D1 and a bad result on D2. After carefully examining the box plots and line plots of the anomaly score for test and train set, we found that the distribution of abnormal and normal sets was similar, which shouldn't happen. Because DOUST works on maximizing the difference between distribution of abnormal and normal data. Then, we thought about transforming the time series data to tabular data by random shuffling the timestamps and re-applying DOUST without any feature extraction. The results were flabbergasting, since it really helped DOUST to score 0.9 AUC-ROC on both datasets. And it was confirmed that shuffling helps DOUST's performance by comparing anomaly plots with and without random shuffling.

Since there was a foundational problem in DOUST's loss function when applied on sequential data, we tried several modifications of loss functions to

minimize the effect of temporal paradox, or benefit from it. A total of 17 loss functions were carefully designed to study the behaviour, hence, 17 DOUST versions were created covering different viewpoints of temporal data and were eventually applied on two simple toy examples, E1 and E2. It was found that `l2_10_25_50_75_90`, `l1_without_median` and `wasserstein` were performing well on both datasets.

We also extracted a general set of diverse features that try to extract a wide range of information from time series data. For each dimension in a time series, a total of 18 features were extracted, including sliding window features, upto 5th-order differencing, fourier transform, time series decomposition, and more. This was done for DOUST to have enough information and extra knowledge obtained from temporal data.

Only 3 loss functions were selected, that showed potential to solve the Temporal paradox. After applying 3 versions of DOUST with feature extraction on 193 simulated benchmark datasets from GUTENTAG, we found that `DOUST_wasserstein` worked well and showed competitive edge, showed an average AUC-ROC score of 0.5865. This version outperformed KNN by a small margin of 1.5%. While analyzing results on the basis of "base oscillation" of GUTENTAG, it was found that `DOUST_wasserstein` performed relatively better in case of `ecg` and `polynomial`, whereas `DOUST_L1_without_median` performed good in case of `random walk`. For `sine` and `cbf`, no DOUST version could outperform KNN.

Further, the results were also analyzed on the basis of number of dimensions in GUTENTAG, it was found that `DOUST_wasserstein` works relatively better for most of the dimensions except when number dimensions are 3, whereas `DOUST_l1_without_median` seems to be performing bad.

Now, DOUST was also applied to real-world datasets of NASA SMAP and MSL and further analyzed against the state of the art method, Telemanom. It was found that for NASA SMAP datasets, DOUST versions were unable

to beat the Telemanom. Although, `DOUST_L1_without_median` showed the best performance among other variants. In NASA MSL datasets, all the versions were left behind Telemanom in terms of mean AUC-ROC score and it was found that two versions of DOUST were really buggy and couldn't finish running for half of the datasets in MSL, except `DOUST_wasserstein`. The possible reason of the failure of DOUST version when applied to NASA MSL datasets might be a low number of observations in the dataset.

In future, the loss functions of DOUST can be further researched and experimented to find the solution to concept drift in time series. There are several types of concept drift which can be countered just by modifying the loss function or even re-training of the trained model in order to adjust it's weights. Further, model's interpretability can be added to see if some features are helping or not (Kluettermann et al., 2023). Since there are several ways to do it, creating Shapely values for each feature may help showing what helps the model negatively or positively in terms of metrics. Attention based methods can also be explored and researched to learn more from the temporal data (Vaswani et al., 2017).

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. `https://www.tensorflow.org/`. Software available from tensorflow.org.

Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. 2016. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems* 55 (2016), 278–288.

Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE* 2, 1 (2015), 1–18.

Jessa Bekker and Jesse Davis. 2020. Learning from positive and unlabeled data: A survey. *Machine Learning* 109, 4 (2020), 719–760.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015a. *Time series analysis: forecasting and control*. John Wiley & Sons.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015b. *Time series analysis: forecasting and control*. John Wiley & Sons.

Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. ACM, 93–104. `https://doi.org/10.1145/335191.335388`

François Chollet et al. 2015. Keras. https://github.com/keras-team/keras. Accessed: 2025-04-19.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

Rahul Dey and Fathi M Salem. 2017. Gate-variants of gated recurrent unit (GRU) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 1597–1600.

Yadolah Dodge. 2008. *The concise encyclopedia of statistics*. Springer Science & Business Media.

Tom Fawcett. 2006. Introduction to ROC analysis. *Pattern Recognition Letters* 27 (06 2006), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Everette S Gardner Jr. 1985. Exponential smoothing: The state of the art. *Journal of forecasting* 4, 1 (1985), 1–28.

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. 2020. Array programming with NumPy. *Nature* 585, 7825 (2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting spacecraft anomalies using lstms

and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 387–395.

John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

Rob J. Hyndman and George Athanasopoulos. 2018. *Forecasting: Principles and Practice* (2nd ed.). OTexts, Melbourne, Australia. https://otexts.com/fpp2 Accessed on April 18, 2025.

Pooja Kamat and Rekha Sugandhi. 2020. Anomaly detection for predictive maintenance in industry 4.0-A survey. In *E3S web of conferences*, Vol. 170. EDP Sciences, 02007.

Shehroz S Khan and Michael G Madden. 2014. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review* 29, 3 (2014), 345–374.

Simon Kluettermann, Chiara Balestra, and Emmanuel Müller. 2023. On the Efficient Explanation of Outlier Detection Ensembles through Shapley Values. In *Proceedings of the ECML PKDD 2023: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. https://psorus.github.io/papers/deanshap.pdf

S. Klüttermann and E. Müller. 2022. DEAN: Deep Ensemble Anomaly Detection. https://github.com/KDD-OpenSource/DEAN/. Accessed: 2025-04-18.

S. Klüttermann and E. Müller. 2024. About Test-time training for outlier detection. https://arxiv.org/abs/2404.03495. Accessed: 2025-04-18.

Mario Köppen. 2000. The curse of dimensionality. In *5th online world conference on soft computing in industrial applications (WSC5)*, Vol. 1. 4–8.

Chunyan Li. 2022. *Time Series Data Analysis in Oceanography*. Cambridge University Press, Cambridge, i–ii.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. 413–422. https://doi.org/10.1109/ICDM.2008.17

Yuejiang Liu, Parth Kothari, Bastien Van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. 2021. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems* 34 (2021), 21808–21820.

Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489

Peter S Maybeck. 1982. *Stochastic models, estimation, and control*. Vol. 3. Academic press.

Larry Medsker and Lakhmi C Jain. 1999. *Recurrent neural networks: design and applications*. CRC press.

Keiron O'shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (Dallas, Texas, USA) *(SIGMOD '00)*. Association for Computing Machinery, New York, NY, USA, 427–438. https://doi.org/10.1145/342009.335437

Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, Simon Hawkins, Gfyoung, Sinhrks, Alexander Klein, Michael Roeschke, et al. 2020. pandas-dev/pandas: Pandas. *Zenodo* (2020). https://doi.org/10.5281/zenodo.3509134

Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. 2018. Deep One-Class Classification. In *International conference on machine learning*. PMLR, 4393–4402.

Yasin Sahin and Emre Duman. 2011. Detecting credit card fraud by decision trees and support vector machines. *Proceedings of the International MultiConference of Engineers and Computer Scientists* 1 (2011), 442–447.

Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13, 7 (2001), 1443–1471.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. 2020. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*. PMLR, 9229–9248.

Jian Tang, Zhexue He, and Shengchun Deng. 2002. Enhancing effectiveness of outlier detections for low density patterns. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD)*. Springer, 535–548. Available at: `https://www.cs.cmu.edu/~christos/courses/826.S03/papers/cof.pdf`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

Cédric Villani. 2009. *The Wasserstein distances*. Springer Berlin Heidelberg, Berlin, Heidelberg, 93–111. `https://doi.org/10.1007/978-3-540-71050-9_6`

Pauli Virtanen and et. al Gommers. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

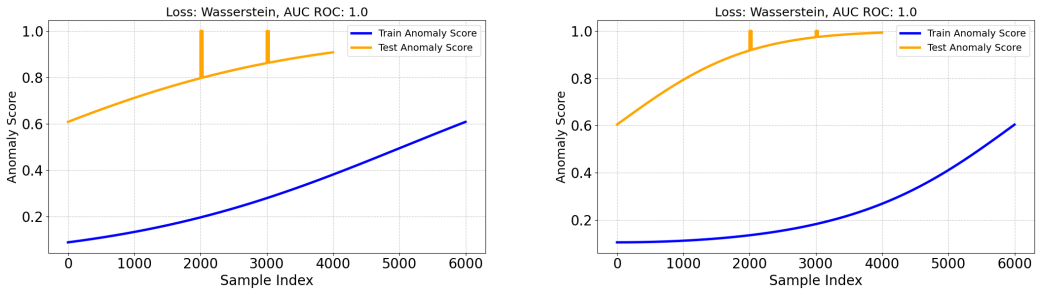
Michael L. Waskom. 2021. Seaborn: statistical data visualization. *Journal of Open Source Software* 6, 60 (2021), 3021. https://doi.org/10.21105/joss.03021

Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. 2022. TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms. *Proceedings of the VLDB Endowment (PVLDB)* 15, 12 (2022), 3678–3681. https://doi.org/10.14778/3554821.3554873

Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 665–674.

Appendix A

Appendix Figures



(a) E1 dataset (b) E2 dataset

Figure A.1: Visualization of anomaly scores using wasserstein loss on toy datasets E1 and E2.

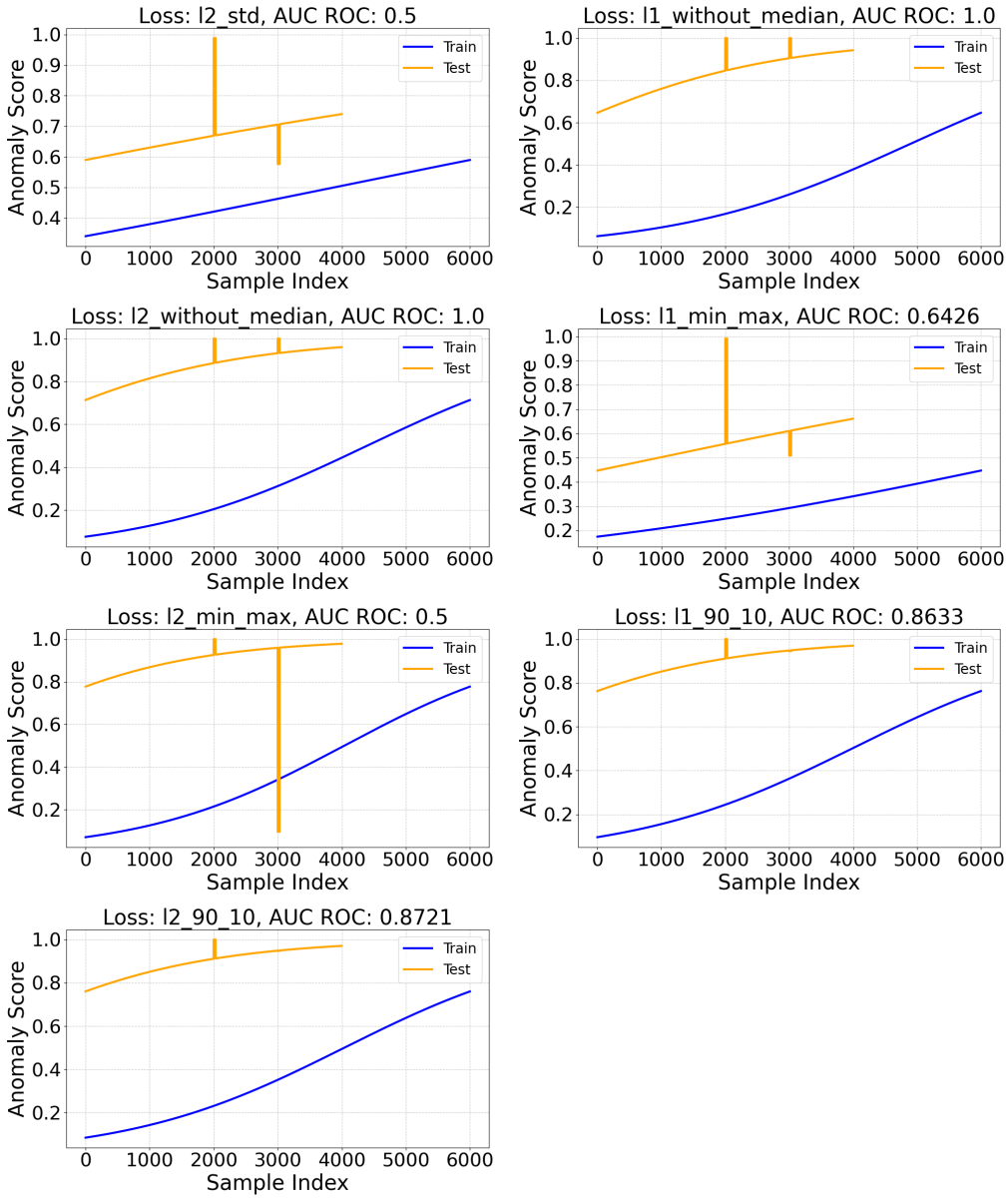


Figure A.2: Anomaly scores of seven loss functions on E1 dataset (Part 1).

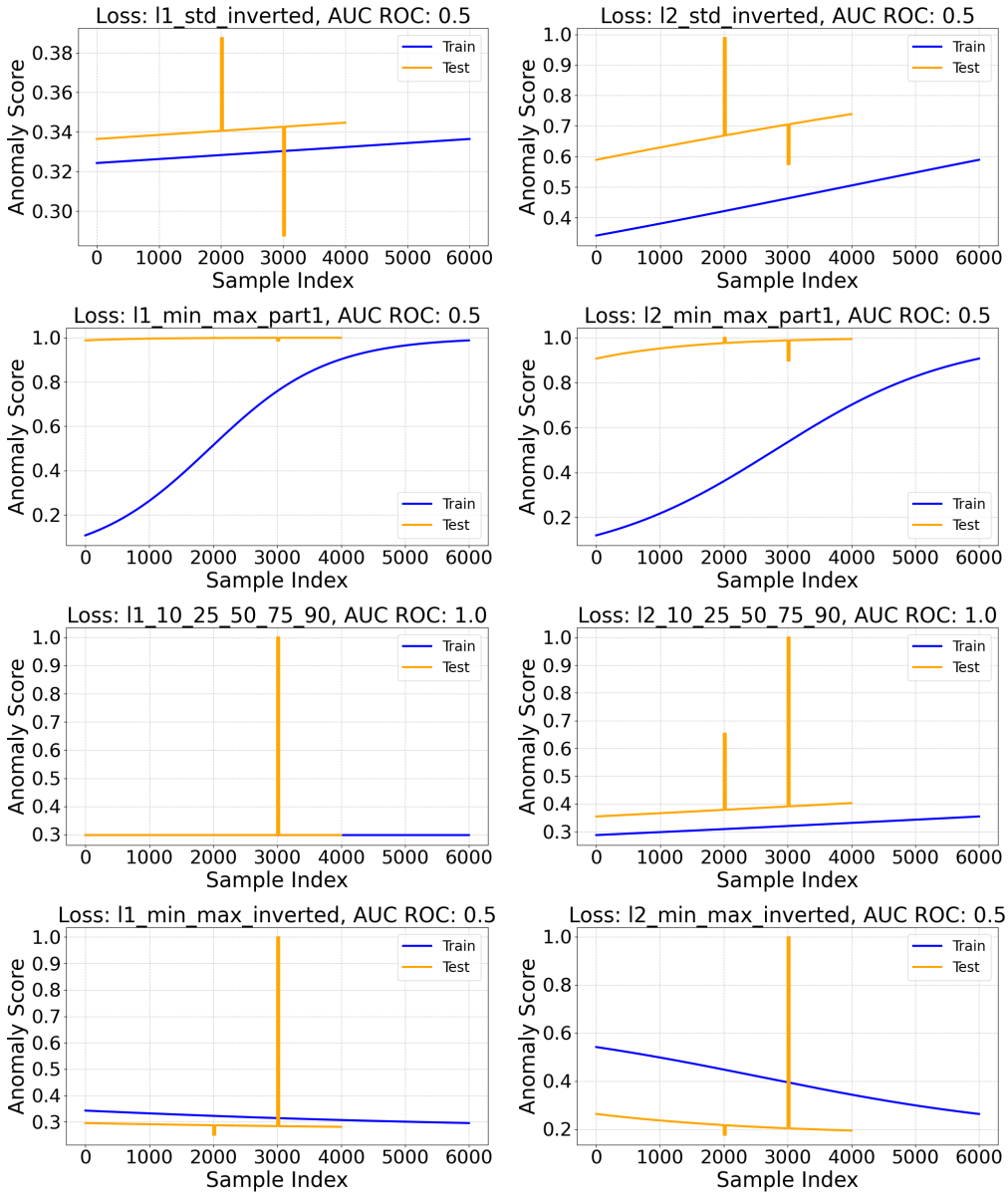


Figure A.3: Anomaly scores of eight loss functions on E1 dataset (Part 2).

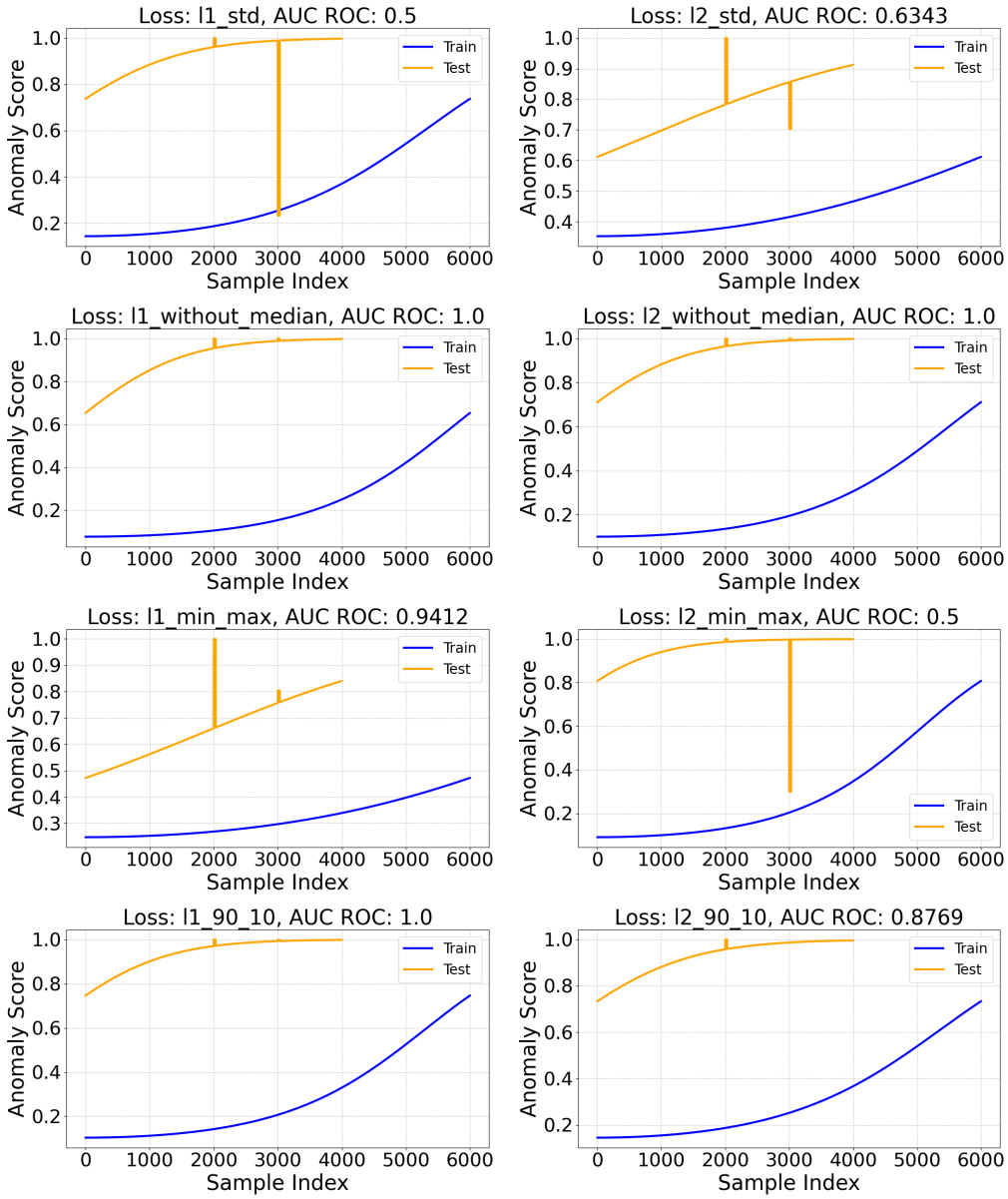
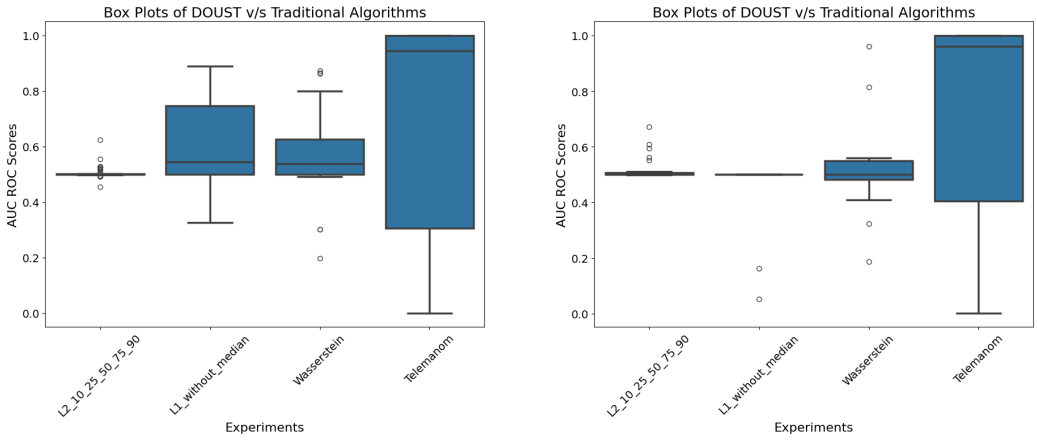


Figure A.4: Anomaly scores of eight loss functions on E2 dataset (Part 1).



(a) SMAP dataset

(b) MSL dataset

Figure A.5: Visualization of anomaly scores of DOUST versions and Telemanom on NASA SMAP and MSL.

Eidesstattliche Versicherung

(Affidavit)

Kumar Vikas

Name, Vorname
(surname, first name)

231578

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Test-time training for Anomaly Detection in time Series

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund 20.04.2025

Ort, Datum
(place, date)

Vikas.

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund 20.04.2025

Ort, Datum
(place, date)

Vikas.

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**