Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

DOCTORAL THESIS

Mathematical Modeling of Anomalies in Large-Scale Vector and Structural Data

Prague 2023 Ing. Martin Flusser

Bibliografický záznam

Autor:	Ing. Martin Flusser České vysoké učení technické v Praze Fakulta jaderná a fyzikálně inženýrská Katedra softwarového inženýrství
Název práce:	Matematické modelování anomálií ve vektorových a strukturovaných datech velkého rozsahu
Studijní program:	Aplikace přírodních věd
Studijní obor:	Matematické inženýrství
Školitel:	RNDr. Petr Somol, Ph.D. 1) Ústav teorie informace a automatizace Akademie věd České republiky 2) Gen Digital
Školitel specialista:	Ing. Vladimír Jarý, Ph.D. České vysoké učení technické v Praze Fakulta jaderná a fyzikálně inženýrská
Akademický rok:	2022/2023
Počet stran:	159
Klíčová slova:	Detekce anomálií, neuronová síť, online učení, detekce anomálií pro multiple instance learning

Bibliographic Entry

Author:	Ing. Martin Flusser
	Czech Technical University in Prague Faculty of Nuclear Sciences and Physical Engineering Department of Software Engineering
Title:	Mathematical Modeling of Anomalies in Large-Scale Vector and Structural Data
Degree Programme:	Application of Natural Sciences
Field of Study:	Mathematical Engineering
Supervisor:	RNDr. Petr Somol, Ph.D.
	1) Institute of Information Theory and Automation Czech Academy of Sciences
	2) Gen Digital
Supervisor specialist:	Ing. Vladimír Jarý, Ph.D.
	Czech Technical University in Prague Faculty of Nuclear Sciences and Physical Engineering
Academic year:	2022/2023
Number of Pages:	159
Key words:	Anomaly detection, Neural network, Online learning, Multiple instance anomaly detection

Declaration

I hereby declare I have written this doctoral thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

Prague, May 2023

.....
Ing. Martin Flusser

Abstrakt

Detekci anomálií lze považovat za otevřený problém, a to i přes rostoucí množství známých metod. Použitelnost různých anomálních detektorů se může lišit v závislosti na oblasti použití a okolností aplikace. Zejména v průmyslovém prostředí s velkým objemem dat je rozhodujícím faktorem rychlost inference, která může učinit i velmi přesný detektor anomálií nepoužitelným.

V této práci navrhujeme zmíněný problém řešit trénováním zástupné neuronové sítě založené na pomocné trénovací množině aproximující výstup zdrojového anomálního detektoru.

Nejprve ukážeme, že stávající detektory anomálií lze aproximovat s vysokou přesností při dosažení inferenční rychlosti, která umožňuje reálnou aplikaci. Koncept ověřujeme na 64 problémech založených na veřejně dostupných srovnávacích datových sadách a poté metodu vyhodnocujeme v kontextu kybernetické bezpečnosti, kde porovnáváme náš přístup s řadou nejmodernějších algoritmů: one-class k-nearest-neighbors, local outlier factor, isolation forest, auto-encoder a dva typy generativních adversariálních sítí. Naše výsledky ukazují, že nově navrhovaný přístup může úspěšně zastoupit nejpřesnější, ale neúměrně pomalý model. Navíc pozorujeme, že zástupná neuronová síť může dokonce zlepšit přesnost zdrojového modelu.

Dále se zabýváme problematikou aktualizace modelu s ohledem na nepřetržitý tok velkých objemů příchozích dat. Navrhujeme efektivní metodiku pro aktualizaci modelu. Demonstrujeme, že výsledný online model má srovnatelnou přesnost s offline modelem a zároveň je schopen překonat konkurenci mezi online modely.

Nakonec, na základě skutečnosti, že mnoho klíčových problémů v kybernetické bezpečnosti a dalších odvětvích často závisí na strojově generovaných strukturálních datech, jako jsou logy spouštění aplikací nebo telemetrie síťové komunikace, přejímáme navrženou metodu pro použití v prostředí učení na více instancích (multiple instance learning). Na rozdíl od dosavadních metod přináší navrhované řešení hned dvě zásadní výhody: zabraňuje ztrátě informací (vyhýbá se nutnosti explicitní transformace strukturních vzorků do vektorové podoby) a poskytuje zásadně vyšší rychlost inference než specializované detektory strukturních anomálií. Evaluace na několika veřejně dostupných strukturálních datových sadách ukazuje, že navrhovaná metoda je schopna překonat dosavadní metody v přesnosti, a potvrzuje zásadní výhodu rychlosti inference.

Abstract

Anomaly Detection can be viewed as an open problem, despite the growing plethora of known anomaly detection techniques. The applicability of various anomaly detectors can vary depending on the application area and problem settings. Especially in the Big Data industrial setting, inference speed is a crucial factor that may render even a highly accurate anomaly detector useless.

In this work, we propose to address this problem by training a surrogate neural network based on an auxiliary training set approximating the source anomaly detector output.

First, we show that existing anomaly detectors can be approximated with high accuracy and with application-enabling inference speed. We validate the concept on 64 problems based on public benchmark data sets and then we evaluate the method in the context of cyber-security, where we compare our approach to a number of state-of-the-art algorithms: one-class k-nearest-neighbors, local outlier factor, isolation forest, auto-encoder, and two types of generative adversarial networks. Our results show that the proposed approach can successfully replace the most accurate but prohibitively slow model. Moreover, we observe that the surrogate neural network may even improve the source model accuracy.

Next, we address the problem of keeping the model up-to-date with respect to a continuous stream of large volumes of incoming data. We propose an efficient methodology for updating the model. We demonstrate that the resulting online model has comparable accuracy to the offline model while being capable of outperforming the competitors in this setting.

Finally, based on the fact that many crucial problems in cyber-security and other industries often depend on machine-generated structural data (like application execution logs or network communication telemetry), we adopt the proposed method for use in multiple-instance-learning setting. In contrast to prior art, our solution brings two crucial advantages at once: it prevents loss of information (it avoids the need for explicit transformation of structural samples into vector form) and it provides principally faster inference speed than specialized structural anomaly detectors. The evaluation on numerous publicly available structural data sets shows that the proposed method is capable of outperforming the state-of-the-art in accuracy, and confirms the principal inference speed advantage.

Acknowledgements

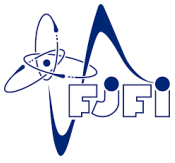
First of all, I would like to thank my parents for their enormous support and love throughout my life. Without their encouragement, I would not have been able to embark on this incredible educational journey culminating in my doctoral studies. I would also like to thank my love, Adéla, for her patience, support, and unconditional love. She has always been a constant source of energy, inspiration, and optimism, especially in the challenging times of the pandemic lockdowns.

I am grateful to my supervisor Petr Somol for guiding me through my doctoral studies, managing my scientific and professional development, and sharing his solid expertise. I also thank Petr for his patience and enthusiasm especially when he spent numerous evenings and weekends revising our research papers or the doctoral thesis itself.

Next, I would like to thank all the other people who participated, specifically the supervisor specialist Vladimír Jarý, and my colleagues at Cisco System Tomáš Pevný, Jan Brabec, and Martin Kopp. I also thank Marek Dědič for sharing his technical expertise with multiple instance learning neural library and helping me to solve implementation issues.

Further, I would like to thank Cisco Systems company (Prague) and the team for the opportunity to participate in applied research in the field of cyber-security and also for providing me with the data, infrastructure, and mainly time flexibility and support necessary for the research.

I would also like to express my sincere gratitude to the Czech Technical University in Prague (CTU) and the Faculty of Nuclear Sciences and Physical Engineering for the opportunity to take part in the doctoral study program. Furthermore, I thank the student grant competition of CTU for supporting my research with grants No. SGS20/188/OHK4/3T/14 and SGS23/190/OHK4/3T/14 and also to the supercomputer services of the computing and information center at CTU.



Contents

Introduction

Anomaly detection (AD) is gaining on importance with the massive increase of data we can observe in every domain of human activity. In many applications, the goal is to recognize objects or events with unclear definitions and missing prior ground truth, while the only assumed certainty is that these entities should be different from what we know well. The problem can thus be seen as the problem of modeling what is common and then identifying outliers. Anomaly detection is a crucial technique in cyber-security, industrial quality control, banking, credit card fraud detection, medical diagnostics, and many other fields [1].

Although AD as a general problem has been widely studied (cf. Sect. 1.5), the progress is arguably slower than in supervised learning. Particularly, the recent rapid advances in neural networks for classification (see, e.g., [2], [3]) seem harder to replicate in AD. The primary neural models used in AD are unsupervised generative models, typically auto-encoders (AE) or generative adversarial networks (GAN) [4]. Although there is great promise in GAN models [5], they can be more difficult to successfully apply [6] than traditional techniques. Traditional techniques thus often remain the straightforward choice, especially in industrial applications. Among traditional AD principles, density-based techniques like k-nearest neighbor (kNN) [7] [8], isolation forest [9] or local outlier factor [10] quite often achieve surprisingly good accuracy. At the same time, many such models can become computationally expensive or even prohibitive in an industrial setting.

Our ultimate goal is to design models well applicable to large-scale data modeling in the area of cyber-security. To solve this problem, we would need either to reduce the complexity of an existing AD without compromising its accuracy, or to approximate it by a different, cheaper, but comparably accurate model. Although indirectly related ideas exist (e.g. [11]), there seems to be a lack of solutions addressing this problem in the AD context. For that reason we proposed to address the problem using neural networks due to their efficient inference speed and their mature support in an industrial setting [12].

The next important challenge regarding the application in large-scale data and industrial settings is the ability to operate online and to update the model. There is a challenge of balancing accuracy with memory efficiency such that the model's memory requirements should not grow with constantly repeating update procedures and thus the model is sustainable for application purposes.

In many applications including cyber-security, the examined objects or events are essentially described with structural data rather than with vector data. For example,

machine-generated data in JSON format have such a structure where the schema is variable and consists of various data types. Such data have excellent explainable value, however, are difficult to utilize. The default approach is to define an explicit conversion to vector representations so that standard techniques can be used, however, this leads to loss of information by a forced intervention and therefore it is preferable to have methods operating on the structural data. Multiple instance learning (MIL) [13] is a special type of learning where the object is described typically with more vectors (instances) of the same dimension. AD techniques for MIL are very least researched and are very difficult to operate on large-scale due to their computational complexity, however, on the small-scale scientific level were found beneficial for e.g. cyber-security [14], steganalysis [15], text document comparison [16] or social media analysis [17]. The narrow prior-art is mainly limited by its computational complexity as most of them are based on kNN techniques that are incomparably slower than kNN for vector data because they cannot be supported with search structures and each pairwise comparison is more difficult.

Scientific Goals

In this thesis, we aim at applied research motivated by the field of computer security, where anomaly detection is an essential mechanism. Thus, our goals are to develop a reliable and robust anomaly detector that is inherently suitable for the industrial setting. The attributes of interest thus are the ability to process large-scale data, scalability, inference speed, limitations of deployment, etc. Most of these attributes are well fulfilled with neural models that are constantly gaining importance mainly due to their massive software and hardware support. Furthermore, neural networks are also relevant for applications in embedded systems, IoT, and mobile phones, which, have even been equipped with neural processing modules in the last few years.

We will try to find the answers to several following hypotheses, which can be the key to solving the crucial challenges towards efficient and reliable anomaly detection in the large-scale and industrial setup in cyber-security:

- **Hypothesis 1** - Since the neural models have not fully developed their potential in AD for cyber-security and the non-neural models often provide better accuracy than the neural models, the hypothesis to be tested is whether a novel neural paradigm for anomaly detection, which combines density-based models with neural models, can outperform existing neural models for applications in cyber-security.

- **Hypothesis 2** - In order to ensure compliance with the rigorous requirements imposed in actual deployment scenarios, the hypothesis to be tested is whether it is possible to implement a solution for updating the model to reflect the changes in the modeled environment that is adhere to strict performance criteria, namely achieving a high degree of model accuracy while maintaining computationally and memory-efficient operations.

- **Hypothesis 3** - Finally, as there is a gap in efficient AD for structural data that has a great fit to cyber-security problems, the hypothesis to be tested is whether there is an option for a more efficient and reliable solution resulting in a more comprehensive and accurate detection. Additionally, as anomaly detection on structural data is still in its early stages of development, proposing a new solution may push the boundaries of the field.

Thesis Structure

The high-level introduction to AD is given in Chapt 1. We provide intuition into the anomalies based on data type and provide numerous examples. Next, we focus on the issue of thresholding and related evaluation metrics and we reason the metric selection used in this thesis. Further, we give an overview of the frequently utilized data sets and comment on their issues and we refer to the most advanced evaluation methodology based on utilizing numerous various data sets proposed by Emmott et al. Finally, we provide a comprehensive overview of the prior art in AD.

In Chapt. 2, we address hypothesis Nr. 1 and propose a novel neural paradigm to take use of the distance-based kNN principle to enable the training of neural models for AD with multiple potential advantages: low computational complexity leading to high detection speed as well as better robustness against noise. Particularly the performance is an important parameter, especially in online and embedded anomaly detection applications like network security.

The basic idea is simple: First, a set of generated auxiliary samples (AUX) is constructed with each sample labeled by its anomaly score as inferred by kNN. Second, a multilayer perceptron (MLP) neural network is trained from such auxiliary data.

In Chapt. 3, we address hypothesis Nr. 2 thus we further explore the potential of the proposed neural model and develop its capability of online operation which is typically crucial in cyber-security. We address the challenge of the model update procedure where the trade-off between memory efficiency and accuracy plays an important role (see Sect. 3.3.1). For example, it is basically impossible to successfully update the model without historical training data and thus it appears that the accuracy-optimal model should store all historical data. We propose to overcome the apparent trade-off by storing the distribution of the AUX set via Gaussian mixture models [18] instead of the raw training set. The advantage is that the AUX samples are not limited to a specific position in the vector space for each sample, but the objective of the AUX is to randomly fulfill the given distribution. The generated AUX is enriched with their anomaly scores by utilizing the trained neural model. As a result, the update procedure is extremely memory efficient and accurate.

Next, to address hypothesis Nr. 3, we design a framework for neural multiple instance anomaly detection (see Chapt. 4). Anomaly detection based on MIL pairwise distance definition and density-based techniques seems to be one of the most reliable techniques. Its computational complexity, however, is even more limiting for MIL due the to more demanding pairwise comparison, and thus the applicability of such

methods is mostly prohibitive. For this reason, we adapt the proposed neural AD model to MIL.

The adaption, however, is not straightforward. When designing the neural MIL AD framework, we have to address the problem of MIL NN design and also the challenge of creating MIL AUX set which is a generative task in the MIL (non-vector) space. This is the most challenging and crucial part of the algorithm and to solve it, we propose four various strategies and combine them together.

Finally, the research is summarized in the conclusion. We further discuss future opportunities and highlight the key contributions.

Chapter 1

Anomaly Detection

This chapter provides a brief and high-level introduction to AD and an explanation of related evaluation methodologies.

1.1 General Description

Anomaly detection is a sub-field of machine learning and it is related to outlier detection and novelty detection. The goal is to detect observations that are somehow different from expected patterns or from other observations we already know well, without knowing the exact definition of *different*. Hence, anomaly detection techniques focus on modeling what is expected and subsequently marking as an anomaly anything sufficiently different from the expected.

Contrary to other machine learning tasks such as classification, anomaly detection is more difficult because the character of the anomalous data is unknown when the model is trained. In addition, the decision on how much the sample must be different from others, to be detected as anomalous, is a problem. Thus most of the AD techniques instead of providing categorical decision (anomalous, normal) provide anomaly score that is more beneficial. If the binary decision is needed, it could simply be obtained with a threshold.

1.2 Understanding Anomality Based on Data Type

The understanding of anomaly is strongly depending on what category of data is utilized. Most commonly, anomaly detection is performed for:

1. Vector (tabular) data
2. Time series
3. Image data

This work is dedicated to anomaly detection on vector data, however, we provide a brief overview of anomalies for all three listed data types.

1.2.1 Anomalies in Vector Data

For data with a vector representation, the anomalies are relatively intuitive as demonstrated in a simple example in Fig. 1.1. Each observation represents a single credit card and the horizontal axis indicates average purchases in a week while the horizontal axes purchases in a specific and examined week. The data are expectably correlated due to the consistent behavior of the cardholders. And more importantly, the data are relatively compact which is demonstrated by the green soft boundary. The example also illustrates the application of one-class kNN and the resulting anomaly score. The anomaly score computation with one-class kNN algorithm ($k = 3$) is depicted for the possible fraud sample and one of the typical activity samples. The anomaly score is computed as a mean distance to the three nearest samples and based on the distance (compare the length of the red and green arrows), we can deduce the anomaly.

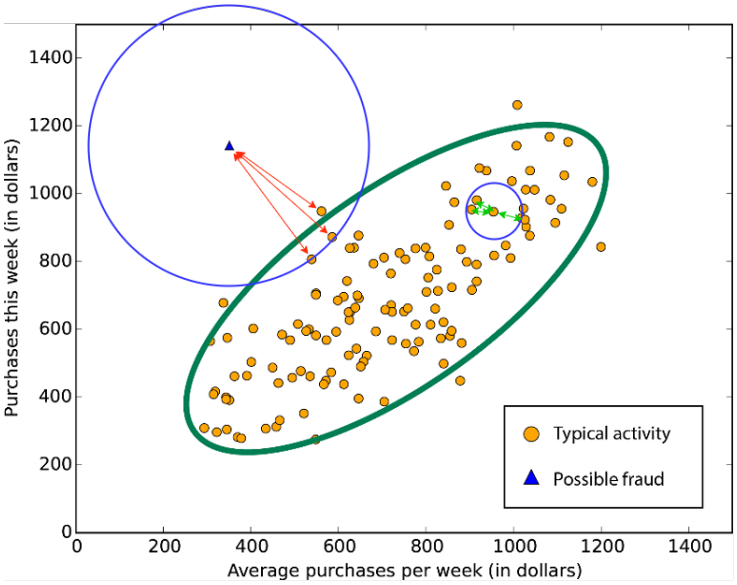


Figure 1.1: Example of a toy problem AD from the field of credit card fraud detection. Anomaly score computation is depicted for two selected samples via kNN ($k = 3$). (Credit: edited from original source: [19])

1.2.2 Anomalies in Time-series

In the time-series data, there are three basic types of anomalies:

1. Point anomaly
2. Contextual anomaly
3. Collective anomaly

Point anomalies (see Fig. 1.2) are similar to the anomalies discussed for the vector data. These are measures that different from all others regardless of the time context.

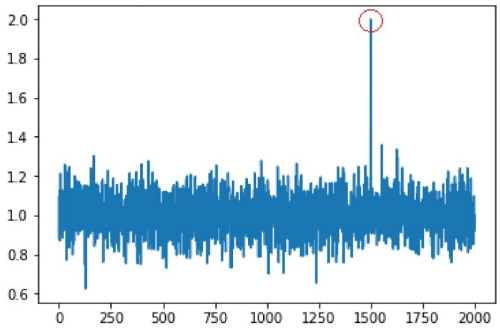


Figure 1.2: Point anomaly (circled in red) in time-series data generated with gaussian noise [20]

Contextual anomalies (see Fig. 1.3) are samples or measurements that might not be anomalous when simply compared to the entire data set. Here the time-series comes into the role and the sample is evaluated whether it fits the surrounding observations. In other words, whether the data fit the context.

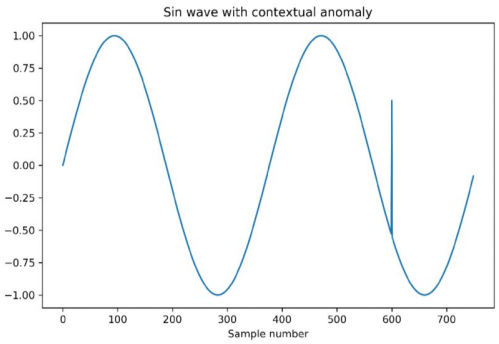


Figure 1.3: Contextual anomaly in time-series data [20] - the value at 600 does not exceed the range of the data, however, it is anomalous in the context.

Collective anomaly (see Fig. 1.4) is a group (collection) of data that does not fit the expected (high-level) pattern of the time-series. As shown in the example, the collective anomaly could consist of data that are even normal in the sense of point and contextual anomality.

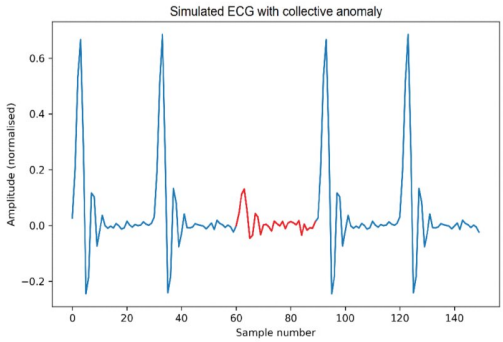


Figure 1.4: Collective anomaly in simulated ECG time-series data [20]

1.2.3 Anomalies in Image Data

In the image analysis, the anomalies might be images that belong to entirely different classes as demonstrated in Fig. 1.5 where the MNIST [21] data set consists of handwritten digits and the task is to recognize objects that are not handwritten. A similar task could also be to operate on pictures of faces and to detect fake faces or pictures without a face.

Figure 1.5: Images from MNIST as the normal data and 2 images from Fashion-MNIST as the anomaly data. [22]

Anomaly detection on image data is extensively applied to medical imaging where the anomalous objects are mostly characterized by localized structural differences from what is common in healthy patients as demonstrated in Fig. 1.6. A similar approach could be applied to manufacturing where camera-based data are used to detect defects in production.

Figure 1.6: Anomaly detection in medical imaging. The localized formation (circled in red) on the patient's chest makes the image anomalous. [23]

1.3 Thresholding and Evaluation

Most anomaly detectors provide anomaly scores rather than a binary decision on the output. In this section, we describe the basics of thresholding and evaluation metrics that are threshold-independent.

1.3.1 Sensitivity

Sensitivity is an essential issue of all anomaly detection problems. In practice, different setups are required according to the application. For example, medical tests must be performed highly sensitively to avoid neglecting an ill patient. On contrary, the system health monitoring must not be too sensitive because the operator would ignore the alarm after many false alarms. Without a doubt, no setting provides a perfect outcome. The mentioned widely used setup of sensitivity in medicine gives an opportunity for a healthy patient to be redundantly treated and detained in the hospital. On the other side, the system health could run without a raised alarm even if the system does not run optimally. However, this is still a better case, than an ignored ill patient or a crashed system due to alarm avoidance.

1.3.2 Threshold

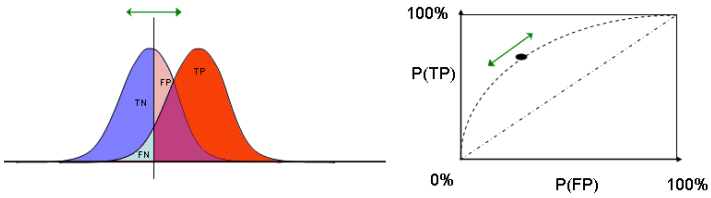


Figure 1.7: Thresholding - The graph in the upper left corner shows the distribution of anomaly scores for the regular samples (left peak) and anomalous samples (right peak). The possible threshold is demonstrated with the vertical line and the consequential classification is indicated with colors and labels (True negative, false negative, false positive, true positive). The ROC curve, which is plotted in the lower part, demonstrates all possible thresholds and their probability of true positive and false negative. [24]

The threshold is a numerical representation of the sensitivity and it decides whether the tested sample is anomalous or not according to the anomaly score. The threshold is tuned to the optimal value for a specific application. Theoretically, if the tested subject is simple or the detection is performed perfectly, it is possible to find a perfect threshold with a total true rate. In other words, the informative value of the test's result is in the separability of the distribution of regular and anomalous samples (see Fig.1.7). In addition to the method's quality, the training set has a significant influence on the result of the thresholding. Therefore it is tuned as one of the last parameters depending on the known and current data. Anyway, since

the thresholds may differ across applications, it is appropriate to utilize threshold-independent evaluation (see 1.3.3) for anomaly detection performance. If there was a single threshold, the percentage of success could be used. [25]

Note that the distributions shown in Fig. 1.7 are collected from the test data using the ground truth labels. However, in real-world applications, the distributions are often unknown when the detection pipeline is being designed and thus optimization of the threshold might be more difficult.

1.3.3 Receiver Operator Characteristics and AUC

The performance measure of the anomaly detection method must take into account all possible thresholds. Receiver operator characteristics (ROC) are utilized to analyze the performance over all thresholds. The graphical representation, which is shown in Fig. 1.7, is a parametric plot that shows the proportion of true positive and false positive rates for all possible thresholds. Note that these proportions are based on the data set. The curve always starts and finishes in the corners because the lowest threshold classifies all samples as positive thus the false and true positive rate is 1. Similarly, the highest threshold hits the opposite corner. In an optimal case, the curve is plotted near the top left corner that represents a high true positive and low false positive rate. On contrary, thresholding a random variable (random decision-making) will form a curve near the diagonal. In other words, most methods should have the curve above the diagonal.

To conclude, it has been shown that the better the method is the higher the curve is plotted which allows us to represent the quality of the method with a scalar that is independent on a specific threshold. This metric is defined as the area under the ROC curve (AUC) and it is often used in anomaly detection. [26] [27] [28] [29]

1.3.4 Other Metrics and Metric Selection

Besides ROC [31], there are a few more metrics for AD evaluation. When focused on the threshold-independent evaluation, the precision-recall (PR) curve [32] and its AUC is an alternative option. Precision, recall, and F1 are typical metrics for a non-threshold (or static threshold) binary decision.

In this work, we decided to utilize ROC AUC over PR AUC for the following reasons:

- ROC AUC is the most frequently used robust metric in AD literature. We analyzed the most cited recent surveys and comparisons of AD techniques (e.g. [33, 34, 35, 36, 37]) and all of them used ROC AUC as the main or the only metric.

- Utilizing of PR curve and F1 score is rather seldom in the AD literature. These are also known for various types of inconsistency and vulnerability to yield a biased measure (e.g. [38, 39]).

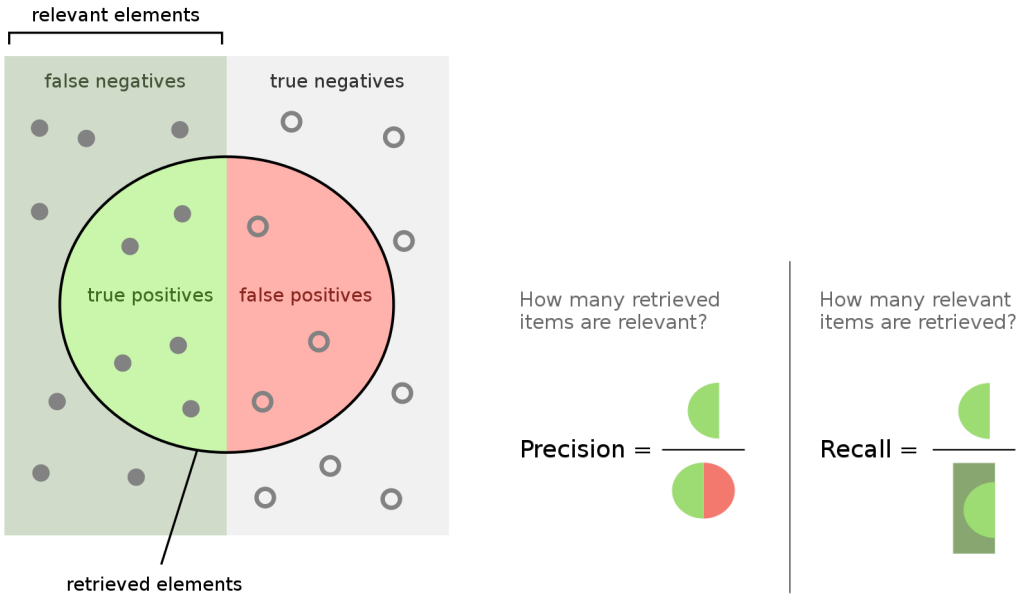


Figure 1.8: Visual demonstration of precision and recall [30]

1.4 Data Sets in Anomaly Detection Literature

In this section, we illustrate the difficulty of evaluating anomaly detection methods and depict research progress in the last decades.

A number of different benchmark sets and metrics have been used for the anomaly detection performance evaluation in the literature, thus it often is difficult to obtain a mutual comparison among the paper works. However, several benchmark sets are more frequent in the literature than others because they are built for a specific purpose such as intrusion detection or image recognition, and are widely used by their community. Three iconic data sets are described in the next paragraphs.

1.4.1 Overview of Popular Data Sets and Issues

KDD-99 [40] is a data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

MNIST [21] is a database of handwritten digits. It has been created as a sample of NIST database and the data have been pre-processed and formatted for easier usage. These data are real-world based and widely used for image recognition and many other machine learning branches due to the simplification of MNIST set.

99 DARPA IDEVAL [41] is a data set for intrusion detection. It contains network traffic and audit logs collected on a simulation network in three weeks. The first and third weeks do not contain any attack contrary to the second week when the network faced various types of attacks.

The great advantage of using one of these sets is the comparability of the results among the prior art. However, it is important to note that the data sets mentioned may no longer be relevant to current issues and could therefore become outdated. In addition to that, the sets are narrowly focused on a specific problem thus they are inappropriate to create a general benchmark for anomaly detection. As a consequence, many authors in the field of anomaly detection rather constructed their own artificial data because the existing data sets were too different from their problem.

In 2014 Sakurada [42] constructed artificial data from the Lorenz system for the purpose of processing the spacecraft's telemetry data. In 2005 Sarasamma [43] used an expert knowledge of KDD-99 (internet security) to present his method to operate optimally. He selected only the most representative features in advance, predefined several classes of outliers to the model, and moreover, modified the data set. However, this could have significantly affected the performance. Such an approach prefers the best results under given conditions (typically used in practice) rather than measuring the performance of the proposed method in general.

1.4.2 Evaluation Methodology by Emmott Et Al.

In 2013, Emmott probably reacted to the situation of missing general comparison data set for anomaly detection and introduced his methodology of creating general AD benchmarking sets using multi-class data from the UCI repository in [35].

For each source data set one selected class is used to form the non-anomalous data and some of the others are used as sources of anomalies. The concrete choice of anomaly representing classes leads up to four different data sets of four levels of detection difficulty.

Emmott demonstrated the utility of the approach by creating a number of carefully selected sets and using them to evaluate the performance of six popular AD methods. There is a large number of various multi-class data sets usually based on real-world data in the UCI repository hence the constructed benchmark sets provide a reasonable reality check. The performance evaluation could be more efficient and general due to utilizing a number of different sets.

This might be a breakthrough in anomaly detection performance measurement if other researchers start to utilize it. In 2014 Dau [44] considered the methodology as the most advanced. Since that, many works considered the methodology and adopted some of the parts.

1.5 Prior Art Anomaly Detection Methods

There are a number of methods for anomaly detection that have been studied in the following surveys [1, 4, 45, 36, 37, 35, 34, 46]. To clarify, the listed prior art is mainly focused on solid coverage of methods for vector data, however, there is a significant overlap with image data AD. Completely different methods are used for AD in time-series thus we refer to the following surveys for more details: [20, 47, 48].

1.5.1 Traditional Non-neural Techniques

Nearest neighbor techniques [49, 50] are popular due to their simplicity, reliable accuracy (which is often unsurpassed, cf. [36, 51, 6]), and adaptability to various data types. Their computational complexity, however, grows rapidly with both the dimensionality and size of the training data. Supporting structures thus have been proposed. The *k-d tree* [52] [53] [54] is a binary search tree that uses hyperplanes to divide the space to accelerate the search. The *ball-tree* [55] [54] uses hyperspheres to cover the space recursively. Despite these advances, the problem of kNN computational complexity cannot be considered as resolved.

Local outlier factor (LOF) [10] and subsequent ideas like probabilistic LOF [56] can be useful with unevenly distributed data sets. The key idea here is to deem a sample anomalous if it is significantly farther from its neighbors than they are from each other.

Isolation forest (IF) [9] and subsequent ideas like *extended* [57], *functional* [58] or *kernel* isolation [59] proved to be practical for high-dimensional problems. The key idea is to build projection trees and evaluate at which depth a sample becomes isolated. It is expected that more anomalous samples are easier to isolate, thus appearing closer to the tree root.

AD can be naturally performed using statistical approximation models. *Gaussian mixture models* have been shown useful in mammography [60], sea surveillance [61], flight operations monitoring [62], and in some cases are beneficial to use in combination with auto-encoder [63]. A simple option is to utilize *Parzen* models [64].

The standard anomaly detection knowledge base also includes *kernel PCA* methods [65], *kernel density estimation* (KDE) including *robust* KDE [66], *Histogram-based outlier score* (HBOS) [67] and *one-class support vector machines* (SVM) [68] that all have been compared to and partly outperformed by neural models, see, e.g., [69].

1.5.2 Neural Network-based Techniques

The simplest form of a neural network traditionally used for unsupervised AD is *auto-encoder* [70], where reconstruction error typically serves as a proxy to measure the anomality. Many extensions of the idea exist, e.g., [69], [71], [72], [44], [42], [73]. Auto-encoders can be viewed as the primary unsupervised neural AD technique.

They do not model the distribution of anomalies but optimize a proxy criterion like the reconstruction error. This can limit the success of AEs.

Other types of neural network AD models depend on additional knowledge about possible outliers or other indirect information about the anomaly. Therefore the models are more related to classification or other fields. (see, e.g., [74], [75], [43], [76], [77]).

More recently, various types of *generative neural models* have been applied to anomaly detection in fields including clinical imaging, industrial time series and intrusion detection [78],[79],[80]. A particularly successful semi-supervised *GANomaly* model has been applied in X-ray screening [5]. The model jointly learns the generation of high-dimensional image space and the inference of latent space. For a wider overview of generative AD techniques, see [4].

1.5.3 Neural Network-based Techniques Using Auxiliary Data

Other types of neural networks have been used to estimate and simulate the nature of the anomalies in the training phase. In other words, *auxiliary data* is used (explicitly or implicitly) when training the neural model. An intuitive auxiliary set with binary labels was utilized in [81] to represent the manifold. However, the method is limited by the assumption that the non-anomalous data lie on a well-sampled, locally linear low dimensional manifold. The auxiliary set consists of the training set and potentially anomalous samples that are generated with the Euclidean radius around the training data. The authors claim that the collision probability of the generated anomalous samples and training set is low due to the assumptions.

Supervised AD is performed in [82] where *outlier exposure* is used to train the model with an auxiliary set skimmed from other sources (i.e., pictures skimmed from the web) in addition to the training set. The auxiliary set is purified not to contain similar samples to the training class, thus, the detector is effectively trained with two different classes. The authors also consider the difficulty of creating the artificial auxiliary set (i.e., with Gaussian noise) that teaches the network to generalize the unseen anomaly distributions in the unsupervised scenario in contrast to the supervised.

1.5.4 Detector Ensembles

In many applications, it has been shown that *ensembles of anomaly detectors* perform better than a single detector [83] [84] [85] [86]. This is common, particularly in cyber-security [87] [88].

1.6 Summary on Limitations of Prior Art in Anomaly Detection

The listed prior art captures the clear gap between methods offering excellent accuracy and methods that are suitable for application on large-scale data. Specifically, distance-based methods are operating well on a wide range of problems, but their scalability is limiting. On the other hand, methods based on neural networks are well scalable and have major HW and SW support for large-scale and industrial applications, however, they do not provide such reliability in AD to be massively used in the real-world environments in contrast to classification problems where the neural models are constantly gaining importance.

The ideas provided in this work follow an alternative approach to unsupervised AD consisting of a neural network-based approximation of an existing anomaly detector's score function. Initially introduced in our previous works [89], [90], and [91], this class of methods depends on auxiliary samples with a non-binary label, which cover both the area of the training set (non-anomalous) and its close and more distant neighborhood, i.e., the area potentially significant to detect anomalies.

Chapter 2

Proposing Surrogate Neural Network for Anomaly Detection

In this chapter, we introduce an alternative use of neural networks in AD [90, 89, 91] that enables mimicking any other existing AD method. As a result of this, we will show computational performance gains and also AD accuracy gains coming from a form of regularization that the use of a neural model enables.

2.1 Motivation

In the following, we will consider the most common type of neural network-based AD – the auto-encoder – and we will outline its limitations. The newest class of neural ADs takes use of the advances of generative adversarial networks (GAN [4] and GANomaly [5]). Although GAN-based ADs proved capable of outperforming AEs, major problems persist, especially in large-scale application areas, moreover, are quite difficult to train to satisfactory results [6].

In contrast to these mainstream developments, we propose a more straightforward and universal approach to utilizing neural models in AD. In the following, we introduce in detail our results that were initially published in [90] and [89] and later summarized in-depth in [91].

2.2 Surrogate Neural Networks for Anomaly Detection

The methodology we propose aims at approximating an existing anomaly detector's score function using a surrogate neural network. Let us refer to the detector to be approximated as the *source anomaly detector*.

In the following, we will consider kNN in most cases in place of the *source anomaly detector*. This is well in line with our goal to address scalability in industrial systems.

The kNN AD is known to perform exceptionally well in many fields including cybersecurity. At the same time, it is challenging to apply kNN in many industrial settings due to its space and time complexity properties.

First, we create an auxiliary data set covering the source detector's input space. For each sample in the auxiliary data set, the source detector's anomaly score is computed and assigned to the sample as its label. Then, the auxiliary data set is used to train a standard multilayer perceptron (MLP) with a single output.

Having the training set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where $\mathbf{x}_i \in \mathbb{R}^d, \forall i \in \{1, \ldots, n\}$, and $\mathbb{R}^d$ is a d-dimensional vector space. Let us denote $\mathbf{A}$ the auxiliary data set of m samples where

$\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m\}$, $\mathbf{a}_i \in \mathbb{R}^d$, $\forall i \in \{1, \ldots, m\}$

and Y be the vector of respective anomaly scores computed using the source anomaly detector, where $Y = \{y_1, y_2, \ldots, y_m\}$, $y_i \in \mathbb{R}$, $\forall i \in \{1, \ldots, m\}$.

For simplicity, we assume that the *size* of MLP hidden layers and their *number* can be viewed as hyper-parameters p and q, respectively, and that both parameters can be determined through hyper-parameter search.

2.2.1 Auxiliary Data Sets

At first, the auxiliary set $\mathbf{A}$ needs to be computed from the training set $\mathbf{X}$. The actual auxiliary set construction can be done in many ways. In the following, we discuss two options: the trivial coverage of the input space by a hyper-block (*unifrom approach*), and efficient construction, which employs the distribution of the input data (*adaptive approach*). Fig. 2.2 illustrates the two options.

Uniform Auxiliary Data Set

The baseline idea of the *uniform auxiliary set* construction that we proposed in [90] is naïve as it attempts to cover the space uniformly on a rectangular subspace defined as the smallest enclosing hyper-block that contains all points in the input data space. More specifically:

1. A bounding hyper-block of $\mathbf{X}$ is determined as the smallest enclosure of the input data, defined by the vector of lower bounds $\mathbf{h}_l$ and upper bounds $\mathbf{h}_u$ such that $\mathbf{h}_l^{(j)} \leqslant \mathbf{x}_i^{(j)} \leqslant \mathbf{h}_u^{(j)} \quad \forall i \in \{1, \ldots, n\} \quad \forall j \in \{1, \ldots, d\}$ where $\mathbf{x}_i^{(j)}$ represents j-th element of i-th vector from $\mathbf{X}$

2. The hyper-block is filled with randomly generated and uniformly distributed samples $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m\}$. By default, we consider uniform random sampling. Note that the choice of m for concrete problem may depend on n and d (see also Sect. 2.4.2).

3. The anomaly score vector Y is constructed so that for each auxiliary sample $\mathbf{a}_i, i \in \{1, \ldots, m\}$ the respective $y_i \in Y$ is computed as the source anomaly detector's score on $\mathbf{a}_i$.

Remark: in case of kNN the y_i is computed as mean distance $G(\cdot)$:

$$y_i = G(\mathbf{a}_i) = \frac{1}{k} \sum_{j=1}^{k} D_j(\mathbf{a}_i) \tag{2.1}$$

where $D_j(\mathbf{a}_i)$ represents the j-th smallest distance of $\mathbf{a}_i$ to samples from $\mathbf{X}$. Note that the number of neighbors k is a parameter [92, 93].

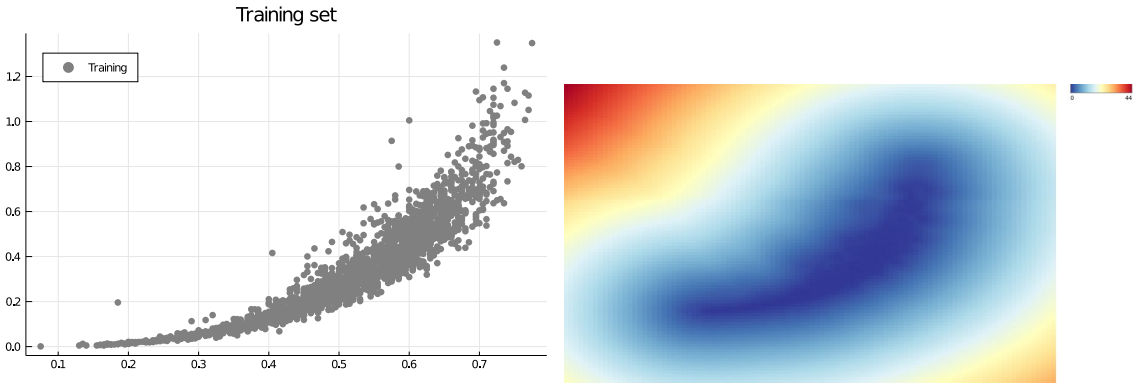


Figure 2.1: Heat-map illustration of anomaly scores induced by kNN anomaly detector on benchmark Abalone data set. See: a) input (training) data, b) anomaly score heat-map.

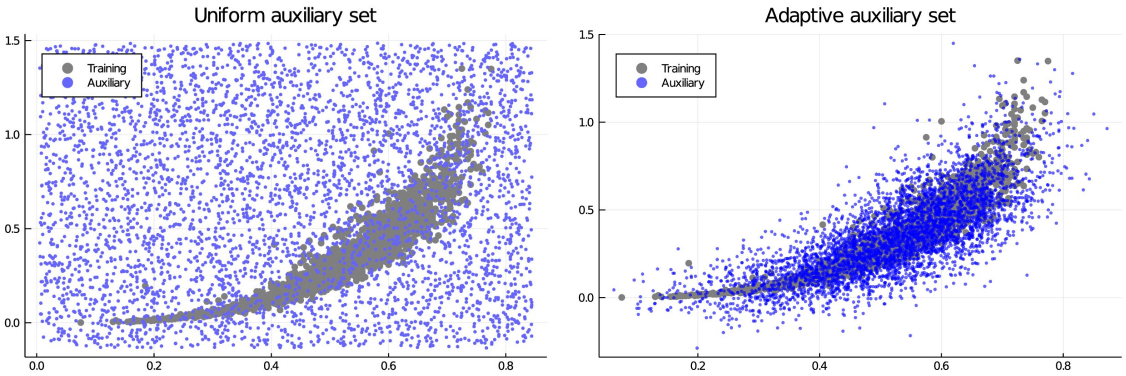


Figure 2.2: Construction of the auxiliary set(s) to be eventually used to train a neural anomaly detector. Auxiliary samples need to cover all areas of notable kNN anomaly score variance.

Adaptive Auxiliary Data Set

The uniform auxiliary set as defined above is sub-optimal due to multiple reasons. Clearly, the distribution of points in the uniform auxiliary set does not reflect the

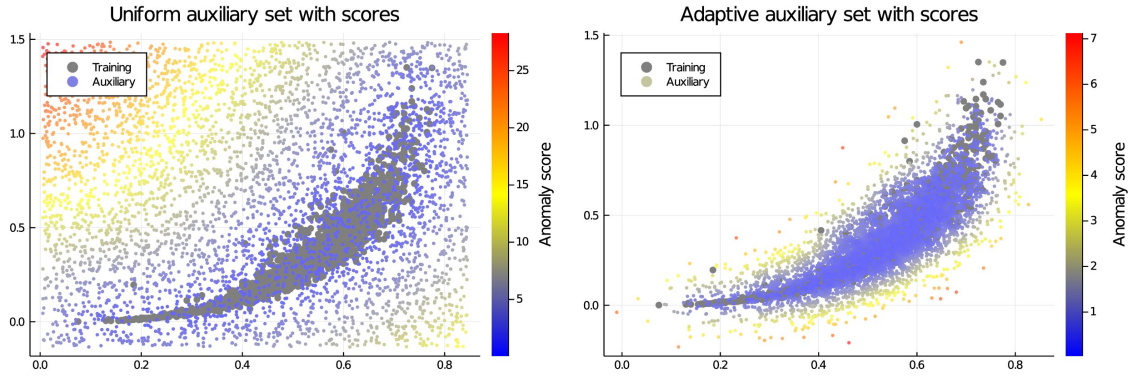


Figure 2.3: Finalizing the construction of training set(s) to be eventually used to train a neural anomaly detector. Each auxiliary sample gets labeled by its respective anomaly score (illustrated here by colors on a heat scale) computed using kNN from input data (overlaid gray dots). Compare the distribution of auxiliary anomaly scores to the heat map in Fig. 2.1.

varying importance of various regions in the auxiliary space; the uniform auxiliary set can easily waste sampled points in regions of no importance while lacking coverage in dense and complicated manifolds.

This problem gets worse with increasing dimensionality. This "curse of dimensionality effect" can be illustrated by the simple example of data distributed within a hyper-sphere of unit radius. Assuming we have the hyper-sphere enclosed in an auxiliary hypercube with edge length equal to 2 (the radius of the bounded spherical cluster is thus equal to 1), the ratio of hyper-sphere volume over hypercube volume decreases with increasing dimensionality (see Fig. 2.4). Only a negligible fraction of auxiliary samples would be relevant in problems with more than low single-digit dimensionality. For example, only about 16% of total auxiliary samples will be relevant for $d = 5$, 3.7% for $d = 7$ and 0.25% for $d = 10$ if we used the uniform auxiliary set in this case.

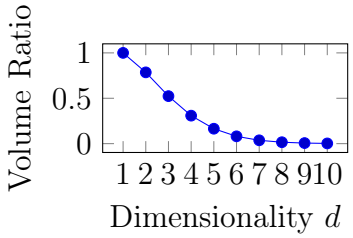


Figure 2.4: Inefficiency of covering input space by uniform auxiliary data sets: d-dimensional hyper-sphere to hypercube volume ratio

Another problem with hyper-block is the possible loss of information. Auxiliary data generated strictly within a hyper-block cannot approximate the continuity of anomaly scores with growing distance from the input samples. The sharp auxiliary set boundary thus can distort the eventual surrogate model.

To resolve both of the problems above, we propose to construct the auxiliary data set

adaptively to reflect the distribution in input data. This is achieved by generating auxiliary samples according to a modified Parzen estimate of the input density. No bounding hyper-block is thus needed, while the auxiliary samples now become more frequent in areas of more detail.

Such auxiliary data set should provide more detailed coverage of anomaly score distribution than the uniform auxiliary data set with the same number of auxiliary samples. The *adaptive auxiliary data set* is constructed as follows:

1. Optimal variance h for Parzen window approximation of $\mathbf{X}$ is determined (by default we use cross-validation and random search on training data).
2. The auxiliary set $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m\}$ is generated as realization of the Parzen distribution as follows: iterate over samples of $\mathbf{X}$ and create $\mathbf{a}_i = \mathbf{x}_i + \mathcal{N}(0,\, h \cdot k_{var})$ where k_{var} (variance multiplicative coefficient) is a parameter:

   $$\forall i \in \{1, \ldots, m\} : \ \mathbf{a}_i = \mathbf{x}_{(i \bmod n)} + \mathcal{N}(0,\, h \cdot k_{var})$$

   Note that if $m > n$, multiple auxiliary samples get generated based on a single input sample. The choice of m and k_{var} for the concrete problem is discussed in Sect. 2.4.2.
3. The anomaly score vector Y is constructed in the same way as for the uniform auxiliary set (see Eq. 2.1).

See Fig. 2.2 for the difference between *uniform* and *adaptive* auxiliary sample distributions. See Fig. 2.3 for the same auxiliary sample distributions enriched by anomaly score labels. The impact of the improved *adaptive* auxiliary set efficiency is also shown in Fig. 2.15.

2.2.2 Training the Surrogate Neural Model

We can now train a multilayer perceptron (MLP) on the auxiliary training set $\mathbf{A}$ to predict anomaly scores Y. We parametrize the size of hidden layers p and the number of hidden layers q (see Fig. 2.5). We minimize the mean squared error (MSE) between the *predicted* scores and *ground truth* scores using the *Adam* optimizer [94]. For further details of our experimental setup see Sect. 2.4.2.

In more detail, the input vector $\mathbf{a}_i \in \mathbb{R}^d$ is projected to $y'_i \in \mathbb{R}$ as follows:

$$y'_i = f_\theta(\mathbf{a}_i) = f^{(q+1)}_{\theta^{(q+1)}}(\ldots f^{(2)}_{\theta^{(2)}}(f^{(1)}_{\theta^{(1)}}(\mathbf{a}_i))) \tag{2.2}$$

where $f^{(j)}_{\theta^{(j)}}$ represents the j-th layer of the NN and the layer propagation is defined as:

$$f^{(j)}_{\theta^{(j)}}(\mathbf{a}_i) = c(\mathbf{W}^{(j)}\mathbf{a_i} + \mathbf{b}^{(j)}) \tag{2.3}$$

thus $f^{(j)}$ is parameterized by $\theta^{(j)} = \{\mathbf{W}^{(j)}, \mathbf{b}^{(j)}\}$, c is an activation function, $\mathbf{W^{(j)}}$ is a weight matrix and $\mathbf{b^{(j)}}$ is a bias vector of the j-th layer.

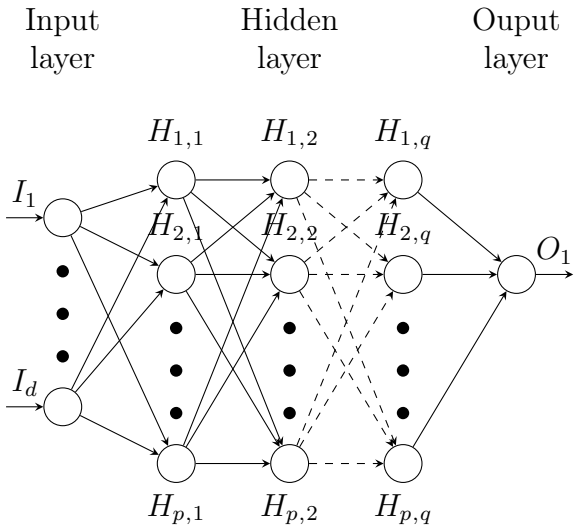


Figure 2.5: Default structure of the surrogate neural network. The size of hidden layers p and the number of hidden layers q are problem-dependent parameters subject to optimization

The parameters of the model are optimized with $\mathbf{A}$ and Y such that the average loss function is minimized. By default we use MSE:

$$\theta^* = \arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \left(y_i - y_i' \right)^2 \tag{2.4}$$

2.2.3 Model Applicability

In the following evaluation sections, we will provide two disstinct evaluation scenarios aimed at illustrating the applicability of the proposed model both to common benchmark problems (Sect. 2.3) and to large-scale industrial problem (Sect. 2.4).

The method was originally proposed with the uniform approach which is simple and straightforward while performing well on publicly available data sets and the widely used evaluation methodology proposed by Emmott [35]. We compare the *uniform approach* with the kNN as a source detector and with AE on 64 problems.

Our ultimate goal, however, was to perform anomaly detection on industrial data from the field of computer security. Such data differ from the public data sets substantially, typically are large-scale, unevenly distributed, and often more complex to utilize for the ML methods. Since the *uniform approach* does not reach the high-accuracy standards that are common in the industrial context, due to its deficiencies (see Sect. 2.2.1), we developed the *adaptive approach* which is more powerful for such a complex data.

The evaluation of the *adaptive approach* is given in Sect. 2.4 where the evaluation mainly focuses on the industrial data set and comparison among a number of state-of.the-art detectors with a detailed study of statistical significance with confidence intervals.

2.3 Experimental Evaluation on Public Benchmarks

We compare the proposed methodology to standard kNN-based anomaly detection and then to auto-encoder-based anomaly detection. We compare the area under the curve of receiver operating characteristics (AUC of ROC) [31] of the three approaches on a body of benchmark data. Additionally, we compare the computational complexity of the proposed method to kNN with respect to increasing dimensionality and training data size.

2.3.1 Data Sets

Our aim is to compare the methods on a variety of data sets of various properties, ideally such that are widely used in literature. To do so we have adopted the experimental protocol of Emmott [35], who has introduced a methodology of creating general AD benchmarking sets using multi-class data from the UCI repository (see Sect. 1.4.2).

For our comparison, we thus include 64 data sets generated using Emmott's methodology from 18 source data sets representing real-world data. To give more insight into methods' performance under various conditions the data sets are grouped according to their difficulty. We thus perform our evaluation on *easy* (see Table 2.3), *medium* (Table 2.4), *hard* (Table 2.5) and *very hard* (Table 2.6) problems. The numbers of samples in data sets vary from 66 to 12332, dimensionalities vary from 4 to 90. For details on individual data sets see [35, 95].

2.3.2 Evaluation Setup

Evaluation Methodology

To construct training and testing sets, in all cases random resampling is used such that 75 % of normal (non-anomalous) samples are used for training and the rest 25 % for testing. The anomalous samples are only used for testing. The accuracy score is measured by AUC of ROC [31] as it is common in literature. The advantage of this metric is discussed in sect.1.3.3.

Setup of k-Nearest Neighbors

To evaluate kNN accuracy we compute AUC according to the anomaly score obtained as mean distance $G(\cdot)$ introduced in Equation (2.1).

To evaluate kNN computational complexity we consider multiple kNN versions: the *basic kNN*, which is implemented as a brute tree, *k-d tree*, and *ball tree*. The *k-d tree* [52, 53, 54] is a special type of binary search tree that uses hyperplanes to divide the space to accelerate the search. Similarly, the *ball-tree* [55, 54] uses hyperspheres

to cover the space recursively. The usage of the supporting structure does not affect the precision of the method. However, the time complexity may differ significantly.

The optimal choice of the parameter k which is essential for kNN is not addressed in this work. However, we observed $k = 5$ as the best performing on average for all the sets thus it is used for all presented experiments. Remark: note that the proposed method is applicable for any k.

Auto-Encoders Setup

We evaluate the denoising three-layer auto-encoder according to [44, 72]. When computing AUC, the anomaly score is proxied by reconstruction error. AEs are subject to parametrization; for the tests, we needed to decide on the number of neurons per hidden layer, initialization, and the type and magnitude of noise. To ensure a unified approach across all data sets and to avoid the worst local maxima we opted for a meta-optimization procedure. For each benchmark data set, we trained multiple AE models with varying parameters to eventually retain the one with the best loss function result.

We observed that *Gaussian noise* worked better than *salt and pepper noise* across the considered data sets. Four different magnitudes of noise are tried with deviations between 0.01 and 0.2 while the samples were scaled to $[0, 1]$ for each dimension.

To set the number of hidden neurons we propose to take guidance from the pre-analysis of the data. The number of hidden neurons is selected empirically among six various setups implied by expected relative variance. First, we define a relative variance (0.7, 0.8, 0.9, 0.95, 0.97, 0.98) that we want to preserve with the encoding. Then the required number of hidden neurons is estimated using *principal component analysis* (PCA) [96] as the lowest number of dimensions required to preserve the relative variance.

We observed only negligible improvement from repeated random initialization. All models were trained in 300 000 iterations; varying the number of iterations also proved to have only negligible impact.

Hence the eventual meta-optimization procedure consists of building the 24 models (4 noise parameters, 6 hidden layer size parameters) and retaining the one with the best-achieved loss function result.

Proposed Method Setup

To evaluate the accuracy of the proposed model we compute AUCs using the neural network introduced in Section 2.2. The method is subject to parametrization: its performance can be affected by the properties of the auxiliary data set as well as by the standard neural model parametrization (number of layers, number of neurons in layers, etc).

We fixed the auxiliary set construction parameters for all experiments as follows. We fixed $k = 5$ in kNN used for auxiliary data set construction to get results

comparable to the standalone kNN anomaly detector. The auxiliary data set is constructed as described in Section 2.2.1 with the total number of auxiliary samples set to $m = n \cdot d^2 \cdot 150$. The choice of the parameter m is empirical and reflects a trade-off between model accuracy and the computational complexity of the training.

ReLU ($f(x) = \max(0, x)$) activation function is used for all neurons (except input). The size of the batch is set always to 80. We opted for a simple meta-optimization of neural model parameters so as to avoid the worst local optima. The same procedure is applied across all benchmark data. For this purpose we train for each training data set multiple models, to eventually retain the version with the best loss function result. The variation across training runs consists of 2 or 3 hidden layers, hidden layer size $3d$ or $5d$, multiple random weight initializations, number of iterations thresholded by six values between 1000 and 300000.

2.3.3 Detection Accuracy Results

Assessing the results of three methods over multiple datasets can be done in multiple ways [97]. We focus on the pair-wise comparison of the *proposed method* separately to *kNN* and *auto-encoder*.

We summarize the best achieved AUC accuracies in four tables, each covering one problem difficulty level: Table 2.3 for *easy*, Table 2.4 for *medium*, Table 2.5 for *hard* and Table 2.6 for *very hard*. Note that we report results rounded to 4 decimal places. Pair-wise better results are emphasized in bold.

To obtain a global picture we then aggregate results over data sets and use Wilcoxon signed rank test to verify the statistical significance (at 0.05 level) of one method's win over the other.

When compared to kNN, Table 2.1 shows that the proposed method performed better on the notable majority of *very hard* problems. Overall, the proposed method performed better on 36 problems while the kNN on 28. However, the statistical significance is not approved for any of the problem difficulty groups. Nevertheless, we have achieved the primary goal of providing a neural model that achieves comparable or better accuracy when compared to kNN with considerably lower computational complexity in the application phase (see also Section 2.3.4).

When compared to auto-encoders, Table 2.2 shows that the proposed method outperformed AEs in all difficulty levels. The most notable success occurs for *medium* and *very hard* problem levels where statistical significance is achieved. In summary, the NN performed better for 43 problems while the auto-encoder for 21.

Remark: Note that the Wilcoxon test could not be performed across difficulty groups, because it requires independent results.

Table 2.1: Counts of wins of the *proposed method* versus *kNN*, grouped by problem difficulty. Wilcoxon signed rank test at 0.05 level is used to verify the statistical significance of wins

	Easy	Medium	Hard	V. Hard	Sum
Proposed NN	10	8	9	9	36
kNN	8	10	7	3	28
Significance	no	no	no	no	–

Table 2.2: Counts of wins of the *proposed method* versus *auto-encoder*, grouped by problem difficulty. Wilcoxon signed rank test at 0.05 level is used to verify the statistical significance of wins

	Easy	Medium	Hard	V. Hard	Sum
Proposed NN	12	13	9	9	43
Auto-encoder	6	5	7	3	21
Significance	no	yes	no	yes	–

Table 2.3: AUC scores for *easy* problems. Table provides two pairwise comparisons of proposed NN vs kNN and auto-encoder

Set	d	NN	kNN	NN	AE
abalone	10	0.972	**0.994**	**0.972**	0.961
blood-transfusion	4	0.955	**0.987**	0.955	**0.991**
breast-cancer-wis.	30	**0.989**	0.969	**0.989**	0.978
breast-tissue	9	**1.000**	0.997	**1.000**	0.996
cardiotocography	27	**0.884**	0.566	**0.884**	0.617
ecoli	7	**0.927**	0.918	**0.927**	0.876
glass	10	**0.816**	0.785	**0.816**	0.803
haberman	3	**0.996**	0.972	**0.996**	0.911
ionosphere	33	0.811	**0.962**	0.811	**0.984**
iris	4	**1.000**	0.936	**1.000**	0.569
libras	90	0.500	**0.768**	0.500	**0.562**
magic-telescope	10	0.901	**0.946**	**0.901**	0.896
page-blocks	10	**0.990**	0.981	**0.990**	0.976
parkinsons	22	**0.895**	0.830	**0.895**	0.862
pendigits	16	0.975	**0.994**	**0.975**	0.945
pima-indians	8	0.890	**0.920**	0.890	**0.892**
sonar	60	0.500	**0.605**	0.500	**0.664**
spect-heart	44	**0.500**	0.277	0.500	**0.502**

Table 2.4: AUC scores for *medium* problems. Table provides two pairwise comparisons of proposed NN vs kNN and auto-encoder

Set	d	NN	kNN	NN	AE
abalone	10	0.871	**0.935**	**0.871**	0.825
blood-transfusion	4	0.800	**0.806**	0.800	**0.902**
breast-cancer-wis.	30	**0.973**	0.918	**0.973**	0.954
breast-tissue	9	**1.000**	0.964	**1.000**	0.958
cardiotocography	27	**0.855**	0.561	**0.855**	0.622
ecoli	7	0.837	**0.843**	**0.837**	0.790
glass	10	**0.711**	0.685	**0.711**	0.703
haberman	3	**0.963**	0.955	**0.963**	0.865
ionosphere	33	0.961	**0.993**	0.961	**0.988**
iris	4	**0.980**	0.924	**0.980**	0.858
libras	90	0.500	**0.628**	0.500	**0.532**
magic-telescope	10	0.868	**0.898**	**0.868**	0.850
page-blocks	10	0.956	**0.983**	**0.956**	0.952
parkinsons	22	**0.638**	0.525	**0.638**	0.551
pendigits	16	0.869	**0.974**	0.869	**0.873**
pima-indians	8	0.787	**0.789**	**0.787**	0.735
sonar	60	0.502	**0.717**	0.502	**0.701**
spect-heart	44	**0.589**	0.236	**0.589**	0.434

Table 2.5: AUC scores for *hard* problems. Table provides two pairwise comparisons of proposed NN vs kNN and auto-encoder

Set	d	NN	kNN	NN	AE
abalone	10	0.520	**0.550**	0.520	**0.529**
blood-transfusion	4	**0.716**	0.521	**0.716**	0.714
breast-cancer-wis.	30	**0.735**	0.627	0.735	**0.742**
breast-tissue	9	**0.542**	0.463	0.542	**0.559**
cardiotocography	27	**0.780**	0.372	**0.780**	0.526
ecoli	7	0.702	**0.736**	**0.702**	0.666
glass	10	**0.607**	0.556	0.607	**0.651**
haberman	3	**0.937**	0.922	**0.937**	0.796
ionosphere	33	0.542	**0.863**	0.542	**0.782**
iris	4	**0.880**	0.842	**0.880**	0.574
magic-telescope	10	0.835	**0.850**	**0.835**	0.807
page-blocks	10	0.937	**0.963**	0.937	**0.941**
parkinsons	22	**0.432**	0.365	0.432	**0.515**
pendigits	16	0.886	**0.977**	**0.886**	0.878
pima-indians	8	0.606	**0.647**	**0.606**	0.606
spect-heart	44	**0.500**	0.085	**0.500**	0.129

Table 2.6: AUC scores for *very hard* problems. Table provides two pairwise comparisons of proposed NN vs kNN and auto-encoder

Set	d	NN	kNN	NN	AE
abalone	10	**0.521**	0.447	**0.521**	0.498
blood-transfusion	4	**0.506**	0.366	**0.506**	0.471
breast-tissue	9	**0.504**	0.413	**0.504**	0.450
cardiotocography	27	**0.790**	0.307	**0.790**	0.437
ecoli	7	**0.567**	0.504	**0.567**	0.566
glass	10	0.321	**0.568**	0.321	**0.529**
haberman	3	**0.555**	0.529	**0.555**	0.498
iris	4	**0.740**	0.632	**0.740**	0.493
magic-telescope	10	**0.667**	0.643	**0.667**	0.590
page-blocks	10	0.854	**0.874**	0.854	**0.872**
pendigits	16	0.857	**0.924**	**0.857**	0.817
pima-indians	8	**0.470**	0.428	0.470	**0.471**

2.3.4 Inference Speed Results

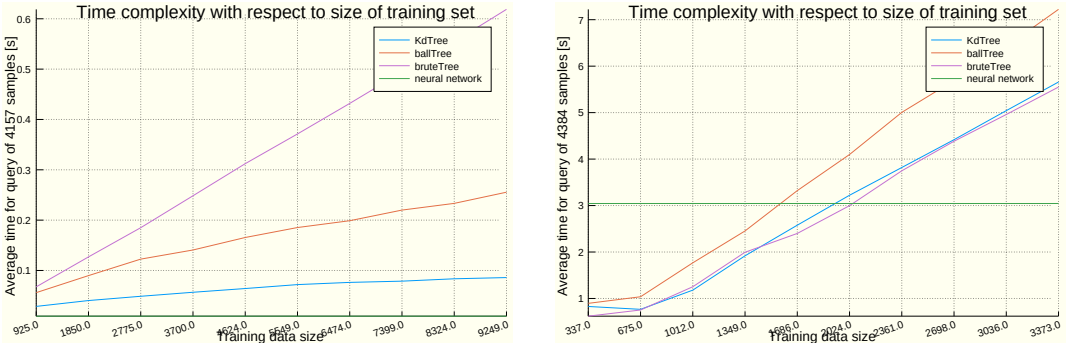


Figure 2.6: Anomaly detectors' prediction time dependence on training data size in the application phase. Tested on *magic telescope* and *Isolet* data sets. Neural model prediction speed does not depend on training data size (note the close-to-zero time in *magic telescope* case)

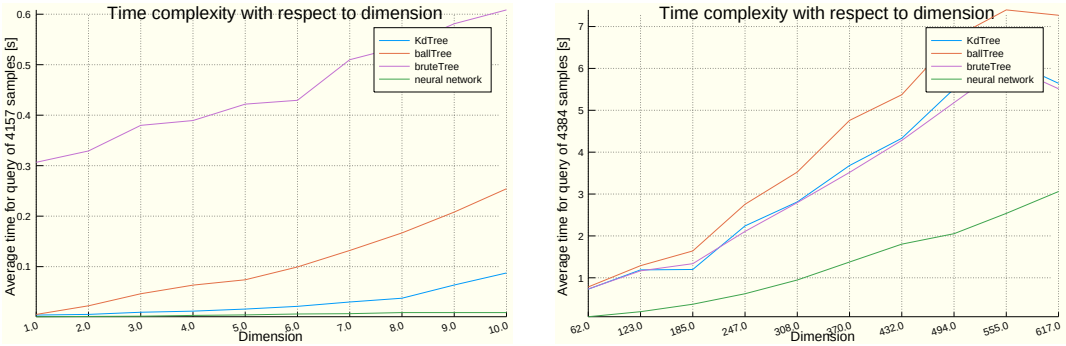


Figure 2.7: Anomaly detectors' prediction time dependence on dimensionality in the application phase. Tested on *magic telescope* and *Isolet* data sets

The time complexity of the proposed method in the detection phase is its main expected advantage over kNN, including advanced kNN forms that utilize search trees. When measuring the detection time we assume that the neural network is already trained and the kNN search tree (if any) built.

We illustrate the advantage on two data sets: *magic telescope*, (10-dim., 12332 samples) and *Isolet* (617-dim., 4497 samples), both of medium difficulty. Figure 2.6 compares the detection speed of the proposed neural model to various forms of kNN with respect to dependency on training data size. To construct the graph we run a series of tests on the same data set while gradually removing samples. Figure 2.7 compares the same anomaly detectors with respect to dependency on data dimensionality. To construct the graph we run a series of tests on the same data set while gradually (randomly) removing features.

Note that the neural model has constant complexity with respect to the number of training samples. This makes it potentially very useful whenever a kNN would perform well only with large sample sets. Figure 2.7 illustrates that even with respect

to growing dimensionality the neural model can be expected to keep the edge over kNN models.

Remark: Note that to measure the time complexity of kNN and of the proposed method we set the test environment to use a single CPU core on a computer in a controlled environment to minimize unwanted system influences.

2.3.5 Discussion

To give more insight into how the proposed model replicates kNN-induced anomaly distribution we provide heat-maps in Figures 2.8 and 2.9. To construct the heat-maps the data sets were transformed into 2D space using PCA. The respective anomaly in each pixel position is marked by color on a scale from blue (lowest anomaly) to red (highest anomaly).

From the results achieved so far, we observed that for the proposed neural network it gets more difficult to keep up AUC with kNN with increasing dimensionality. This is not surprising as neural networks are known to require large numbers of samples; with increasing dimensionality, this effect gets more pronounced. If training complexity is not the limiting factor, then the effect can be compensated for by the increased size of the auxiliary set. The applicational phase time complexity advantage is then however the more striking.

The accuracy of the proposed method depends crucially on the number and distribution of auxiliary samples. In this evaluation of the concept, we assume only the simplest definition of the auxiliary set *uniform approach* and setup (cf. Section 2.3.2).

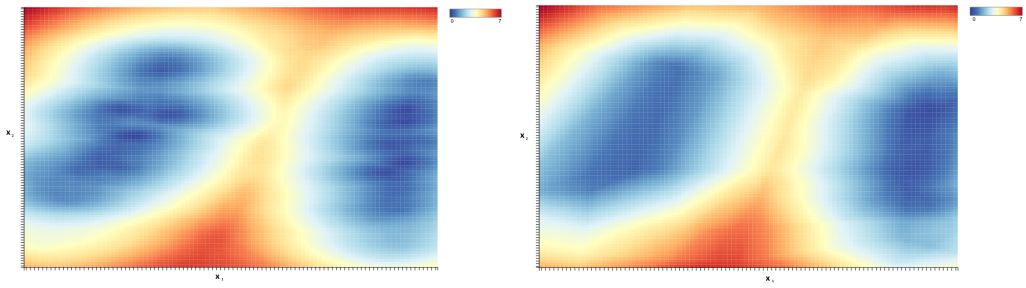


Figure 2.8: Anomaly scores on a 2D projection of *Iris* data set. Left: anomaly scores obtained by kNN. Right: anomaly scores obtained by the proposed model. Warmer color depicts higher anomaly.

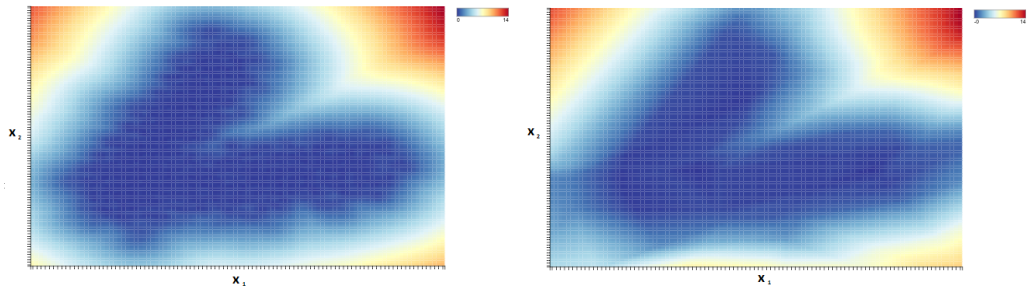


Figure 2.9: Anomaly scores on a 2D projection of *waveform* data set. Left: anomaly scores obtained by kNN. Right: anomaly scores obtained by the proposed model. Warmer color depicts higher anomaly

2.4 Experimental Evaluation in Industrial Setup

We evaluate the proposed method (both *uniform* and *adaptive* approach) on a real problem in cyber-security—the discovery of outlying (and thus suspicious) events in large-scale computer networks. For this purpose, we use network traffic telemetry data.

To obtain a baseline we evaluate a number of state-of-the-art algorithms: one class kNN [50], LOF [10], IF [9], auto-encoder, and generative adversarial networks [4][5] as well as a simple Parzen detector [64]

Subsequently, we construct a surrogate anomaly detector from the best performing (but slow) baseline source detector. We then include the surrogate detector in the overall comparison. We primarily verify the achieved improvement of inference speed. Secondarily, we verify whether the surrogate detector can match the accuracy of the baseline source detector.

Additionally, we test the robustness of the proposed surrogate approach to variations in auxiliary data set parametrization and to modifications of the auxiliary set construction procedure.

2.4.1 Cyber-Security Data Set

We perform the evaluation on a data set provided by Cisco Systems, described in detail in [98]. The data represents persistent connections observed in computer network traffic using the *NetFlow* protocol. A connection between each device-server pair is logged as a series of flow records. Because a single flow record contains only minimal usable information (transferred data sizes, timing and source-destination identifiers), it is needed to build models from at least sequences of flows. We follow the methodology from [98] where a series of flows is transformed into a vector through feature extraction. The features are expert-designed with the aim to maximally preserve connection characteristics. They are:

- `Average flow duration`
- `Flows inter-arrival times mean`

- `Flows inter-arrival times variance`
- `Target autonomous system uniqueness`
- `Target autonomous system per-service uniqueness`
- `Unique local ports count`
- `Byte count weighted by target autonomous system uniqueness`
- `Device overall daily activity deviation from normal`
- `Remote service entropy`
- `Remote service ratio`

In our data set each sample vector represents a 5-minute traffic window, which is the standard in industrial detection systems. The number of samples is 222 455. The data is multi-class, with classes distinguishing various types of benign and malicious connections.

Preprocessing Data for Anomaly Detection

In order to evaluate anomaly detectors on the available data, we adopt the experimental protocol of Emmott [35] (see Sect. 1.4.2). The protocol defines the transformation of multi-class input data into AD benchmark data. To give a varied view of the evaluated anomaly detectors, we have used the Emmott protocol to produce four AD benchmark data sets of increasing difficulty. Emmott's procedure categorizes malicious classes into four groups based on the evaluation of the anomalousness level of the respective malicious samples. Then, for each of the four groups, a new data set is constructed, taking all benign samples together with samples from the single respective group. In the following, we will thus refer to *easy*, *medium*, *hard*, and *very hard* problems.

2.4.2 Evaluation Setup

To construct the training and testing sets, random resampling (8×) is adopted such that for each sampling iteration, 75% of normal (non-anomalous) samples are utilized for training while the remaining 25% are utilized for testing. The anomalous samples are used only in the testing phase. Anomaly detectors' inference speed is measured in seconds on a single Intel Core i7 vPro 8th Generation. Detection accuracy is measured with AUC of ROC [31] as is common in the literature. Remark: we choose random resampling over cross-validation consistently with the literature [34] [33] [97] to mitigate the data-imbalance problem in AD testing.

Setup of Baseline Detectors

For the evaluation, we set the hyper-parameters and use the baseline anomaly detectors as follows.

To evaluate kNN accuracy, we compute AUC according to the anomaly score obtained as mean distance $G(\cdot)$ introduced in Eq. (2.1). The optimal choice of the

parameter k which is essential for kNN is not addressed in this work. However, we observed $k = 5$ as the best performing across our experiments. To evaluate kNN inference speed, we utilize several variants of search trees identically to the setup given in Sect. 2.4.2.

For *isolation forest*, we performed a grid search to choose the best number of trees from {100, 200, 400} and the number of samples from {256, 512}. We select the best performing parameters (on validation data) for each problem difficulty.

For *local outlier factor*, we set the parameter $k = 5$.

For *auto-encoder* evaluation, we utilize the setup given in Sect. 2.4.2 with the only difference, the number of hidden neurons was selected with a full-grid search in $\{1, \ldots, 10\}$. Thus to briefly recapitulate the setup, the eventual meta-optimization procedure consists of building the 40 models (4 noise parameters, 10 hidden layer size parameters) and choosing the one with the best achieved AUC on validation data for each problem difficulty.

We include two forms of *generative adversarial networks* in the evaluation. First, we include the GAN-based AD by Zenati et al. [80]. Specifically, we used the implementation from [34] and optimized the respective parameters on the following ranges: *dim(**z**)* on $\{2, 4, \ldots, 256\}$, *number of dense layers* on {2, 3, 4}, and α on $\{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. See [34] for details of the parameters and their recommended ranges.

To evaluate the *GANomaly* AD [5], we also used the implementation and from [34]. We optimized the respective parameters on the following ranges: *decay* on $\{0, 0.1, \ldots, 0.5\}$, w_{adv}, w_{con}, w_{enc} on $\{1, 10, 20, \ldots, 100\}$, λ on $\{0.1, 0.2, \ldots, 0.9\}$, $R(x)$ and $L(X)$ on {MAE, MSE}, *number of convolution layers* on $\{1, 2, 3, 4\}$ and *number of channels* $\{8, 16, \ldots, 128\}$. See [34] for details of the parametrization.

For the *Parzen*-based AD, we use the same setup as described in Sect. 2.5.2. The Gaussian kernel is optimized with cross-validation on the training data. The parameter k_{var} is selected from $\{1, 2, 3, \ldots, 10\}$ for best performance on validation data.

Surrogate Neural Network Setup

Based on the observation that kNN achieves outstanding accuracy but poor inference speed on our cyber-security problem, we chose it as the source anomaly detector for building the *surrogate neural network*. We parametrize the surrogate model as follows.

We fixed $k = 5$ in kNN used for auxiliary data set construction to get results comparable to the standalone kNN baseline anomaly detector. The auxiliary data set is constructed as described in Sect. 2.2.1 with the total number of auxiliary samples set to $m = n \cdot d$. (We discuss the impact of auxiliary set parametrization further in Sect. 2.5.)

ReLU activation function is used for all neurons (except for the input ones). The size of the batch is set always to 80. We use MSE as the loss function and train the

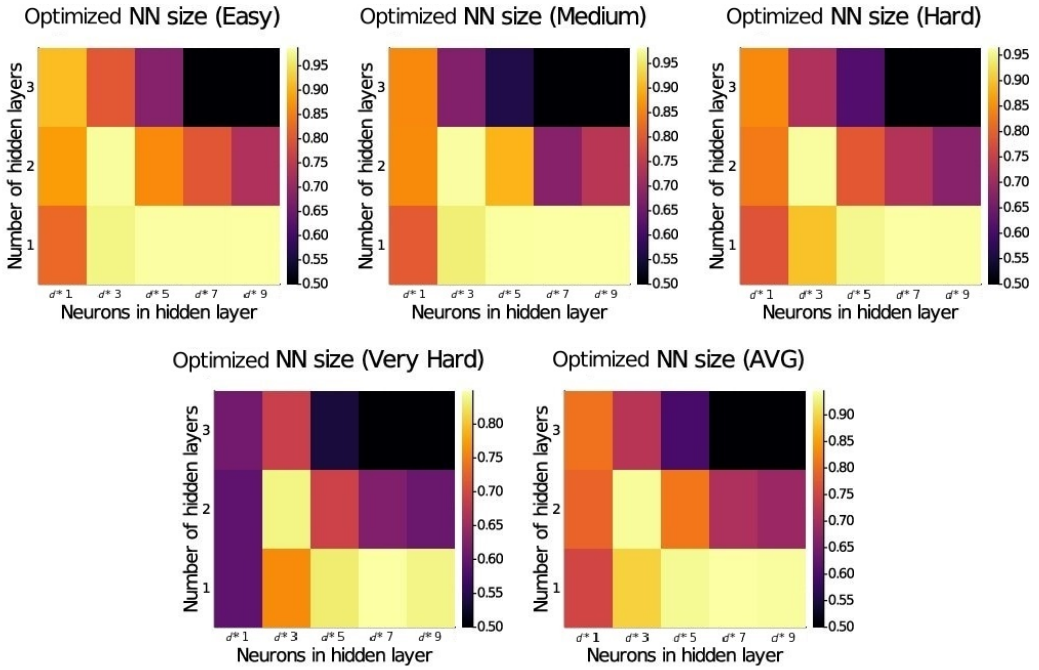


Figure 2.10: Illustrating the impact of surrogate neural network architecture to detection accuracy for the four different problem difficulties: *easy* , *medium* , *hard* and *very hard* (the 5th diagram shows the average). Brighter color depicts higher achieved AUC of ROC

network with *Adam* optimizer.

We opted for a simple meta-optimization of neural model parameters. For each problem difficulty (see Sect. 2.4.1) we train multiple models, to eventually retain the one with the best loss on validation data. The variation across training runs consists in: the number of hidden layers q varies between values $\{1, 2, 3\}$, hidden layer size p varies between values $\{1d, 3d, 5d, 7d, 9d\}$, random weight initialization is repeated $4\times$, the number of iterations is thresholded by six values between 15000 and 700000.

Figure 2.10 illustrates the impact of parameters q and p on the achieved accuracy.

2.4.3 Inference Speed Results

While addressing the cyber-security problem, our primary concern is the ability of surrogate models to improve the inference speed of the best performing baseline anomaly detector. Our secondary concern is the ability of a surrogate model to match the accuracy of its source anomaly detector.

We evaluated the inference speed of all baseline anomaly detectors and compare it to the speed of the surrogate detectors. Inference speed can notably depend on the problem dimensionality. The speed of some detectors—especially the nearest neighbor-based ones—also strongly depends on training data size. We illustrate this observation in Fig. 2.11. All measurements have been done on *medium* problem

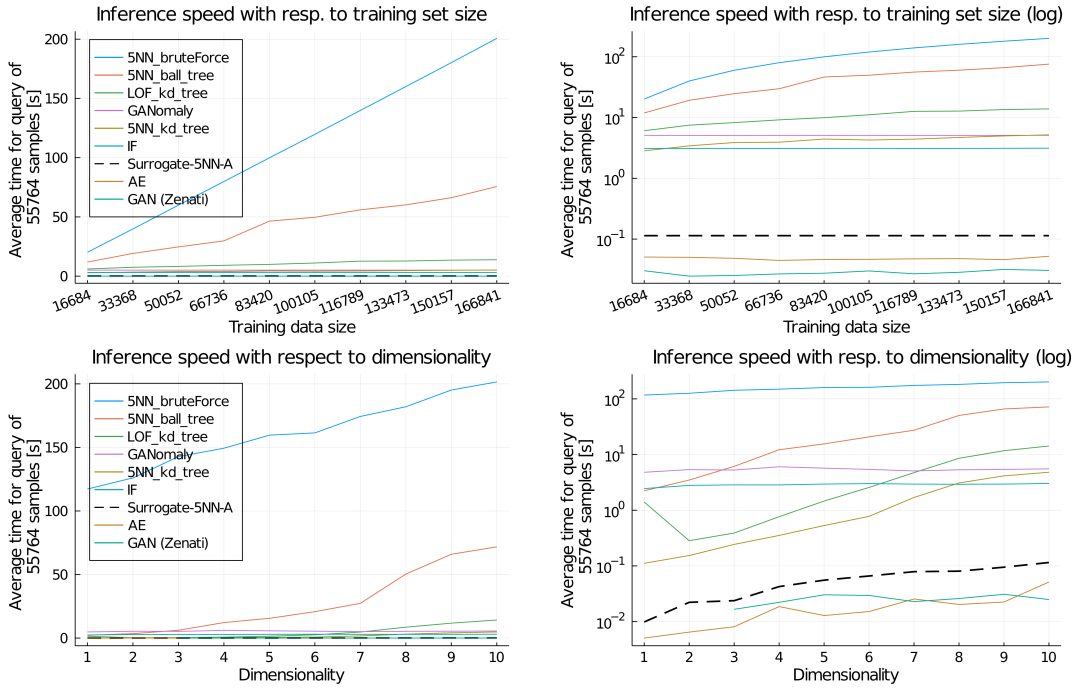


Figure 2.11: Dependence of anomaly detectors' inference speed on *training data size* (top) and *dimensionality* (bottom) in application phase. Note the speed-up achieved through the surrogate neural network with the auxiliary set (dashed line) over its source anomaly detector 5NN. Graphs in linear scale (left) and logarithmic scale (right)

difficulty. Graphs show the time needed to process all samples in the test set, i.e., 25% of all available data. Note that the graphs depict inference speed only, the training overhead is not included. The top plots show inference speed with respect to the size of the training set, the bottom plots show inference speed with respect to the dimensionality. The left plots are shown in linear scale, the right plots are shown in logarithmic scale.

As shown in Fig. 2.11, the inference speed of the neural network-based detectors is the least dependent on the size of training data. Dependence on the size of training data is most notable with nearest neighbor-based detectors, although the supporting structures in ball trees and k-d trees reduce this problem notably. Remark: in computer network analytics the number of samples to be processed online is several orders of magnitude higher than illustrated here.

The neural models including the proposed surrogate model in this test perform faster than the fastest nearest neighbor anomaly detector by at least an order of magnitude. Graphs suggest that this advantage will continue to grow with increasing dimensionality of the problem and even more so with growing training data set size.

Remark: Missing values in Fig. 2.11 are due to the limitation of the GAN implementation from [80].

2.4.4 Accuracy Results

	Easy	Med	Hard	VHard	*Avg*
5NN	96.0	94.9	96.2	**90.8**	***94.5***
Surrogate-5NN-U	94.0	90.3	79.4	65.9	*82.4*
Surrogate-5NN-A	**98.7**	**97.9**	**96.7**	83.2	*94.1*
AE	94.5	92.9	92.2	80.3	*90.0*
LOF	97.1	92.8	90.1	75.2	*88.8*
IF	95.9	94.1	90.4	79.3	*89.9*
Surrogate-IF-A	94.4	92.4	89.9	77.4	*88.5*
GAN (Zenati)	87.0	85.2	87.4	71.7	*82.8*
GANomaly	96.1	93.7	93.7	73.2	*89.2*
Parzen	95.3	94.8	94.1	79.9	*91.0*

Table 2.7: Comparison of best achieved accuracy (AUC of ROC, scaled to [0,100], averaged over 8 runs) by each anomaly detector on network security data set. Results grouped by problem difficulty. Suffix U marks surrogate model with uniform auxiliary set, A marks adaptive auxiliary set

As we expected, the best baseline accuracy on our cyber-security problem has been obtained from kNN anomaly detectors on all problem difficulties with $k = 5$, reaching the average AUC of 0.945. Accordingly, we focused on constructing and evaluating surrogate neural network detectors with 5NN as the source anomaly detector. However, we also replicate IF to demonstrate the versatility of the surrogate NN.

When evaluating the accuracy of surrogate anomaly detector models we primarily aim at verifying whether the surrogate model succeeds in matching the accuracy of its source anomaly detector. Improvement of accuracy is not expected although it can happen.

In Table 2.7, we primarily focus on comparing the surrogate neural network detectors (built from 5NN source detector with either uniform or adaptive auxiliary set) to the baseline 5NN detector. We then compare these to all the other baseline anomaly detectors. Each column in the table covers one problem difficulty, with the last column covering the average. Best achieved results are set in bold.

Assessing the accuracy of results over multiple data sets (difficulties) can be done in multiple ways [97]. We provide more detailed results in the form of confidence intervals [99] at the level of 95% in Fig. 2.12 to demonstrate the statistical significance of the results.

On the cyber-security problem, we observe an unexpectedly good performance of the surrogate neural network model with the adaptive auxiliary set. In all but *very hard* problem difficulty, the proposed method beats all other tested anomaly detectors including the source 5NN. 5NN remains notably best of all in the *very hard* problem difficulty. Here, the surrogate model still is the second best. Overall the surrogate model succeeds in retaining the average AUC of 0.941, which equals a drop of only 0.004 when compared to the average AUC 0.945 of 5NN detector (the average was computed over all problem difficulties).

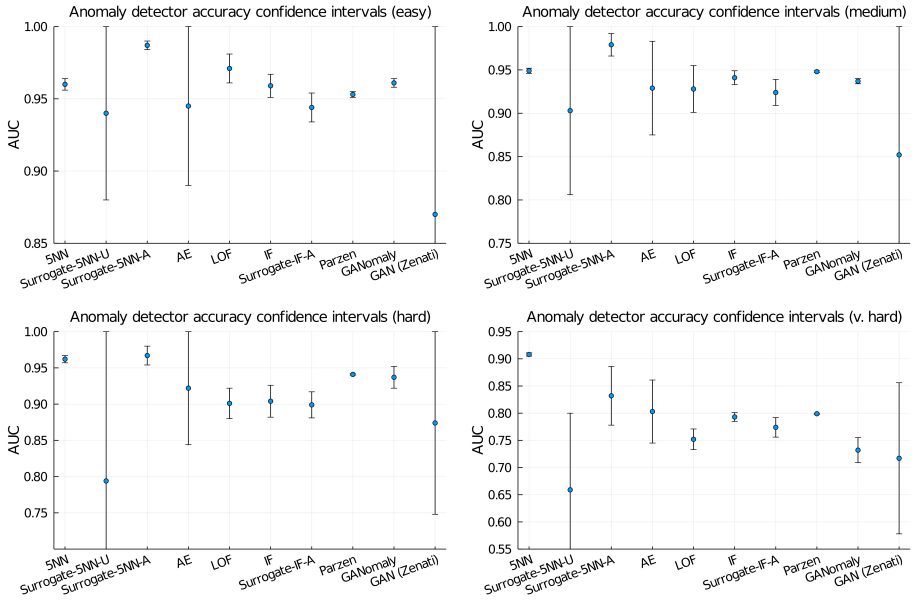


Figure 2.12: Comparing the accuracy achieved by the baseline and surrogate anomaly detectors on the network security problem. Confidence intervals for mean AUC of ROC at 95% level. Four problem difficulties (top left to bottom right): *easy*, *medium*, *hard*, *very hard*. Compare particularly the 5NN anomaly detector to the respective surrogate neural network with adaptive auxiliary set *Surrogate-5NN-A*

The success of the surrogate model with the adaptive auxiliary set is significant (cf. Fig. 2.12). The reasoning about why a surrogate model can surpass the accuracy of a source AD may relate to the more general question of why and when parametric models can surpass nonparametric ones. Our results on *easy* and *medium* problems appear consistent with the known observation that parametric models tend to generalize better, especially on simpler problems (cf., e.g., [100]).

In the other cases, we observed, as expected, an accuracy slightly below or on par with the source anomaly detectors. To illustrate this we have performed one additional experiment. We constructed a surrogate neural network with IF as the source detector and compared the two. The results are included in Table 2.7. In this case, the AUC of the surrogate model is on average 0.014 lower than the AUC of IF, with no observed improvement in any of the problem difficulties.

2.4.5 Discussion

In Fig. 2.13 we compare the source and surrogate detectors visually using a 2D projection with anomaly score heat map (projection to the first two PCA principal components).

The adaptive auxiliary set has enabled the considerable improvement of accuracy achieved by the neural model on a real-world data set that can be considered challenging. The low computational complexity in the application phase is not affected by switching to the adaptive approach. See Fig. 2.11 for time complexity analysis.

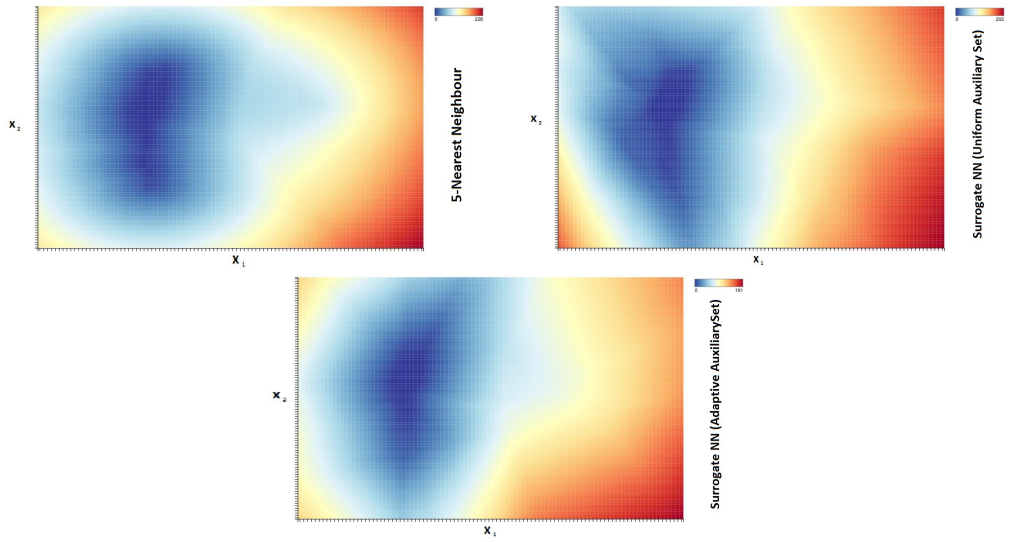


Figure 2.13: Heat-map illustration of anomaly scores induced on computer network security data set by (top to bottom): 5NN anomaly detector, surrogate neural network (with 5NN source detector) using the uniform auxiliary set, surrogate neural network (with 5NN source detector) using the adaptive auxiliary set. 2D projection to the first two principal components. Warmer color depicts a higher anomaly.

2.5 Robustness of Surrogate Detectors

Surrogate neural network-based detectors depend on a number of parameters. The neural model itself depends on all the standard parametrization common in neural networks, the analysis of which is not the subject of this work (see Sect. 2.4.2 for details of how parameters are set in this work).

The surrogate neural detector additionally depends on the properties of the auxiliary data set used in its training. In the following, we discuss their impact.

2.5.1 Auxiliary Set Size

An important question concerns the required size of the auxiliary data set. Clearly, the auxiliary set needs to be sufficiently large to replicate the space of anomaly scores induced by the source detectors. We cannot give a universal answer due to the variety of scenarios where surrogate neural network detectors can be applicable. Instead, we show the dependence of surrogate detector accuracy on the auxiliary set size in the case of our cyber-security problem. In Fig. 2.14, the x-axis shows the ratio of the auxiliary set size to the input training data size and the y-axis shows the achieved AUC. It can be seen that already a surprisingly small number of auxiliary samples (equal to 5% of the input data set size) proved sufficient to enable very good eventual accuracy on the network security data set (cf. Sect. 2.4.1).

To complement the picture, we include Fig. 2.15 to show the growth of surrogate model accuracy depending on the growing auxiliary set size on the benchmark

Abalone data set (cf. Sect. 2.3.1). Even in this case, it can be seen that only a fractional auxiliary set size when compared to the size of the input data is sufficient to achieve very good accuracy. Figure 2.15 also illustrates the efficiency of adaptive auxiliary sets (cf. Sect. 2.2.1) in contrast to uniform auxiliary sets (cf. Sect. 2.2.1).

2.5.2 Adaptive Auxiliary Set Parametrization

The adaptive auxiliary set construction procedure uses parameter k_{var} (variance multiplicative coefficient, see Sect. 2.2.1) which is essential to achieve optimal surrogate anomaly detector accuracy. The variance estimated from input data for the purpose of Parzen window sizing does not necessarily lead to the best possible auxiliary set. Therefore, for the auxiliary data set generation purpose, we multiply the estimated variance by the k_{var} coefficient, which needs to be optimized for each problem separately. The impact of various coefficient values is illustrated in Fig. 2.16 on the network security problem. Note that different levels of problem difficulty (cf. Sect. 2.4.1) may require different k_{var} values. In the experimental evaluation (cf. Sect. 2.4.4), however, we fixed one parameter only for all levels of difficulty.

2.5.3 Adaptive Auxiliary Set Efficiency

The auxiliary set construction procedure as described in Sect. 2.2.1 has been experimentally shown to provide results competitive with benchmark anomaly detectors (see Sect. 2.4.4). Let us discuss the question of whether there is still space for its improvement.

Arguably, for the purpose of AD, the most important regions in the modeled space may be those with a lower density of input samples. It appears that such regions would benefit from a higher density of generated auxiliary samples than regions

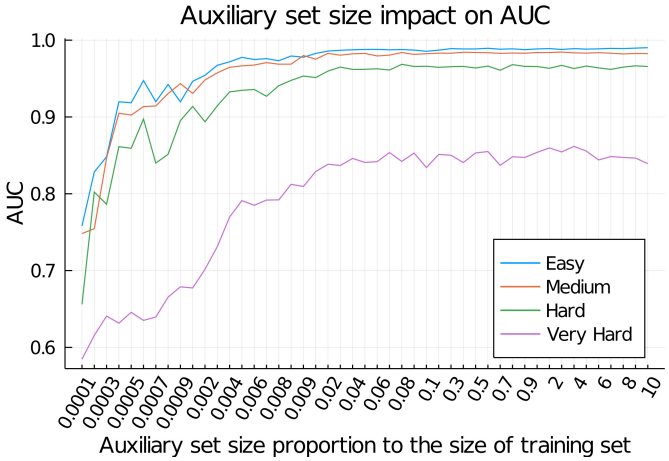


Figure 2.14: Accuracy of surrogate neural network detector depending on the adaptive auxiliary set size. Network security data set. Note that very small auxiliary set (about 5% of the input data size) can suffice to achieve best effect

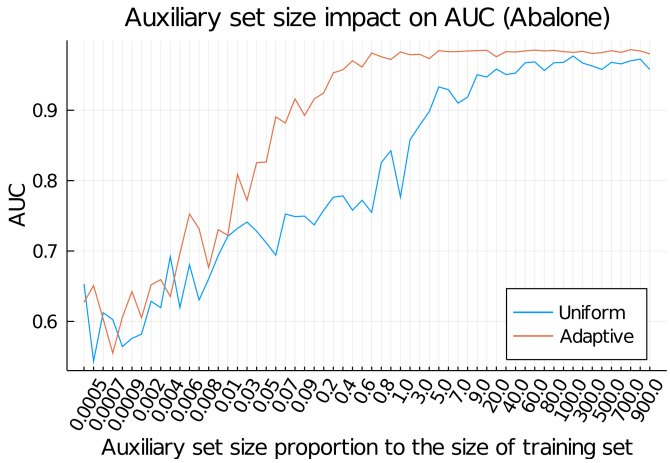


Figure 2.15: Accuracy of surrogate neural network detector depending on the auxiliary set size. Abalone data set, accuracy averaged over all levels of difficulty. Note the higher efficiency of the *adaptive* over the *uniform* auxiliary set

where the mass of input samples lies. The adaptive auxiliary set generating procedure, however, tends to produce the opposite. It generates more auxiliary samples and thus captures more details of the input distribution for areas of high input data density while areas of low density and singular samples get covered by fewer auxiliary samples.

Based on this observation, we implemented two modifications of the generation algorithm: 1. the relative number of auxiliary samples generated per Parzen window is made inversely proportional to the baseline anomaly score of the Parzen window center point (this effect is controlled through multiplication by constant σ where $\sigma = 1$ is equivalent to the original algorithm), 2. we also made the Parzen window width inversely proportional to the baseline anomaly score (this effect is controlled through multiplication by constant δ where $\delta = 1$ is equivalent to the original algorithm).

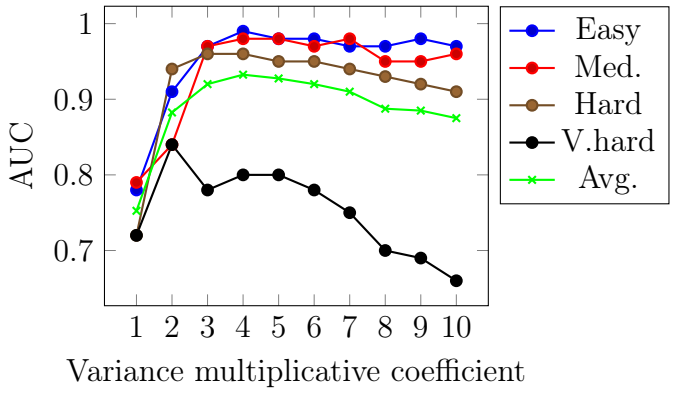


Figure 2.16: Surrogate neural network detector accuracy (AUC of ROC) depends on a parameter *variance multiplicative coefficient* k_{var} if the adaptive auxiliary set is used.

We performed a large number of tests for a grid of values $\sigma \in \{1, \ldots, 50\}$ and $\delta \in \{0.1, \ldots, 10\}$. In Fig. 2.17, we show the effect on four illustrative examples (compare to the baseline adaptive method result in Fig. 2.2). We do not include more details on the results because in all cases the resulting surrogate neural network detector accuracy dropped below the baseline adaptive method defined in Sect. 2.2.1. We

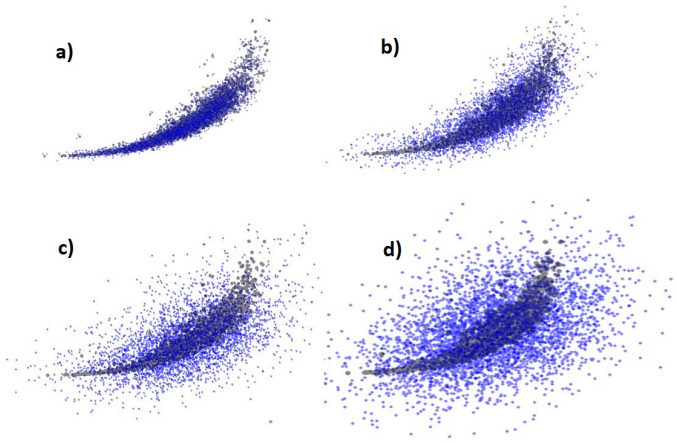


Figure 2.17: Modified adaptive auxiliary set generation procedure redistributes auxiliary samples from denser to less dense regions of the input space. Images illustrate modified auxiliary set variants from a) those with reduced window size in anomalous regions $\delta \ll 1$ to d) those with enlarged window size and weight in anomalous regions $\sigma > 1$ and $\delta \gg 1$

also tested the impact on auxiliary data set size efficiency (compare to Sect. 2.5.1). Again, no improvement has been reached. Therefore, the default adaptive auxiliary set generation procedure (cf. Sect. 2.2.1) remains the recommended option.

2.6 Discussion

The idea of constructing a surrogate neural network detector to replicate a source anomaly detector (which uses a non-neural model) has been motivated by the intention to improve inference speed in a large-scale industrial setting. We have shown the usefulness of the idea in a real cyber-security problem.

We have observed that the effect of introducing a surrogate neural network can have a positive impact also on the accuracy, although such an effect cannot be guaranteed. We have also observed that the size of the adaptive auxiliary set can be considerably smaller than the size of the input data set, without a negative impact. In general, it should be expected though that surrogate models achieve comparable or slightly worse accuracy than the source anomaly detector.

Note that the idea of cloning models has been also studied with the recent advances in deep neural networks and is referred to as *knowledge distillation* (see survey [101]). The main motivation is identical to ours: to simplify the model for better efficiency

and easier deployment. However, in practice, this technique is mostly focused on transferring knowledge from a large neural network to a simple neural network for classification tasks and typically for image data.

The surprising result that a shallow surrogate neural network could over-perform deep neural models (see Table 2.7) is a direct consequence of the fact that its source AD the kNN performed better than deep models on our cyber-security problem. Arguably, the high expressivity of deep models can be a disadvantage in a setting where generalization is very difficult (see particularly VHard in Table 2.7).

Note that for other problem areas, the idea of surrogate anomaly detectors can be decoupled into two separate parts: the construction of an auxiliary training set and the training of a model on top of it.

Once an auxiliary training set is constructed from a source anomaly detector, it is imaginable to train a non-neural model on top of it. We have not investigated such an option further.

Another interesting option is to utilize auxiliary sets for collecting information from multiple source anomaly detectors. We discuss this option in more detail in the following.

2.6.1 Fusion of Multiple Anomaly Detectors

Many different types of anomaly detector ensembles have been proposed in the literature [84] [86] [83] [85] to mitigate the limitations of individual detectors. Specifically in the area of our practical interest—in network security—ensembles of predictors are commonly applied [88].

The flexibility of the auxiliary set construction procedure trivially enables fusing outputs from multiple baseline detectors into a single auxiliary set. We envisage multiple implications as follows.

Mitigating the Problem of Incorrect Parametrization

Fusing outputs from different instances of the same type of detector can help smooth out the impact of potentially incorrect parametrization. This may become useful if there is uncertainty about which parameters to choose. In Tab. 2.8, we illustrate this effect by fusing a number of kNN detectors for multiple different values of k. Fusion in this case does not lead to the overall best accuracy but provides better accuracy than is the average of individual accuracies of the fused detectors (note the last two pairs of rows in the table).

Fusing Detectors to Reduce Ensemble Complexity

A practical use of detector fusion can expectably be the option to train a single surrogate neural network detector to replace an ensemble of detectors. Especially in

Table 2.8: Accuracy of fused surrogate anomaly detectors compared to individual surrogate detectors (AUC of ROC, scaled to [0,100]). Note that fusing various kNN detectors into a single surrogate anomaly detector can lead to better accuracy than is the average accuracy over the various standalone detectors. Evaluated with the *adaptive approach* on computer network data set.

Detector	Easy	Med	Hard	VHard
1-NN	68.29	75.79	66.55	62.57
3-NN	86.48	92.25	90.63	72.78
5-NN	98.72	97.89	96.66	83.24
7-NN	98.44	98.16	95.96	83.62
9-NN	98.63	98.22	95.65	83.57
avg 3NN,5NN,7NN	94.55	96.10	**94.42**	79.88
fused (3,5,7)-NN	**98.07**	**97.21**	94.38	**81.23**
avg 1NN,3NN...9NN	90.11	92.46	89.09	77.16
fused (1,3,5,7,9)NN	**97.86**	**94.39**	**94.08**	**79.47**

the case of large ensembles or ensembles of detectors of mixed types, the advantage can be not just the expected inference speed-up, but also the simplification of the overall deployed anomaly detection system.

The problem is that various source detectors may provide anomaly scores at different intervals or even unbounded. The prerequisite to their fusion therefore would be the normalization of the individual detectors' output. Normalization is possible in multiple ways. Platt scaling [102] can be considered. As a simpler option (inspired by the discussion in [103]) we propose the following.

Assuming we have a training set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where $\mathbf{x}_i \in \mathbb{R}^d$, $\forall i \in \{1, \ldots, n\}$ and the corresponding anomaly scores obtained from a generic anomaly detector $Y = \{y_1, y_2, \ldots, y_n\}$ where $y_i \in \mathbb{R}$, $\forall i \in \{1, \ldots, n\}$, the normalized anomaly score vector $\bar{Y}$ is obtained as

$$\bar{y_i} = \frac{L(y_i)}{n}, \quad \forall i \in \{1, \ldots, n\}$$

where

$$L(y) = \sum_{i=1}^{n} \begin{cases} 1, & \text{if } y_i < y \\ 0, & \text{otherwise} \end{cases}$$

Assuming a finite size of $\mathbf{X}$, the normalized anomaly score for sample $\mathbf{x}_i$ equals to the proportion of samples in $\mathbf{X}$ with lower anomaly scores than is the anomaly score of $\mathbf{x}_i$. The normalized scores are then from $[0, 1)$.

Fusing Detectors to Optimize Response

The most complex fusion that we envisage should enable the optimization of anomaly detection accuracy locally across the input sample space. It is based on the observation that principally different detection models are likely to have different strengths

and weaknesses, presumably in different parts of the input space. Let us assume that for each detector in a collection of various detectors, it is possible to estimate the confidence of its output for a specific sample. Then, using the normalization (see Sect. 2.6.1) an auxiliary set can be constructed from outputs of all the detectors, where the contribution of each detector to a single auxiliary sample is conditioned by the detector's sufficient confidence. In this way, various detectors from the collection would cover various parts of the auxiliary space, presumably leading to a more robust surrogate anomaly detector. The prerequisite here would be the ability to evaluate the confidence of each considered source detector. We refer to [103] for a solution to this problem.

2.7 Summary

Motivated by the needs of large-scale cyber-security systems we addressed the problem of anomaly detection inference speed. We proposed to construct surrogate neural network anomaly detectors to replace existing slow anomaly detectors or detector ensembles. We have shown that simple neural network formalism can be used to solve this problem. We have shown that it is possible to construct fast surrogate anomaly detectors without notable loss of accuracy. We have shown that the idea of surrogate anomaly detectors can also enable simplification of deployed anomaly detection systems, especially in the case of ensembles. We have observed that at least in network security the use of surrogate neural network detectors can occasionally improve the accuracy of the best baseline anomaly detectors.

Chapter 3

Surrogate Anomaly Detectors in Online Learning

3.1 Introduction to Online Learning

3.1.1 Motivation

In real-world scenarios, the models often run in dynamic environments that require constant adaptation of the model. An essential strategy is to enrich the model with up-to-date training data periodically or even continuously and thus reflect the novelty.

Let us describe a typical use case from the field of cyber-security. Assuming an initial state to be a well-trained anomaly detector that well recognizes all known standard behavior patterns of examined assets. However, after some time, the environment could change in many different ways, for example, the users might update or start using entirely new software, obtain a new class of device, etc... Consequently, the model needs to expand the knowledge base of what is meant to be normal behavior. Another more pragmatic use case is to update the model with false positives that are recognized on the next layers of the detection engine. Note that the detection engines are multilayer while the AD is performed on the first layers followed by classification engines and in some cases by human-based analysis and response.

3.1.2 Terminology

We emphasize the following terminology consistent with the literature (see Sect 3.1.5),

- Online predictions - the ML system is able to make predictions in real time (online).
- Offline predictions - complementary to online predictions. Typical use cases are recommendation systems in e.g. movie and music streaming platforms where

the operating cycle takes several hours. It is not suitable for applications like self-driving cars, face ID, and most security applications.

- Online learning - incorporating new data to update the model in real-time, typically in batch mode.
- Online training - learning from each incoming data point. This approach is rarely used in practice and even when it is used, the new model is not deployed with every single data observation.

In this chapter, we focus on online learning models that are capable of providing online predictions. Regarding online learning, we further use the following terms:

- Initialization (initial training) - First training of the model
- Update - a procedure of incorporating new data (batch) into the model.
- Batch - a subset of data utilized for a single update procedure

3.1.3 Expected Behavior of Online Learning Models

The tradeoff between stability and plasticity is an essential issue for online learning in general (see Sect. 3.1.5). In the anomaly detection context, the trade-off could be understood as the ability to include recent normal (non-anomalous) samples vs retaining the information learned in the past.

The optimal behavior of the model depends mainly on the application environment and the character of the training data. There are three main scenarios related to anomaly detection based on the character of the data:

1. Training data follow the same distribution. With more and more data, we can model the distribution with better precision.
2. The training distribution expands its support. Observations that have been normal are still normal, however, some observations that have been anomalous are becoming normal.
3. The training distribution shifts. Observations that have been normal might become anomalous and observations that have been anomalous might become normal.

In case number one and two, the optimal result of the model is equal to a model trained from scratch on the same data at the same time. In the terms of stability and plasticity, we see the ultimate challenge in achieving both. In other words, maximal stability in remembering once-trained samples and also maximal plasticity to react to newly trained data.

Note that constructing a model that maximizes only the stability which is a static, non-changing model is pointless and on the contrary, the maximum plasticity model can be supplied by training from scratch.

In case number three, the key role is to balance the mentioned trade-off between stability and plasticity thus forgetting the old training samples might be taken into account. In the industrial setup, it might be practical to train the model from scratch with the data from a recent period.

3.1.4 Online Learning Performance Reqirements

In practice, employing the models capable of online learning is often more complicated in comparison to offline learning. The emerging challenges and difficulties are depending on the utilized model. In general, the model-driven methods might be difficult to update with the new data but on the other hand, the data-driven models might be updated easily on the fly.

An example of a data-driven model could be the naive one-class kNN [7] which, in the naive form, could be updated instantly. However, it is limited by the growing memory consumption and it also suffers from increasing inference time (see Fig 2.11) with a continuously growing training set. As a result, such an approach is not sustainable in regular and real-world conditions from the speed and memory perspective. To clarify, we discuss the specific case without updating overhead, otherwise, there are techniques to mitigate the problem such as search trees [52, 53, 54] or subsampling approaches [104, 105] that require further computation with the update.

The more important factor, than the training overhead, is the cost of the resources when the algorithm is deployed which means the demands on the inference and the storage. Regarding the inference, the goal is to maximize the inference speed and minimize the size of the model because in the industrial setting, the detection is carried out in parallel, and the memory requirements of each detection node have an important role. Another factor is a memory requirement that is not directly related to the inference but it is necessary for the update procedure (e.g. storing historical training data).

3.1.5 Prior Art in Online Learning for Anomaly Detection

The field of anomaly detection and online learning were separately well studied in the last decades. However, the online learning approaches for anomaly detection are the very least researched and the surveys are lacking. As the target field combines two domains, we provide the related art for anomaly detection in Sect. 1.5 and an overview of online learning in the next paragraph.

Online learning also referred to as continual learning, lifelong learning, sequential learning, or incremental learning is well studied but mostly on classification tasks. The most recent survey [106] addresses the trade-off between stability and plasticity and provides a comprehensive experimental comparison of 11 state-of-the-art continual learning methods. A large overview of various approaches and algorithms is provided in [107] and similarly in [108] with a focus on dynamic environments for robotics, and [109] provides a comprehensive application-oriented study of catastrophic forgetting in deep neural networks. [110] surveys and examines current

evaluation methodologies and provides a case study on a common practice.

Online Learning Methods for Anomaly Detection

OLINDDA (OnLIne Novelty and Drift Detection Algorithm) [111] uses k-means clustering techniques to handle non-stationary data distributions. The idea is to group the outlining data and create new clusters to adapt to the novel concept rather than to detect the anomalies.

Centroid-based anomaly detection with a finite sliding window of training data is utilized in [112]. Once the centroids are found, the anomaly score of a sample is examined with the euclidian distance to the centroid. The advantage is a trivial and extremely time-efficient update of the centroids computed as a weighted mean of the former centroid and the new sample.

The histogram-based approach was utilized in LODA [33]. The algorithm projects the data to a number of one-dimensional histograms while each represents the density in the projection vector and forms a weak detector. Then the resulting detection is performed as an aggregation of the histograms (weak detectors). The benefit is a simple and efficient update of the histograms.

The Half-Space trees (HS-Trees) [113] is an analogy to isolation forest for evolving data streams. The decision rules in the tree nodes are generated randomly, and thus the model can be initialized without any data and trained gradually. However, there is an assumption that the data is scaled to [0, 1] which could be a minor complication in some real-world scenarios.

Online learning and anomaly detection was successfully applied to time-series processing in [114] for sensor systems-based data and similarly in [115] for data streams from various sources. Statistical techniques for online anomaly detection in data centers are described in [116]. A framework for system log processing using operators' feedback is proposed in [117]. The online learning AD is also applied to image processing in [118] to recognize anomalous behavior in a crowd of people while adapting to changing environment.

In addition to the specific online learning methods listed above, some of the commonly used AD models are almost natively capable of online learning. Data-driven models are straightforward to update with new data. For example, the naive version of kNN can be easily updated however, kNN is typically used with supporting structures such as kd-tree which are also updated with some minor resources.

3.1.6 Prior Art Relevance for Cyber-security

Our goal is to provide a high-efficient and high-accuracy solution to the field of computer security. In this section, we discuss and define the target properties of the algorithm.

Our use case is mainly bullet point number two in Sect. 3.1.3 and thus the model

should be updated without forgetting. In other words, Once the data is used for initialization or update, the obtained information should be stored in the model over all future updates.

Another limiting factor is the memory requirement when the model is deployed and stored. The model must be capable to embrace information from large and continuously growing data while the memory storage options are typically limited. The deployment efficacy and inference speed are also crucial for industrial applications.

Finally and most importantly, the accuracy of the model is an important factor. The goal of the online learning algorithm is to minimize the accuracy drop caused by iterative learning. In other words, the accuracy of the online model should be comparable to the corresponding model trained offline (all at once).

The models listed in Sect. 3.1.5 are mostly capable of efficient update procedures and are relatively memory efficient mostly due to simplicity and all of them are designed to perform the update procedures without forgetting. However, the expected accuracy of those models is not much promising. In the best scenario, the resulting accuracy should be equal to the offline approach which is the regular AD technique. Clustering-based and centroid-based techniques are well-known in the AD context for their simplicity but they are not expected to provide sufficient performance and thus are even not considered in the AD benchmarks e.g. [34, 35]. Similarly, the HS-Trees were not considered in standard AD benchmarks and a little more attention deserved LODA that was for example considered in [34].

In general, the listed online models are expected to provide less satisfactory accuracy in comparison to the offline AD models. Note that each online model can also be used offline and compete in offline AD. In a view of Chapt. 2, where we proposed an offline AD model SNN that outperforms its competitors and is extremely beneficial for industrial applications, it would be highly desirable to design an online variant of SNN .

3.2 Towards Online Surrogate Neural Models

The SNN is originally an efficient detector capable of making predictions online, and it was shown that in addition to its inference speed, its accuracy is competitive to the relevant state-of-the-art. (See [90, 89, 91] and Chapt.2) However, the current form does not adequately address the problem of online learning. The opportunity of updating the model or even online learning is missing and thus if the baseline method is forced to update, the only possible option is to train a new model from scratch. Besides the common disadvantages, training from scratch is problematic for real-world applications because the training telemetry is usually too expensive to be stored over a longer period for repetitive training.

3.2.1 Ideal Properties of the Algorithm

Optimally, we aim at discovering an updating paradigm that, in accordance with Sect. 3.1.6, minimizes the need of storing historical data and computational complexity while retaining the maximal information from the history in the model. The accuracy and inference speed should be similar to the offline SNN which is satisfactory.

Hypothetically, in the best scenario, the update process could be simple and straightforward as shown in Fig. 3.1. In this chapter, we need to resolve two fundamental questions:

1. It is possible to update the neural model directly with the update batch only and reach the equivalent accuracy as with the offline SNN ?
2. If not, it is possible to at least approximate the model satisfactorily? What information needs to be available?

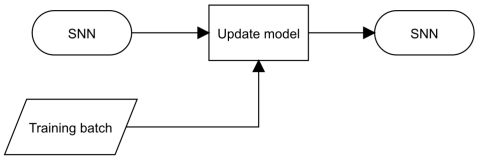


Figure 3.1: Flowchart of optimal model update process

3.3 Training and Auxiliary Data Lifecycle in Online Setting

The challenge of defining online SNN consists in reflecting the fact that it may not be possible to access all (especially historical) training and auxiliary data at the time of model updates. In this section, we discuss the impact of partial inaccessibility to both sets on the model's performance.

3.3.1 Training Set Memorization vs Approximation

SNN is very difficult to update with full precision without all data (historical and update batch) at the same time. Updating the model with new data only is an open problem and seems to only be solvable approximately.

The formal definition of the problem follows:

Let us have an initial training data set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}, \mathbf{x}_i \in \mathbb{R}^d, \forall i \in \{1, .., n\}$ and a labeled data set $\mathbf{A}$ (AUX in our case) of m samples where $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}$,

$\mathbf{a}_i \in \mathbb{R}^d$, $\forall i \in \{1, .., m\}$ and Y the vector of its labels that are respective anomaly scores computed with a source-detector with respect to $\mathbf{X}$, where $Y = \{y_1, y_2, ..., y_m\}$, $y_i \in \mathbb{R}$), $\forall i \in \{1, .., m\}$.

The goal is to update the anomaly scores Y by including a newly coming data $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_o\}$, $\mathbf{z}_i \in \mathbb{R}^d$, $\forall i \in \{1, .., o\}$ such that Y are anomaly scores of $\mathbf{A}$ with respect to $\mathbf{X} \cup \mathbf{Z}$ without utilizing $\mathbf{X}$.

For a general source-detector, the problem is not solvable, even though the problem is dependent on the choice of the source-detector and there might be edge cases where the problem can be solved, for example, for one-class kNN where $k = 1$. Let us explain the problem for one-class kNN where $k = 2$. Note that we select kNN as it proved its performance in Chapt. 2 and $k = 2$ for sake of the simplicity in the demonstration.

Let us assume the following example: the SNN is trained with two training samples (see Fig. 3.2) and the anomaly function is stored in the SNN model. From this point, the training data are not required and thus are deleted. Despite the fact that the model provides an anomaly score for any possible sample, it is very difficult to reconstruct the original training set. Next, we would like to update the mode with a new sample (see Fig. 3.3).

An illustration of the ambiguity in updating the model without the historical data is shown in Fig. 3.4. The goal is to provide an accurate label for a given sample (AUX sample in SNN) based on the anomaly score function provided by SNN and the recently added training sample. The figure demonstrates two possible layouts of the former (deleted) training data, both resulting in the same label from the SNN thus the procedure's input is identical for two distinct cases with two distinct expected outputs when assuming ground truth knowledge of the deleted samples. The figure also demonstrates that in one case the ground truth label is affected by the update sample but in the other case is not.

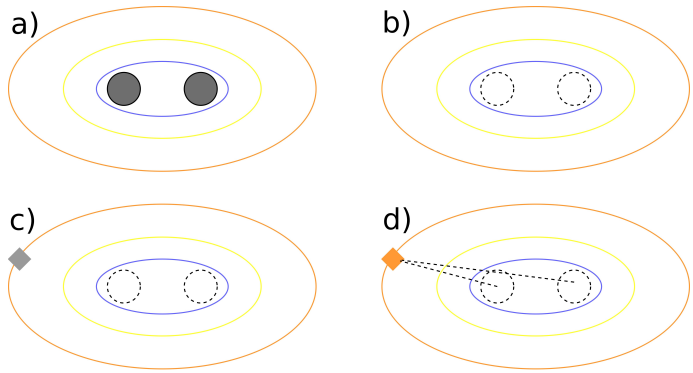


Figure 3.2: Schema of SNN induced with one-class kNN (2NN). a): Two training samples (black) and anomaly function represented by the colored contours. b): Training samples are deleted. c): Sample to be labeled d): Inference carried out with SNN - Anomaly score corresponds to the mean distance from the former training set.

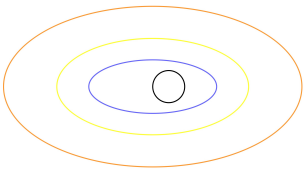


Figure 3.3: Example of possible input for the update of SNN - new sample represented by a black circle

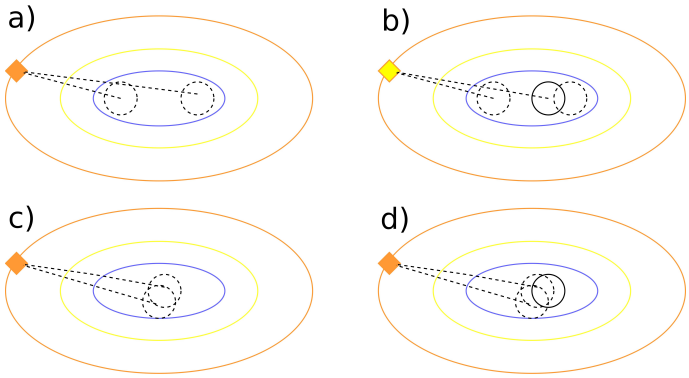


Figure 3.4: Illustration of the ambiguity in updating the SNN without the historical data. On the left side (see a,c), there are two possible layouts of deleted training data resulting in identical anomaly score prediction of the square sample. On the right side (see b,d), distinct ground truth anomaly scores are depicted Note that the update of the anomaly function (contours) is not addressed in this figure.

To conclude this section, we have shown (see Figs. 3.2, 3.3, and 3.4) that either all historical data must be stored to achieve loss-less labeling of AUX or some level of approximation must be accepted to become memory sustainable.

3.3.2 Auxiliary Set Memorization Options

In this section, we will discuss several options for obtaining an updated auxiliary data set and updating the SNN. We will demonstrate how the various update procedures affect the success of the updated surrogate AD model. We address the following questions: Is the AUX necessary? Is it possible to generate AUX from the update batch only?

For each option, we also provide illustrative experiments to support the intuition of the reader. The idea is simple, we train the model with the initial data to obtain the initial state. Then the update experiments with various AUX are provided. The initial state is identical for all experiments (Fig. 3.6 - offline SNN).

Let us initialize the experiment, we utilize a toy data set shown in Fig. 3.5 that contains training data and a handcrafted update batch consisting of a compact cluster of 5 samples (we use 5NN). First, we train SNN with the training data. The resulting anomaly score heatmap can be compared with the source-detector kNN in

Fig.3.6. Next, let us construct the heatmap for the updated data set with the source-detector (see Fig. 3.7) and consider it for reference regarding the expected output of the neural model update procedure. Further, we provide several approaches to update SNN aiming at the reference output.

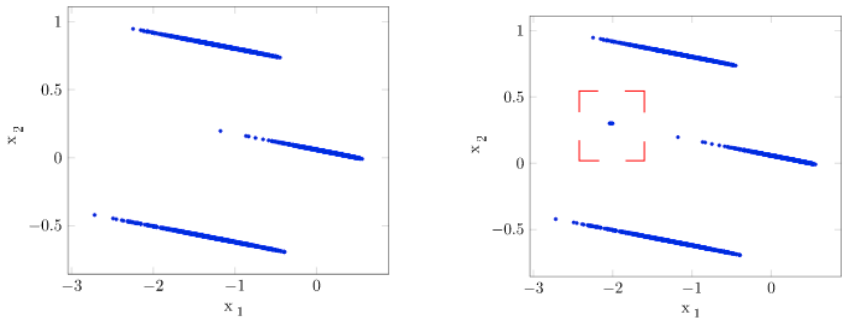


Figure 3.5: Illustrative 2D data set based on Abalone data set. Training data are shown on the left side and the updated data on the right side. The data set is updated with a handcrafted batch of 5 overlaying points.

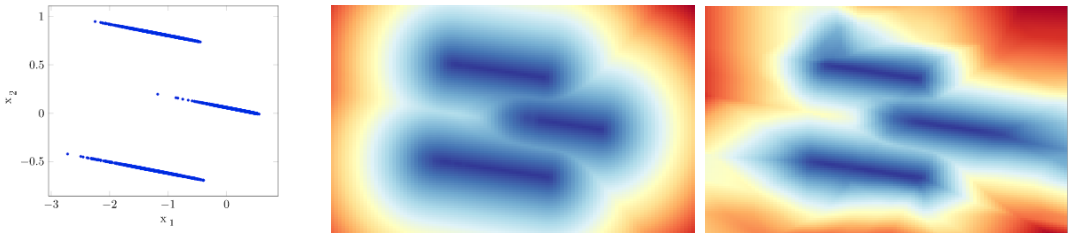


Figure 3.6: Original data set (left) and anomaly score heatmaps inferred with kNN(middle) and offline SNN (right).

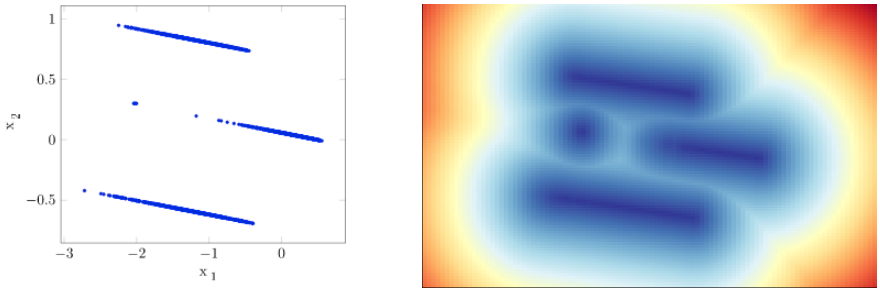


Figure 3.7: Updated dataset and anomaly score heatmap inferred with kNN. Since the kNN is used as a source-detector, the presented heat map depicts an expected output of SNN after the update procedure.

Approach 1

Updating SNN with the update batch only (without AUX set) is a very narrow idea, however, it is worth discussing. The idea is to update the SNN with the update batch instead of the AUX while the anomaly scores of the samples are zero or close to zero as the training and update data are not anomalous.

Examples of the resulting heatmaps of the update procedure are demonstrated in Fig. 3.8. The update procedure is unstable, resulting in various outcomes, so we provide the two most representative examples. In the first case (Fig. 3.8 - left), the anomaly score function really reaches close-to-zero values for the newly added points but the expected pattern of the close neighborhood is not achieved. In addition, the anomaly function is corrupted in terms of its pattern, and also the absolute values of the function are decreased. Moreover, the repetitive training with only zero (or close to zero) labels will gradually corrupt the model to only predict low values. Anyway, the degradation could even be achieved with only one standard update iteration as shown in the second example (Fig. 3.8 - right).

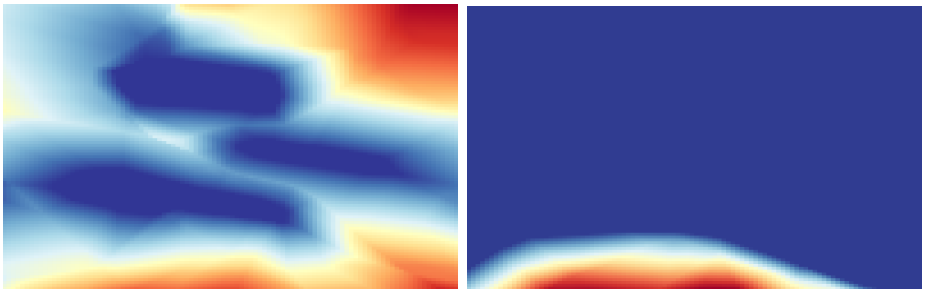


Figure 3.8: Approach 1 – Example of two distinct update attempts of SNN without the AUX data set.

Approach 2

The next approach is to construct the AUX set based on the update batch only to cover its close neighborhood in order to obtain a more relevant pattern. The distribution of the AUX samples and the resulting heatmap is given in Fig. 3.9. The anomaly scores of the AUX samples are computed with all training samples. The resulting heatmap is similar to the case shown in Fig. 3.8 but with a more remarkable degradation of the pattern outside the sampled area.

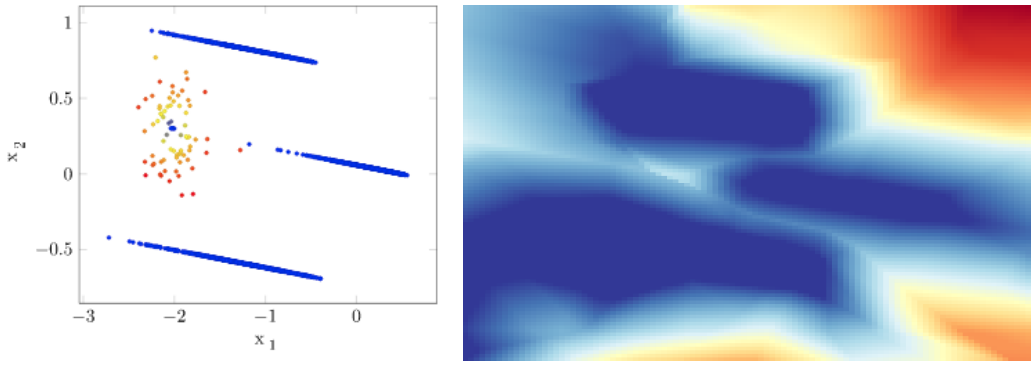


Figure 3.9: Approach 2 - Example of update process via AUX dataset generated in a close neighborhood of the new data. Left: training and update data set (blue), AUX (colored with anomaly score); Right: resulting SNN anomaly score heatmap.

Approach 3

This approach is similar to approach 2 but the AUX coverage is constructed in a larger area in order to capture the expected pattern of the source-detector. The demonstration of the AUX coverage and resulting heatmap is given in Fig. 3.10. The heatmap captures the expected pattern in the covered area but the pattern is collapsed in the uncovered area.

Although this approach provides the best outcome yet, it suffers from degradation due to a lack of coverage in the area of the (former) training data. To overcome this issue the next approaches will have to take into account both update and former training data.

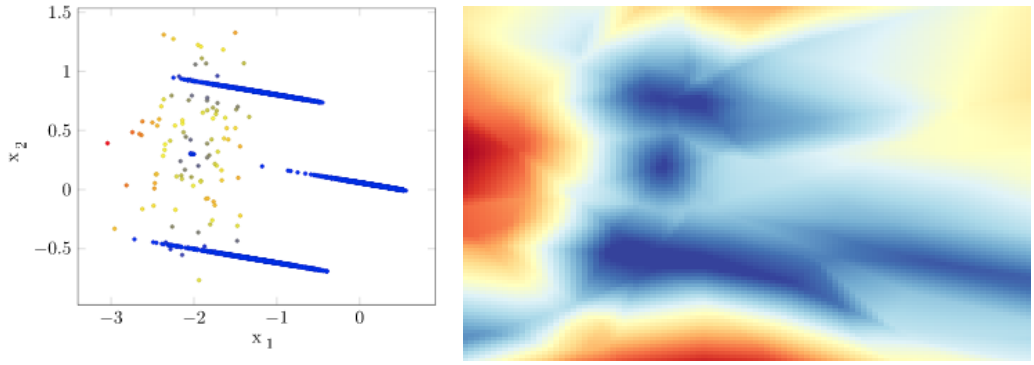


Figure 3.10: Approach 3 - Example of update process with AUX generating on the larger neighborhood. Left: training and update data set (blue), AUX (colored with anomaly score); Right: resulting SNN anomaly score heatmap.

Approach 4

This approach utilizes the coverage for all training and update data in order to provide sufficient local coverage nearby the update batch and to prevent degradation nearby the training data. We utilize *uniform auxiliary data set* construction (see Sect 2.2.1) and increase the number of AUX samples. The demonstration is provided in Fig. 3.11; according to the heatmap, the resulting model slightly follows the expected pattern with some differences mainly at the border of the coverage.

Note that the position of the AUX samples is generated from all training data and thus more information is required than the only update batch. In this case, however, the memory demand is acceptable as the uniform approach generates within the hyper-block thus the min and max vector need to be stored. On the other hand, the construction via *uniform auxiliary data set* is suboptimal as has been shown in Chapt. 2. One of the weaknesses is that it suffers from the course of dimensionality, which does not manifest in the 2D demonstration.

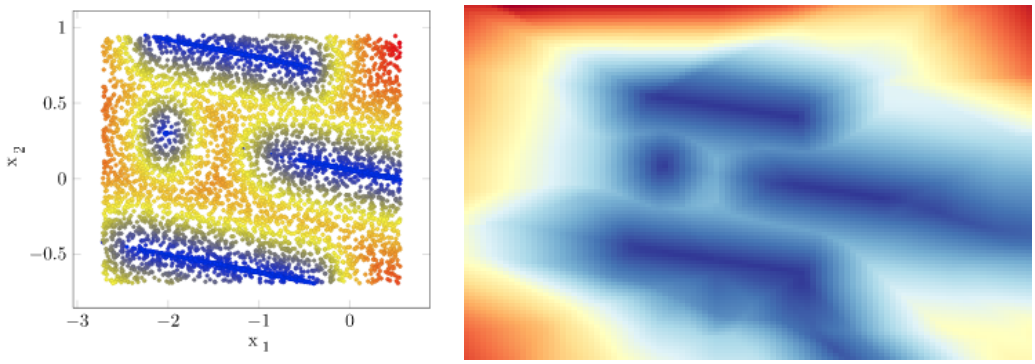


Figure 3.11: Approach 4 example of update process with AUX generated with the *uniform auxiliary data set* and all training data. Left: training and update data set (blue), AUX (colored with anomaly score); Right: resulting SNN anomaly score heatmap.

Approach 5

We follow the idea from *approach 4* with the only difference that we use the *adaptive auxiliary data set* (see Sect. 2.2.1). The process is demonstrated in Fig. 3.12 and it apparently provides the best setup of the update process as the obtained pattern very closely fulfills the expected pattern (see Fig.3.7). Note that we need to store all training data to achieve optimal results.

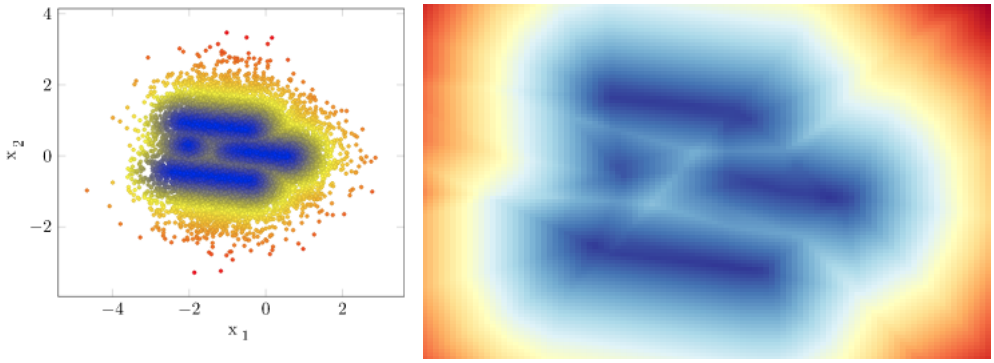


Figure 3.12: Approach 5 example of update process with AUX generated with the *adaptive auxiliary data set* and all training data. Left: training and update data set (blue), AUX (colored with anomaly score); Right: resulting SNN anomaly score heatmap.

Conclusion on AUX update approach experiments

We discussed several variants for constructing the AUX data set used for SNN update. It has been shown that SNN cannot be successfully updated without AUX or with AUX constructed from the update batch only and thus the former training data needs to be used. We also discussed the preference of approach 5 over approach 4.

At this point, we demonstrated a simplified design of the update procedure with a promising behavior (approach 5). However, this design needs to memorize all the historical data in each update process. This is an acceptable status in an early stage of development, but unacceptable for any real application. Note that we utilize all samples (training and update) to compute the labels of the AUX in this section.

3.4 Towards Efficient Model Update Strategy

In Sect. 3.3.2, we have depicted a functional baseline updating procedure that, however, is extremely inefficient. In this section, we further analyze the opportunities for optimization regarding the target properties, especially memory. We propose several possible approaches that are more efficient due to approximation and we discover closely how the approximation affects the final precision.

3.4.1 Problem Overview

In the baseline algorithm, there are two data sets to be stored: training data and AUX data. Theoretically, the AUX could be generated from the training data on demand and thus need not be stored, however, in the industrial setup, the AUX is significantly smaller than the training set and thus this is not a good way to go. Instead, we rather focus on disposing of the training set first and secondly on more efficient storing of AUX.

The memory spent on the training set can be saved if we accept the approximation as shown in Sect. 3.3.1. In Sect. 3.4.2, we propose a simple approximation methodology, that takes the advantage of the trained neural model to infer the anomaly score corresponding to the historical training data.

Storing the AUX is less heavy than the training set, however, its storage can still be optimized. We discuss the problem closely in Sect. 3.4.3.

3.4.2 Approximation of Auxiliary Set Labels - Options

As described above, accepting the approximation of the labels is an essential assumption towards efficiency. Let us recall the formal definition of the labeling problem described in 3.3.1. In short, there is a set $\mathbf{A}$ with labels representing anomaly scores Y generated by the source-detector with respect to $\mathbf{X}$ (training set). Once $\mathbf{X}$ was utilized for computing anomaly scores, it is not stored further. After that, the training set is enriched with new data $\mathbf{Z}$, and the goal is to update the scores Y to correspond to the distances to $\mathbf{X} \cup \mathbf{Z}$.

We propose a simple approximation to compute additional scores $\bar{Y}$ with respect to $\mathbf{Z}$. Thus for each sample, $\mathbf{A}$ exists one anomaly score from Y corresponding to the historical data and the other $\bar{Y}$ representing the anomaly score corresponding to the current batch. The resulting anomaly score is approximated as a minimum of the two scores. Note that the approximated score is always equal to or higher than the non-approximated score would be.

More formally, assuming anomalyScore$(\mathbf{A}, \mathbf{X})$ computed with the source-detector as the anomaly of $\mathbf{A}$ based on training set $\mathbf{X}$, the vector of scores Y^* approximating anomalyScore$(\mathbf{A}, (\mathbf{X} \cup \mathbf{Z}))$ is computed as :

$$y_i^* = \min(y_i, \bar{y_i}) \mid y_i^* \in Y^*, y_i \in Y, \bar{y_i} \in \bar{Y} \mid i \in \{1, ..., m\}; \tag{3.1}$$

where m is number of samples in $\mathbf{A}$, Y is anomalyScore$(\mathbf{A}, \mathbf{X})$ and $\bar{Y}$ is anomalyScore$(\mathbf{A}, \mathbf{Z})$.

Let us discuss the potential loss of the information a bit more for kNN as the source-detector. In the case of $k = 1$, the method is lossless, and another way round, if k was higher than the size of the update batch, it would not be applicable. Our expectation is that the proportion of k and the size of the update batch will affect the accuracy of the approximation. The field we aim at (computer network traffic) perfectly fits this condition because there are a lot of data available and previous experiments (see Sect. 2.4) proved relatively lower k to have optimal performance.

3.4.3 Approximation of Auxiliary Set Distribution - Options

We already demonstrated that AUX is necessary for the updating procedure, however, it can be handled in various ways and enable more memory-efficient approaches. First of all, let us clarify the problem. We assume, that for each update procedure, we need AUX coverage nearby both the historical training data and the current

update batch data. Creating AUX for the update batch is straightforward because the update batch is always available. The goal, therefore, is constructing AUX for the training data seen in the past while focusing on memory consumption.

Baseline

We remind the *naive construction*, which is the simplest approach based on the storage of all the historical training data and the construction of the AUX from the stored data.

Full AUX storage is a coextending method to *naive construction*. The idea is to store each AUX sample once generated. It enables clearing off the historical training data and possible benefits from the significantly smaller AUX in comparison to the training data. Although this approach might be much more efficient than the *naive construction* in the industrial setup, it still suffers from growing memory consumption with each update procedure.

Parametric Approximation

The distribution of AUX samples in the space is more important than the exact location of each sample. Note that AUX is generated randomly in a specific distribution when created from the training samples. Thus the AUX can theoretically be generated with a very good functionality if we know its distribution well. In addition, storing the information about its distribution is significantly less memory demanding.

The idea is to is to extract the distribution information from the current AUX after each training period, store it compressed, and re-create it again for the next training.

Compressing the AUX into a distribution model after the training iteration is an efficient form of storing without a loss of functionality for the main algorithm. We propose to use the Gaussian mixture model [18] due to its relative simplicity and expressivity in contrast to deep generative models and others. The advantages of GM model are mainly the low number of hyper-parameters and also the robustness to the hyper-parameters and relatively fast training using the EM algorithm.

3.4.4 Auxiliary Set Algorithmic (Re)construction

In the previous sections, we sum up several strategies for manipulating both the AUX samples and their labels. The strategies mainly differ in the level of approximation and storage requirements. In this section, we propose several specific algorithms the output of which is the AUX with labels to train the NN model.

Algorithm A: Baseline algorithm that stores all the available data and thus provides theoretically the best accuracy. This is a storage extreme model that memorizes all data in each operation, and thus no approximation is used for computing AUX. This model, due to no loss of information, should provide theoretically the best performance.

Algorithm A The baseline

```
1: global variables:
2:    aux, data-history
3: external functions:
4:    srcDetector(train, predict), generateAux(data)
5: procedure UPDATE(batch)
6:      batchAux = generateAux(batch)
7:      aux = aux ∪ batchAux
8:      dataHistory = dataHistory ∪ batch
9:      labels = srcDetector(dataHistory, aux)
10:     return aux, labels
```

Algorithm B: This algorithm is specifically developed only for kNN as the source-detector. The idea is to store the AUX with anomaly scores instead of the training data which is a significantly more memory-efficient approach. To minimize the negative effect of the missing historical training set, we propose to memorize k nearest distances for each AUX sample, instead of the anomaly score thus the update procedure is more precise. Although remembering individual distances is demanding, this algorithm is more efficient than algorithm A.

There also is a need for an approximation in a minor case. When the new AUX is generated near batch data, we approximate its anomaly score with respect to the historical training data with the neural model. However, we cannot reconstruct k distances corresponding to nearest neighbors thus we assume all the distances equal to the anomaly score.

Algorithm B AUX with distances

```
1: global variables:
2:    aux, distances
3: external functions:
4:    knnDistanceDistances(train, predict), generateAux(data), SNN.predict(data)
5: procedure UPDATE(batch)
6:    COMMENT: Compute distances for old AUX
7:    auxToNewBatchDist = knnDistanceDistances(batch, aux)
8:    distances = select k least distances (distances, auxToNewBatchDist)
9:
10:   COMMENT: Compute AUX near batch with distances
11:   batchAux = generateAux(batch)
12:   batchDistancesHistory = SNN.predict(batchAux)        ▷ k identical values
      (approximation)
13:   batchDistancesCurrent = select k least distances (batch, batchAux)
14:   batchDistances = select k least distances (batchDistancesCurrent, batchDis-
      tancesHistory)
15:
16:   COMMENT: Merge AUX and return training data for NN
17:   aux = aux ∪ batchAux
18:   distances = distances ∪ batchDistances
19:   return aux, mean(distances)
```

Algorithm C: This algorithm evolves from algorithm B by storing only the anomaly score instead of distances. Here, the approximation of two anomaly scores by the minimum is utilized (see Sect.3.4.2) instead of precise computations. As a result, we can closely discover how the approximation of labels can affect the accuracy by comparing algorithms C and B.

Algorithm C AUX with anomaly score

```
1: global variables:
2:    aux, labels
3: external functions:
4:    srcDetector(train, predict), generateAux(data), SNN.predict(data)
5: procedure UPDATE(batch)
6:    COMMENT: Construct batch AUX
7:    batchAux = generateAux(batch)
8:    batchLabelsCurrent = srcDetector(batch, batchAux)
9:    batchLabelsHistory = SNN.predict(batchAux)
10:   batchLabels = min (batchLabelsCurrent, batchLabelsHistory)
11:
12:   COMMENT: Update labels for old AUX
13:   auxToBatchLabels = srcDetector(batch, aux)
14:   labels = min (labels, auxToBatchLabels)
15:
16:   COMMENT: Merge and return
17:   aux = aux ∪ batchAux
18:   labels = labels ∪ batchLabels
19:   return aux, labels
```

Algorithm D: The next step is naturally focused on approaches that are even more memory efficient and thus operate without storing specific AUX or training data. The idea behind this is simple, let us assume the information needed to be reconstructed, for the historical data, is AUX coverage and corresponding labels while the labels can be predicted by the current neural model. When focused on the AUX data set, we remind that it always is generated randomly in a specific distribution and thus we can generate the samples on demand if we know the distribution. We utilize the GM model to represent the distribution as explained in 3.4.3.

Algorithm D AUX via GM model

```
1: global variables:
2:    GMModel
3: external functions:
4:    srcDetector(train, predict), generateAux(data), SNN.predict(data)
5: procedure UPDATE(batch)
6:    COMMENT: Construct batch AUX
7:    batchAux = generateAux(batch)
8:    batchLabelsCurrent = srcDetector(batch, batchAux)
9:    batchLabelsHistory = SNN.predict(batchAux)
10:   batchLabels = min (batchLabelsCurrent, batchLabelsHistory)
11:
12:   COMMENT: Update labels for old AUX
13:   aux = generate(GMModel)
14:   auxToBatchLabels = srcDetector(batch, aux)
15:   labels =SNN.predict(aux)
16:   labels = min (labels, auxToBatchLabels)
17:
18:   COMMENT: Merge and return
19:   aux = aux ∪ batchAux
20:   labels = labels ∪ batchLabels
21:   GMModel = fitGM(aux)
22:   return aux, labels
```

Algorithm E: To provide a more comprehensive study, we also include an algorithm that is derived from algorithm D, however, instead of the GM model, it only generates the data from a normal distribution based on *mean* and *variance*. As a result, the accuracy difference between algorithms D and E will reflect the value of the GM model.

Algorithm selection: We have evaluated the algorithms experimentally under various conditions (see Tab. 3.1) and discovered, that the efficient approximation-based *algorithm D* provides sufficient accuracy, comparable with the baseline while dramatically reducing the storage requirements. Thus, we select *algorithm D* as the potentially best approach and examine it further in the following text.

3.5 Online Learning Method Proposal

The methodology we propose aims at online learning of the SNN model in batch mode, taking use of Algorithm D defined in Sect. 3.4.4. As discussed in Sect. 3.4 the algorithm is designed to maximize accuracy and to minimize computational and memory demands. Two types of routines are utilized in the algorithm. First, the model is initialized and trained with the first batch or data set, then the online learning update routines are applied.

3.5.1 Initialization and First Training

The SNN model is initialized in a very similar manner as the offline SNN is performed (for detailed explanation see Sect. 2.2). In short, the initial training data is utilized to construct the first AUX and to obtain the corresponding labels and then the NN model is trained. The only difference from the offline routine is the computation of the GM model that describes the AUX characteristics. The procedure is depicted in Fig. 3.13.

Once the initial procedure is complete, and the NN and GM model is obtained on the output, all other instances and data become redundant and are need not be stored.

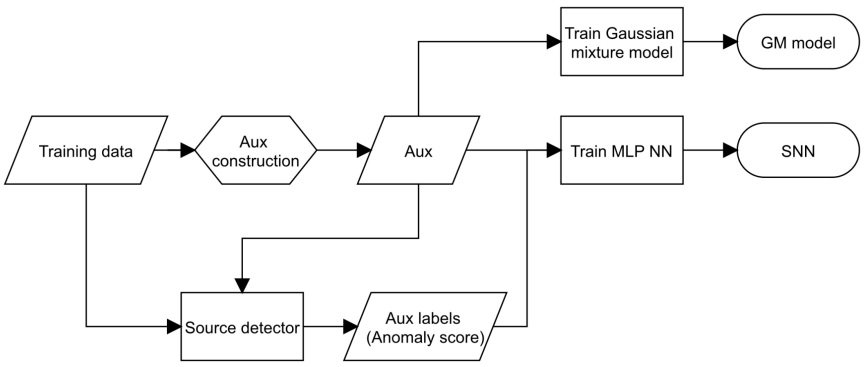


Figure 3.13: Initialization and first training of the model

3.5.2 Online Update Procedure

The online update procedure is applied to enrich the model with a batch of training data. In addition to training data, the GM and NN models are on the input of the procedure. In the high-level view, the procedure could be divided into four logical steps, which are to build AUX samples, calculate their labels, update the NN, and compress the AUX for output.

1) The AUX samples need to cover both the area of the feature space where the training samples were the history and also the area nearby the current update batch.

The first one was covered by the AUX in the last iteration, and thus data in the same distribution can now be generated with the GM model. The current batch is covered simply by the procedure used in the initialization or in the offline model. Then, the final AUX is obtained by joining the AUX for the current batch and the AUX representing the former data.

2) The calculation of labels (anomaly score) for the AUX samples must be calculated with respect to the current training batch and also with respect to the historical training data. Computation anomaly scores respecting the current training batch is straightforward, identical to the initialization, utilizing the source-detector. The other, the anomaly score corresponding to the historical training data, is provided by the NN model, as it is the main feature of the model. As a result, each sample has two anomaly scores and the scores are aggregated via minimum (see Sect. 3.4.2)

3) The neural model is updated with the labeled AUX data set analogically to the offline method (see Sect. 2.2.2). For the sake of simplicity, we update the NN with the same training setup as for the initialization training. However, in some specific applications, a different setup might be used, such as lowered training rate or a number of training iterations, etc.

4) The last part is to fit a GM model on the AUX data. In our experiments, we train the model from scratch with the current AUX.

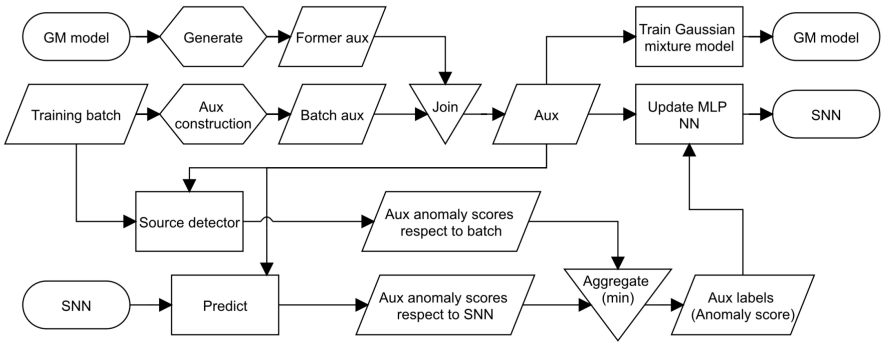


Figure 3.14: Schema of the update procedure

3.6 Experimental Evaluation

3.6.1 Evaluation Schema

For the online anomaly detection experiments, we provide two different evaluation schemes considering evaluation with and without concept drift. In the experiments, we adopt random resampling to provide statistical validation, and thus the evaluation scheme is applied repetitively. In the following text, we refer to one application of the scheme as the evaluation round.

Evaluation Schema Without Concept Drift

The intuition behind this schema is a progressive learning of the unchanging nature of the data, while each learning iteration should increase the quality of the model.

The AD data sets created using Emmott's methodology (see Sect. 1.4.2) are perfectly suitable for this scheme. They typically contain two types of samples, normal and anomalous. To simulate the real-world scenario with the AD data, we propose the evaluation schema shown in Fig. 3.15. The scenario simulates a consistent data stream; thus the accuracy will expectably be growing with every new training data coming into the model.

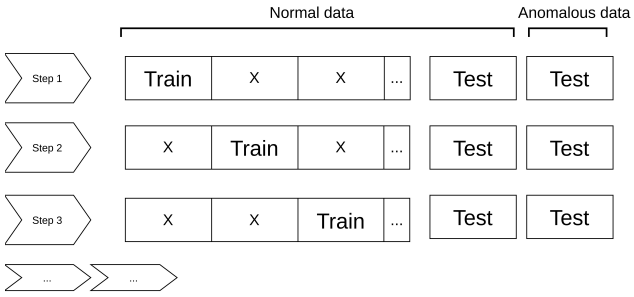


Figure 3.15: Evaluation schema for anomaly detection data sets. For each evaluation (evaluation round), the normal data are split to train and test sets. Subsequently, the train set is split into training batches of the same size. In each step, the model is updated (or trained in the first step) with the corresponding training batch and the accuracy is tested with the test data. Note that the test data are consistent for the whole evaluation (evaluation round) and the number of training batches is equal to the number of steps that is a parameter of the evaluation.

Evaluation Schema With Concept Drift

Utilizing multi-class data set could help us simulate the changing behavior of the data over time. In the first step, we assume a single class be normal and all other classes anomalous. This single class represents a consistent behavior of some examined subject. However, the behavior of a subject could change and a new type of behavior occurs. This is simulated by transferring a single class from anomalous into normal in the next step. In every step, the model is updated with a training subset of the specific class only, thus all the patterns from the previously trained data must be preserved in the model. The full schema is provided in Fig.3.16.

3.6.2 Data Sets

We utilize both the industrial data set with the computer network traffic and also publicly available data sets from the UCI repository [119].

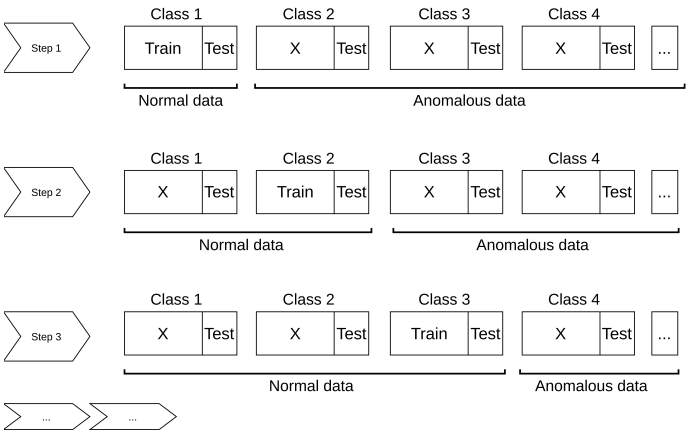


Figure 3.16: Concept drift evaluation schema for multi-class data sets. For each evaluation (evaluation round), each class is split to train and test subset. In each step, the model is updated (or trained in the first step) with the training subset of the corresponding class and the accuracy is tested with the test subsets across all classes. Note that the test subsets are consistent for the whole evaluation (evaluation round) but their labels might change. The number of training batches is equal to the number of steps that is derived from the number of classes in the dataset.

General Preprocess and Criteria

For each evaluation schema, the data set is preprocessed into the corresponding form. For the non-concept drift schema, the data sets are utilized in the form of anomaly detection data set according to the Emmots methodology (see Sect. 1.4.2) thus for each, four levels of difficulty are used. For the concept drift, the data sets are a little pre-processed so that the classes are sorted in descent order according to the number of samples (see Fig.3.17, 3.18).

Since our research line aims at the application in computer security, the security data set is naturally involved and it is utilized only in the non-drift schema. The other data sets are carefully selected based on their properties, mainly on the dimension and number of samples, and for the concept drift schema also on the number of classes and distribution of samples among classes. We aim at data sets with a comparable dimension to the security data set and possibly a larger number of samples. For the drift schema, we search for a data set with a higher number of classes (10 or more) each with a sufficient number of samples (sufficient to train a neural model), which is a relatively strict filter. For this reason, the computer network traffic is not suitable for concept drift schema.

Computer Network Traffic

The data set *persistent-connection* is utilized identically to previous experiments, thus the detailed description is given in section 2.4.1.

Abalone

The data are adopted from the UCI repository and utilized for both schemes. It has 4177 samples with 8 attributes. In addition to its properties, it is also included because it is partly utilized in the previous experiments so it can provide a wider picture.

The data set is prepossessed for both variants in accordance with the description above and for the concept drift, in addition, the number of update iterations is manually limited to 20 due to the very small number of samples in the skipped classes (See Fig. 3.17).

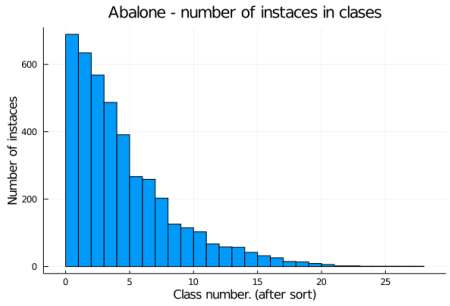


Figure 3.17: Distribution of samples among classes

Avila

Avila data set is also adopted from the UCI repository. It consists of 20867 samples with 10 attributes. It is utilized for the concept drift scheme in accordance with the description above. The size of the classes is demonstrated in Fig. 3.18

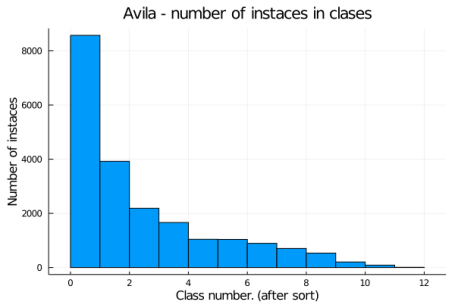


Figure 3.18: Distribution of samples among classes

3.6.3 Evaluation Metric

The evaluation is carried out in steps such that in every step, a simple AD task is evaluated.

To evaluate AD accuracy, we use the area under the curve (AUC) of the receiver operating characteristics (ROC) [31] as it is the most utilized metric for AD in the literature.

Then, to summarize the results across the steps we utilize graphical representation as accuracy in time (steps) and in addition, we compute area under the curve to obtain an over-all score.

3.6.4 Evaluation Setup

General Evaluation Scheme Setup

We use random resampling (8×) with respect to the evaluation scheme for each experiment. For the non-concept drift schema, random re-sampling is adopted such that 75% of normal (non-anomalous) samples are utilized for training while the remaining 25% for testing. The anomalous samples are used only in the testing phase. We evaluate 20 online steps, thus the training set is randomly divided into 20 batches of the same size. For the concept drift schema, random re-sampling is utilized analogically 75/25 but for each class. The number of online steps depends on the number of classes in the data set.

General Setup of the Algorithm

We utilize one-class kNN as the source-detector based on its performance discovered in Chapt. 2 and we consistently use $k = 5$.

The Gaussian mixture model is adopted from "A Julia package for Gaussian Mixture Models" library [120] with the following setup: $NrOfDistributions = \{1, \ldots, 10\}$, $CovMatrixType = diagonal$, $KMeansIters = 10$, $EMIters = 10$, $Method = kmeans$.

We parametrize the neural model as follows. We opted for a simple meta-optimization of neural model hyper-parameters. The number of hidden layers q varies between values $\{1, 2, 3\}$, hidden layer size p varies between values $\{1d, 2d, 3d, 4d, 5d\}$. ReLU activation function is used for all neurons (except for the input ones). The size of the batch is always set to 80 and the number of learning epochs to 20. We use MSE as the loss function and train the network with the *Adam* optimizer. In the first online step, the random weight initialization is repeated 8× and the best model is with respect to the loss function on the training data in the first training period.

Individual Setup

In addition to the generally described setup, there is a need to adjust some parameters in accordance with the data set individually.

For Abalone, the total number of auxiliary samples is set to $m = n \cdot d \cdot 10$ and for the concept drift scenario, we evaluate 20 online steps. For Avila we use the total

number of auxiliary samples $m = n \cdot d$ and for concept drift the number of online steps is 11.

Cisco network dataset (non-concept) For this particular dataset, we take advantage of our prior experiments demonstrated in Sect. 2.4 to provide an even more extensive comparison. We aim to continue the experiments under the most similar conditions to provide clear information on what accuracy loss is caused by iterative learning compared to offline learning.

Practically, most of the setup follows the general pattern. We use the above-mentioned general evaluation setup, while the setup shared with the offline model is adopted from Sect. 2.4.

Setup of Competitors

Despite the kNN being an offline method, it is included in the evaluations because it is the source-detector for SNN and thus it estimates the theoretical accuracy of the SNN . The only hyper-parameter k is set to 5 consistently with Chapt 2. Note that we use the consistent setup for k across all experiments, However, the experiments are replaceable for any k.

Since kNN is evaluated in an offline mode, each evaluation is carried out such that the corresponding data are provided concurrently instead of sequentially. In this way, the model is built up from scratch for each iteration.

To evaluate LODA we use the implementation from Python Outlier Detection (PyOD) library [121]. The number of bits is selected with an automated procedure and the number of random cuts is the subject of simple meta-optimization in the range $\{50, 100, 150, \ldots, 500\}$. We acknowledge that even though LODA is theoretically straightforward to update, the update routine is not implemented in the library and thus we factically evaluate LODA offline similarly to kNN. Note that the accuracy is not negatively affected by this simplification.

3.6.5 Results

With respect to the presented metrics (see Sect. 3.6.3), we provide both the aggregated overall scores in tables 3.1 (for comparison among proposed algorithms) and 3.2 (for comparison with competitors) and a more detailed and statistically descriptive view in Figs 3.19 and 3.20. To bring it back to mind, we evaluate two data sets with a concept drift nature and two data sets without concept drift that are divided into four levels of difficulty.

Experimental Comparison Among Online Update Algorithms

First of all, we provide an experimental evaluation of the proposed algorithms (see Tab. 3.1) for manipulating AUX (see Sect. 3.4.4). To briefly recapitulate, the al-

gorithms are sorted respectively to the level of information stored vs proxied while *Alg. A* is a non-approximated baseline, *Alg. D* memorizes only *Gaussian Mixture model* and is selected for further evaluation and *Alg. E* only stores the mean and variance vector.

Alg.	Abalone Concept Drift	Abalone (easy)	Persistant-conn. (easy)
A	0.747	0.985	0.988
B	0.776	0.976	0.979
C	0.784	0.983	0.970
D	0.776	0.980	0.972
E	0.467	0.700	0.815

Table 3.1: Comparison among algorithms processing AUX

The negligible accuracy difference among *A, B, C, and D* depicts the efficiency of the proposed approximation methods. Moreover, the results are very promising for the *algorithm D* that only stores a piece of lightweight information while the memory demand does not grow with the amount of training data used.

Based on the accuracy of Alg. *A, B, C, and D*, it may deceptively seem that the quality of the algorithm has no effect on the accuracy, and thus we can utilize even more trivial methods, the frivolous Alg. *E* suffers from a significant drop of accuracy with comparable storage requirements to Alg.*D*.

Accuracy Trend

Let us start the discussion with the non-concept drift experiments (see Fig. 3.20). The expectations are that the score of the model should grow with the online iterations because all the training data (in each online batch) have the same properties, and thus the model is supposed to increase its accuracy while continuously obtaining new data of the same nature. Note that the competitive offline kNN is evaluated such that in each step, it is supplied with the corresponding sub-sample of the full training data, and in the last step, it is supplied with the full training data.

The trend of growing accuracy is present in most of the non-drift experiments such that the offline kNN and online NN show stable and smooth increases in each step while LODA increases turbulently and its results also suffer from remarkable deviation and noise. However, some experiments do not expose this phenomenon apparently for one of the two following reasons. First, the model is trained well enough in the very early stage of the online learning simulation thus the model's accuracy almost stagnates at the same level instead of growing. This can be seen for example for kNN and online NN in the plot of Abalone(easy) or online NN in Persistant-connection (easy and medium). The second example of non-growing accuracy can be seen in Abalone (hard and v.hard) where the data set itself is probably too hard for all utilized algorithms because all of them have an AUC score near 0.5 which is the score of a randomly driven decision-making, in other words, the score is the lowest possible and the algorithms learned nothing. The presence of these experiments in this comparison will be discussed more in (Sect. 3.7)

For the concept drift experiments (see Fig. 3.19), there are no specific expectations on the accuracy trend. It could be either downward or upward, and the trend can change with every online step. This is caused by the nature of the data set and by the methodology of how the evaluation schema is designed. Clearly, in every step, a new class is added to the training set while a part of the test set changes its label, thus a partly new problem is evaluated. Then, it depends on the nature of the data and whether such a task is more or less difficult for the algorithm.

Anyway, the changing trend does not make our comparison less valuable, moreover, it could improve comprehension because all the methods face the same conditions and we compare their plots of accuracy in time. It clearly shows that the accuracy among the methods is correlated. In more detail, the experiment with the Avila data set shows a very accurate correlation with an absolute shift given by the accuracy of the algorithms. On the other hand, the experiment with the Abalone data set shows a correlated trend with less precise local similarity. The online NN accuracy plot is the most smooth because the model is re-trained with some persistence. In contrast to that, the offline kNN provides very sharp changes because it is trained from scratch for each iteration without any binding to previous iterations thus it reacts to the new data immediately.

Comparison of Accuracy

We perform an evaluation on two concept drift data sets where online NN outperforms other methods on the Abalone data set and offline kNN outperforms others on the Avila data set. The clarity of the dominance is depicted in Fig.3.19 such that the dominant method outperforms others in every single step of the simulation.

On the abalone non-drift dataset, the offline kNN shows the best accuracy while online NN is very near below kNN for *easy* and slightly below kNN for *medium* difficulty. Note that the experiments with Abalone (hard and v.hard) data sets have a very low descriptive value because all of the utilized algorithms reach a similar performance as a random classifier. From the definition of AUC ROC, the score typically lies between 0.5 (randomly driven decisions) and 1.0.

Evaluation of the peristant-connection data set follows the same phenomenon as observed in offline evaluation (see Sect. 2.4.4), specifically the dominance of the neural model for easier difficulties (*easy* and *medium*) and another way round dominance of kNN for harder difficulties (*hard* and *v. hard*).

In general, LODA provides the lowest accuracy in most of the experiments. In addition to the accuracy, LODA also suffers from lower robustness in comparison to the other methods.

Dataset	Online_NN	Offline_kNN	LODA
Abalone Concept Drift	**0.7764**	0.7089	0.6218
Avila Concept Drift	0.7320	**0.7937**	0.6546
Abalone (easy)	0.9796	**0.9884**	0.7007
Abalone (medium)	0.8697	**0.9235**	0.5799
Abalone (hard)	0.5307	**0.5605**	0.5034
Abalone (v. hard)	0.4734	0.4398	**0.5047**
Persistant-connection (easy)	**0.9725**	0.9507	0.9375
Persistant-connection (medium)	**0.9656**	0.9391	0.9197
Persistant-connection (hard)	0.9441	**0.9521**	0.8763
Persistant-connection (v. hard)	0.8044	**0.9001**	0.7354

Table 3.2: Aggregated scores of online anomaly detection accuracy.

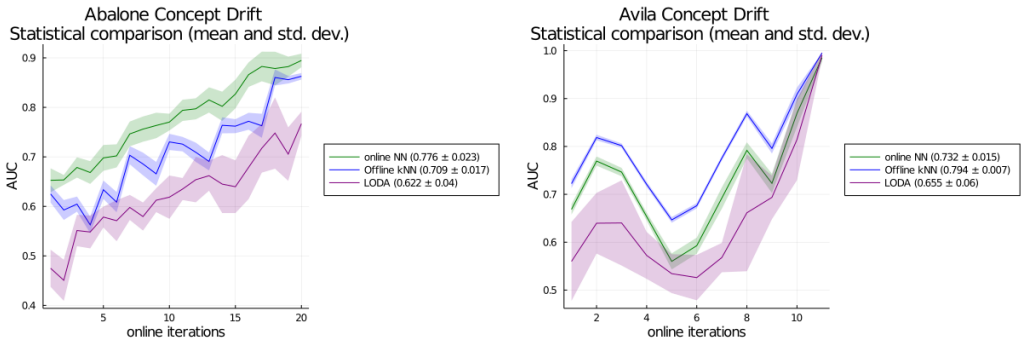


Figure 3.19: Concept drift results

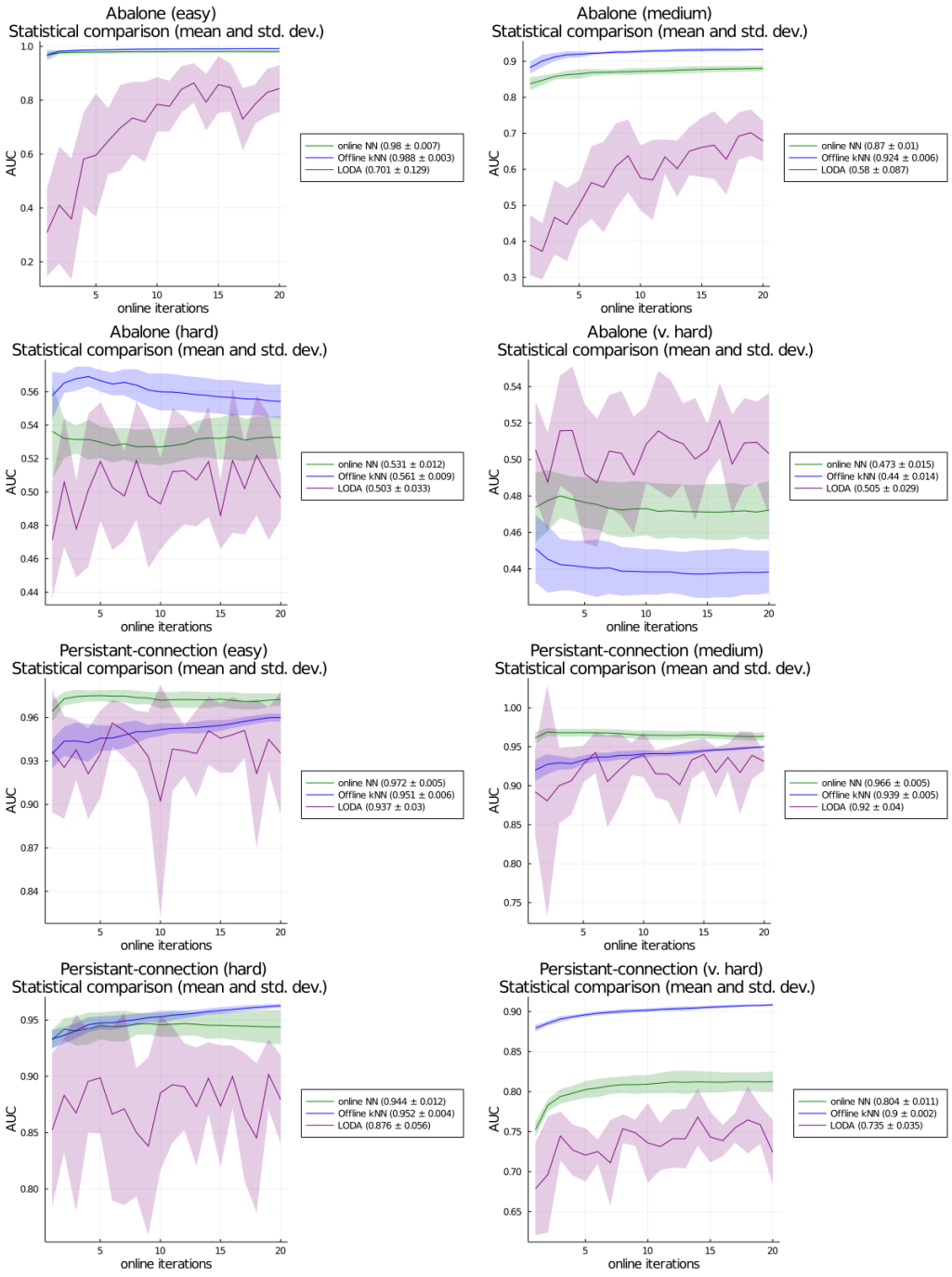


Figure 3.20: Non-concept drift results

Comparison With Offline NN

To evaluate the quality of the online model, we can also benefit from previous experiments with the offline model evaluated on the computer network telemetry (see Sect. 2.4.4) and compare the accuracy of the online and offline models. Note that this comparison has a two-fold descriptive value for our research line. First, it provides a general comparison with precisely described and carried out experiments with the offline model and thus explores a possible loss caused by the switch to the online approach. Second, the comparison is purposely targeted at the detection of anomalies specifically in the computer network security domain and thus the experiment observes the behavior of the model in the target domain.

To examine the potential loss of accuracy caused by switching from offline to online mode, we compare the accuracy of the online model in the last online step with the accuracy of the offline model. Clearly, this shows the difference in the accuracy between a model trained at once and a model trained gradually. Note that the online mode operates under more difficult and constrained conditions while using the identical neural model for the inference thus expectably, the resulting accuracy of the online model might be lower.

A simple comparison of the resulting accuracy (see Tab. 3.3) clearly shows that the online approach suffers from a reduction of accuracy in contrast to the offline approach. However, confidence interval analysis (see Fig. 3.21) shows a significant difference in accuracy only on *easy* difficulty. For the rest of the difficulties, the difference is not statistically significant.

To conclude, the online learning approach achieves slightly lower accuracy in comparison to the offline approach while the difference is mostly not significant. This is a very satisfactory result when taking into account the different learning paradigm and also the computation efficiency based on the extensive use of approximation.

Dataset	Online NN	offline NN
Persistant-connection (easy)	0.972	0.987
Persistant-connection (medium)	0.963	0.979
Persistant-connection (hard)	0.944	0.967
Persistant-connection (v. hard)	0.812	0.832

Table 3.3: Accuracy of the online NN in the last step and offline NN model AUC (ROC)

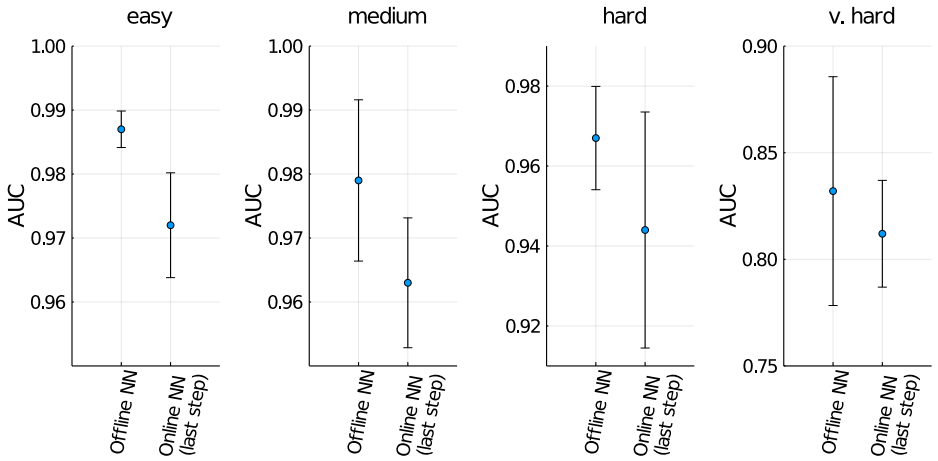


Figure 3.21: Comparing the accuracy achieved by the offline and online (in the last step) detectors on the network security problem. Confidence intervals for mean AUC of ROC at 95% level. Four problem difficulties (left to bottom right): *easy*, *medium*, *hard*, *very hard*.

3.6.6 Stability

The stability of the model is an important factor in model-driven detection. In this section, we discuss and experimentally discover the behavior of the model under specially defined conditions.

In general, the model is naturally exposed to changes in each update routine. To be capable of learning and adapting to the new data, the model might decrease its accuracy with respect to the previously learned data. However, this problem (trade-off between stability and plasticity [106]) is related to any iterative learning and in our work is well covered by the main experiments. More importantly, we address the question of how much is the model exposed to spontaneous degradation when repeatedly updated without any training data.

Analysis

First, the model utilizes the Gaussian mixture model to approximate the distribution of the AUX. Note that the process is fully independent on the neural model and thus we can discuss it separately. The repetitive computation of GM parameters and generating the samples from the distribution could also suffer from degradation to some small extent. The theoretical question of where would the distribution converge is not addressed in this work, because practically, the model is always updated with new data when the update routine runs. Anyway, the AUX is very resistant to slight and moderate changes in the parameters of the distribution according to the previous experiments.

Second, the model inference feature is driven by the NN, which is a crucial part of the whole model. In this point of view, the model is very robust because in each update procedure, the AUX labels are inferred with the current NN. Afterward, the

same labels are used for the NN update and thus the AUX perfectly fixes the NN output on the true values.

Remark: This feature also protects the NN from degradation in the standard update procedure with data. Clearly, it preserves the anomaly score values outside the area of the current update batch as demonstrated in Sect. 3.3.2.

Stability Experiments

To demonstrate the stability of the model, we provide experimental evaluation. The principle is that the model is trained under identical conditions as described in 3.6.4 and after that, a number of update procedures are carried out without data while the accuracy of the model is measured.

We provide this experiment for the setup with the network telemetry data set which is non-concept drift (see Fig. 3.22) and also for the Abalone data set in the concept drift setup (see Fig. 3.23).

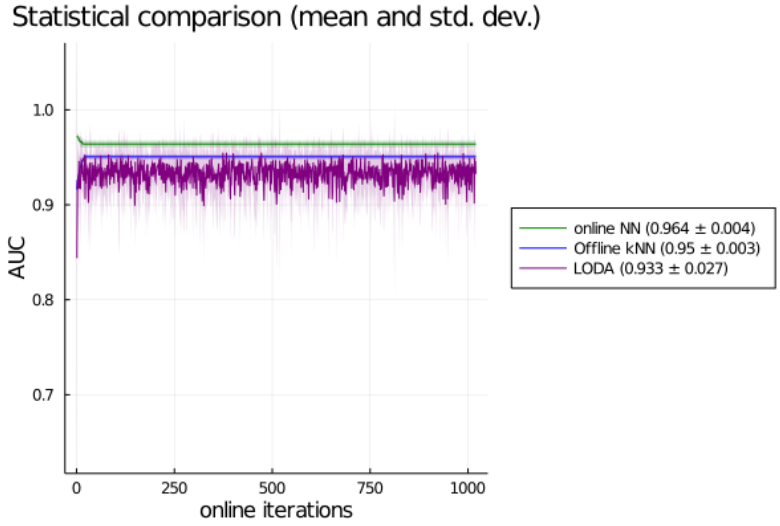


Figure 3.22: Experimental evaluation of stability with the persistent connection (medium) data set based on computer network traffic (non-concept drift). The online NN model is first trained in 20 online iterations with the data under the same conditions as presented in Sect. 3.6.4 with results in Fig. 3.20 and then 1000 training iterations is carried out without any data while the accuracy is measured. Note that the neural model have stable accuracy and thus the degradation of the model does not occur. The values for the competitors in the iteration range 21-1020 are independently computed with all training data since the empty training procedure is not defined and thus the model degradation cannot be examined.

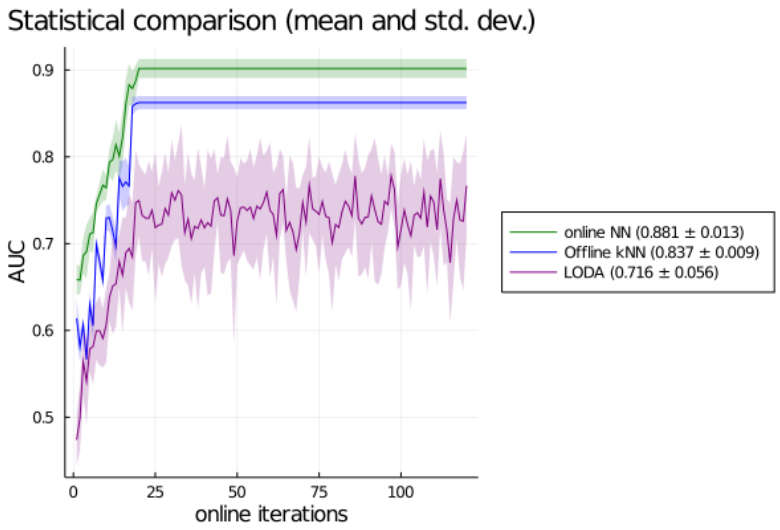


Figure 3.23: Experimental evaluation of stability with concept drift Abalone data set. The online NN model is first trained in 20 online iterations with the data under the same conditions as presented in Sect. 3.6.4 with results in Fig. 3.19 and then 100 training iterations is carried out without any data while the accuracy is measured. Note that the neural model have stable accuracy and thus the degradation of the model does not occur. The values for the competitors in the iteration range 21-120 are independently computed with all training data since the empty training procedure is not defined and thus the model degradation cannot be examined.

3.7 Discussion and Summary

We propose a novel online learning model based on offline SNN while retaining the advantages of the original offline model. The main contribution is the design of the updating procedure while addressing the challenge of memory and computational sustainability when the model is continuously updated. We demonstrate the tradeoff between appropriate computations and efficiency through approximation and propose 5 various algorithms at different levels of approximation while the selected one provides both efficiency and solid accuracy. We further examine the quality of the selected algorithm.

We experimentally compare the accuracy of the proposed method with the most relevant state-of-the-art in online anomaly detection and also to offline kNN that is utilized as a source-detector. We provide the evaluation in two scenarios with and without concept drift. The proposed method delivered a comparable accuracy to kNN while enabling online learning ability and reducing inference speed by orders of magnitude. Furthermore, it remarkably outperforms its competitor in the field of online learning.

We demonstrate that the accuracies of offline and online SNN on the network security data set are statistically comparable. We have replicated the former offline experiment with the online (more difficult) scenario while preserving most of the setup to discover offline vs. online qualitative differences in network telemetry analysis. We observed that three out of four experiments provide statistically comparable results

and one (for *easy* difficulty) depicts a slight and statistically significant accuracy reduction for online setup. However, when we study the accuracy of the *easy* problem in a closer context, we observe that even though the online NN is outperformed by offline NN, it outperforms all other competitors in online and also offline contexts (see Tab. 2.7).

We also discuss and experimentally evaluate that the model is resistant to spontaneous degradation with continuous update procedures.

The other minor advantage of the online approach is the spread of the training overhead. For both the online and offline versions, most training overhead consists of computing the AUX labels with the source-detector (kNN). The online approach can be used also to spread the training overhead among more training actions.

In our evaluated implementation, the model size is constant when the model is stored and deployed but the memory demand for the updating procedure increases with training data due to the full reconstruction of AUX. In most industrial applications, short-time memory demand is not limiting, however, there are solutions for limited memory. For example, the AUX can be reconstructed from the GM in the batches that are directly utilized for training the NN model, and thus no large data needs to be stored.

Future research could also focus on observing other types of models to store the AUX and ultimately to discover new update schema without a separate model. An incremental step might be to apply a neural generative model (GAN, VAE,...) instead of the GM model and to study its pros and cons. However, at this stage of the research, the GM model provides sufficient functionality in the sense of not being the bottleneck. Thus utilizing the generative neural models is not necessarily bringing improvement. And on the other hand, the generative models are harder to represent, understand, and train and even more complex to use in general [122, 123]. In other words, research on deep models for storing AUX is too risky regarding a very low opportunity to improve the properties of the proposed method. On the contrary, shallow and simple generative models might be an interesting opportunity to simplify storing AUX as long as the accuracy does not deteriorate.

Chapter 4

Surrogate Anomaly Detectors in Multiple Instance Learning

4.1 Introduction

In previous chapters, we explored the applicability benefits and limits of surrogate anomaly detectors on a standard vector representation. However, many real-world problems have a structured representation thus it is beneficial to search for more general paradigms.

Across the internet and digital industries, a large amount of data have naturally a tree-structured, multi-type form of constant or flexible structure. This is especially true with machine-generated data like network telemetry, computation logs, transaction logs, and others. Many software systems store and process such data in formats like JSON [124]. Applying standard Machine Learning techniques to such data is possible but not straightforward, and with current techniques likely highly suboptimal. This is due to the necessity to bridge the gap between structural and vector forms of information. Explicitly defined transformations quite often prove sub-optimal. For this reason, the field of Multiple Instance Learning (MIL) has been rapidly expanding recently, as it promises to open paths toward efficient learning from structured data.

Anomaly Detection in MIL setting has been addressed before but prior art is even less satisfactory than in the case of mainstream AD on vector data. In this chapter, we will thus follow the similar reasoning as in Chapter 2 and will strive to build surrogate models for some of the best available MIL AD in a bid to achieve models with comparable accuracy but significantly better applicability in an industrial setting (as we will see, MIL AD known methods have orders of magnitude higher computational complexity than vector ADs).

As shown in the previous chapters the idea of building SNN with kNN as source AD is very powerful for many problems. In this chapter, we show that it can be extended for the MIL AD setting.

Adopting SNN for MIL is requires combining tools from multiple fields: anomaly

detection, neural networks, MIL paradigm including the notion of bag distance metrics, neural networks for MIL, and finally neural anomaly detection for MIL. Such an objective is not straightforward as we are facing a fundamental difference between vector and MIL problems. To solve this, we address the following problems: How to find an optimal coverage of the MIL space and how to construct the auxiliary set? Is it possible to learn the distribution of distributions instead of covering a space? Is it possible in general cases or under the use of certain assumptions in our context? How to design a MIL neural network with the best fit for the NN introduced in Sect. 2.2.2?

4.2 Multiple Instance Learning

In this section, we give an elementary introduction to the MIL paradigm with a focus on basic principles that are related to the problem we intend to solve. The MIL classification methods are well and widely researched as described in a survey [13] that provides basic definitions, notation, and taxonomy of the existing methods (see also other surveys [125, 126]). By the end of the section, we provide a literature review of MIL neural networks (sect. 4.2.5).

MIL is a special type of learning and it has become important in many fields [13] including pharmacy, information retrieval, computer vision, signal processing, economy and cyber-security in particular. Instead of operating on labeled instances, the MIL operates on labeled bags of many various instances. A simple example of a MIL application is given on a key chain problem in [127]. There are several people owning their key chains. Where the chain represents the bag and the key represents the instance. Each door can be accessed with one or more chains. A possible task could be to predict who is able to unlock any selected door.

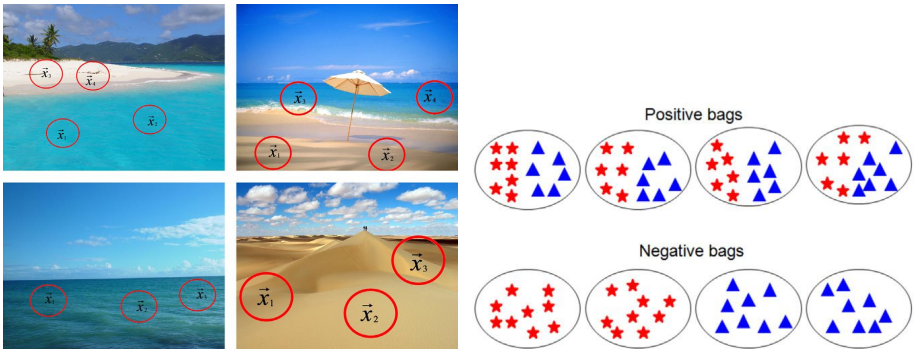


Figure 4.1: Classification of images into the beach (top row) and non-beach (bottom row) based on instances describing the water or the sand. The beach is represented with positive bags. The bag is classified as positive if it consists of instances of both types (water and sand). [13]

Another toy example, presented in [13] shows the benefits of using the MIL paradigm to recognize a beach based on simple instance features describing water or sand (see Fig. 4.1).

4.2.1 Concept

A bag is a set of instances $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ which means that each instance is a feature vector in the instance space of dimensionality d. Cardinality N may be different across bags. Many applications assume binary bag labels (positive or negative) thus the goal is to learn a classification function $F(\mathbf{X}) \in [0, 1]$ that is able to classify unseen bags. Some methods use classification on the instance level $f(\mathbf{x}_i)$. *note: The uppercase is used to refer to the bag level ($F()$, $\mathbf{X}$) and lowercase to the instance level($f()$, $\mathbf{x}_i$)* [13]

We give an overview of the three essential paradigms utilized in MIL in the following sections. The first, bag space, (Sect. 4.2.2) and second, embedded space (Sect. 4.2.3) paradigms are global in the sense of utilizing the bag-level information. The difference is how they deal with the transformation to standard supervise learning. The bag space paradigm uses a distance function for a pair of bags while the embedded space paradigm uses embedding to vector space. The third, instance-level paradigm (Sect. 4.2.4) operates only on the instance level without taking the bag information into account. [13]

4.2.2 Bag Space Paradigm

The bag space paradigm (BS) uses the information at the bag level. Each bag is represented by all instances and thus it adopts global, bag-level information. Since the bag space is a non-vector space, the key point for most non-vector learning algorithms is the definition of the distance function for a pair of bags $D(\mathbf{X}, \mathbf{Y})$. After that, well-known methods such as kNN and SVM can be utilized. The illustration of the paradigm is depicted in Fig 4.2. [13]

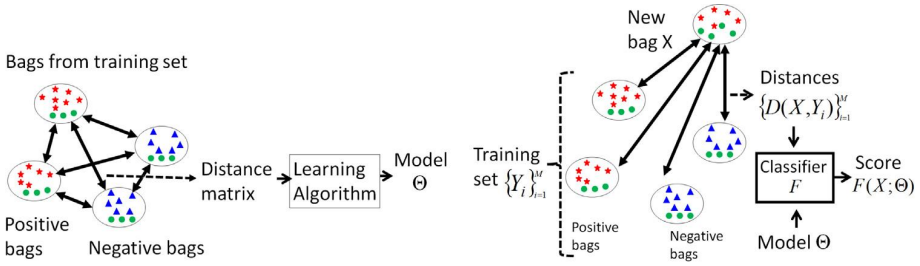


Figure 4.2: Illustration of the BS paradigm: training (left) and inference (right). [13]

As mentioned above, the distance definition is essential for BS. Since the bag is a set of vectors, the distance function $D(\mathbf{X}, \mathbf{Y})$ operates on sets. Such distances are minimal Hausdorff distance [128], Earth Movers Distance (EMD) [129], Chamfer distance[130], the kernel by Gartner, [131] and the advanced maximum mean discrepancy which is described in detail below.

Minimal Hausdorff Distance

We give a brief description of the minimal Hausdorff distance as the simplest one. It was introduced in [128] and it is a modification of Hausdorff distance [132] for the purpose of the MIL. The distance represents the closest distance between sets:

$$D(\mathbf{X}, \mathbf{Y}) = \min_{\mathbf{x} \in \mathbf{X}, \mathbf{y} \in \mathbf{Y}} ||\mathbf{x} - \mathbf{y}||$$

A detailed description of the mentioned and other distances could be found in [13, 133]

Maximum Mean Discrepancy

In MIL the maximum mean discrepancy (MMD) can be utilized for distance measure between two bags. It was introduced in [134] and it is widely used in many fields. The Maximum mean discrepancy is a statistical test to measure the similarity of distributions of two samples.

A brief description of the original idea is as follows. Let us have distributions p and q. To measure the difference, we need to find a smooth function operating on instances such that it "separates" the distributions as much as possible. In other words, a function that projects samples from p to as high numbers as possible and on the contrary samples form q to as small (or negative) as possible. The only possible case in that we are not able to search for such a function is if and only if the distributions are equal. After the function is observed we can measure the difference of the distributions such that we compute the mean value of projections from p and q. Then we can say that the more the average projection of p and q differ the more the distributions differ.

The original paper emphasizes the selection of the function and shows that it affects the quality of MMD. As a result, Gaussian and Laplace kernels are recommended to use and in addition, the linear kernel is used in practice. To define these functions, let us assume a kernel function $\kappa(\mathbf{x}, \mathbf{y}) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ where d is the dimension of instances. Then the linear kernel is defined as: $\kappa(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$ and the Gaussian as $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\gamma||\mathbf{x} - \mathbf{y}||^2)$ where γ represents inverse kernel width. [15]

The distance between two bags is then computed as:

$$MMD(\mathbf{X}, \mathbf{Y}) = \frac{1}{\binom{n}{2}} \sum_{1 \leq i \leq j \leq n} \kappa(\mathbf{x}_i, \mathbf{x}_j) - \kappa(\mathbf{x}_j, \mathbf{y}_i) - \kappa(\mathbf{x}_i, \mathbf{y}_j) + \kappa(\mathbf{y}_i, \mathbf{y}_j)$$

The formula above expects that number of instances is n for both bags. However, the formula for different numbers of instances is given in [134]

4.2.3 Embedded Space Paradigm

In the embedded space paradigm, each bag is transformed into a feature vector that represents the bag. The embedding is carried out such that the global information

is preserved and thus the paradigm is global and uses bag-level information as well as bag space paradigm but in a different way. After the bag space is transformed into an embedded (vector) space, the classifier can be easily learned, using off-the-shelf algorithms. The embedded space paradigm deals with vectors and their labels thus methods such as neural networks, SVM and others can be utilized. The ES paradigm is illustrated in Fig. 4.3. [13]

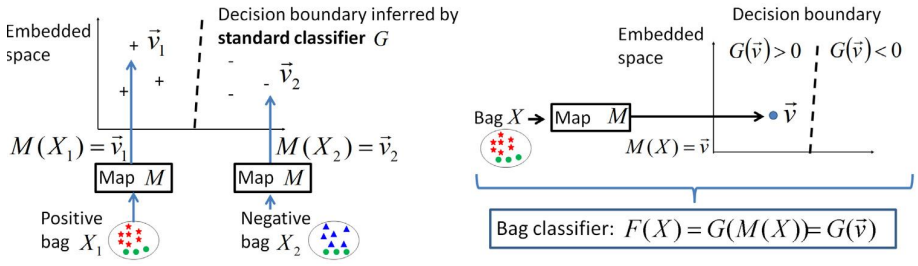


Figure 4.3: Illustration of the ES paradigm: training (left) and inference (right). [13]

The transformation is performed with mapping $\mathcal{M} : \mathbf{X} \rightarrow \mathbf{v}$. The information about the entire bag is coded into one single vector. The mapping algorithm is dependent on the data information and structure and it is crucial for the classification performance. The mapping algorithms have two categories (with vocabulary and without). The non-vocabulary are simpler and aggregate the information of instances in the bag without differentiation. On contrary, the vocabulary methods perform embedding with emphasis on the structure and difference to instances that already have been observed [13].

Non-vocabulary

A simple example of non-vocabulary embedding is called "Simple MI" [135]. The mapped vector is computed as a mean of the instances:

$$\mathcal{M}(\mathbf{X}) = \frac{1}{|\mathbf{X}|}\sum_{\mathbf{x}\in\mathbf{X}}\mathbf{x}$$

Another possible method as shown in [131] is to map the bag into min-max vector that represents min. and max. boundary of the bag. It means $\mathcal{M}(\mathbf{X}) = (a_1, a_2, \ldots, a_d, b_1, b_2, \ldots, b_d)$ where $a_j = \min_{\mathbf{x}\in\mathbf{X}} x_j$, for $j = 1, \ldots, d$ and similarly b is maximum [13].

Note that in addition to the mentioned types of embedding, various others exist or could be defined, and moreover the embeddings could be arbitrarily combined. For example, mean-min-max or only mean-max (*meanmax*) does make sense for various applications.

Vocabulary

The vocabulary approach analyzes all instances in each bag first and then performs the embedding. Firstly, most algorithms observe classes in instance space. This is usually carried out as unsupervised learning because there are no labels inside the bag. To give a simple example of a vocabulary method, let us have 2 classes in the instance space and bags that contain either instances of the same class or mixed instances of both classes. In order to classify one of these two classes of bags, the algorithm performs clustering first then analyzes the number of instance classes for each bag, and finally, for example, utilizes a histogram to represent the bag [13].

4.2.4 Instance Level Paradigm

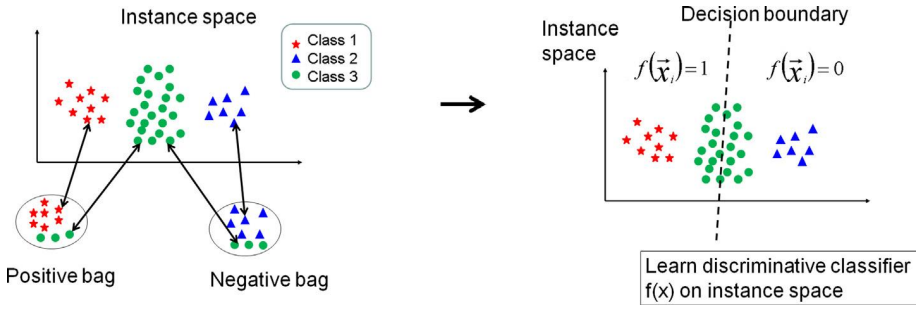


Figure 4.4: Illustration of the IS paradigm [13]

Instance Level Paradigm (IS) operates on the instance level such that a classifier $f()$ for instances is trained. The consequent classification of the bag is performed as an aggregation of classification over all instances in the bag [13]. Formally:

$$F(\mathbf{X}) = \frac{f(\mathbf{x}_1) \circ f(\mathbf{x}_N) \circ ... \circ f(\mathbf{x}_N)}{Z}$$

where $\circ$ denotes aggregation operation and Z is a normalization. An example of normalization could be $Z = N$. However, it practically depends on the MIL algorithm and aggregation. One of the frequent aggregations used for IS is a method following *the collective assumption* as follows[13] :

$$F(\mathbf{X}) = \frac{1}{|X|} \sum_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x})$$

Another often-used aggregation is used when it is assumed that every positive bag contains at least one positive instance (SMI assumption)[13]:

$$F(\mathbf{X}) = \max_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x})$$

4.2.5 Neural Networks for Multiple Instance Learning

In general, the neural networks for MIL are typically composed of three logical parts. First, NN that operates on instances; second, embedding function that aggregates the information from all instances across the bag and provides a vector bag representation; third NN that operates on the bag level.

An early and elementary form of MIL NN was proposed in [136]. The authors define a simple multiple instance error function for training the neural model that operates on instances only. The error function performs a simple aggregation thus the embedding is not explicitly performed. As a result, the neural model operates on the instance level. Later, neural models operating on both instance and bag levels were introduced in [137, 138] where commonly used aggregation functions are utilized. The authors consider *mean*, *max*, *meanmax* and *log-sum-exp*.

More recently, [139] provided a family of permutation invariant functions operating on sets which is a parallel to the aggregation functions described above. The variability enables a range of scenarios and data types for which the NN can be designed. An alternative approach to the pre-defined aggregation function is demonstrated in [140] where an attention mechanism is performed with weighted *mean* embedding that is trained simultaneously with the network.

MIL is also widely used in connection with convolutional neural networks with a frequent application to medical image processing. In some of the applications, it is more beneficial to segment the image into smaller areas and apply the MIL paradigm [141, 142, 143].

4.3 Multiple Instance Learning for Anomaly Detection

The usage of MIL could be beneficial for anomaly detection as well as for classification. The opportunity to take into account more instances for each object (bag) would enable as of now prohibitive problems to be solved. One such area of interest is cyber-security. Known anomaly detection techniques are not well suited to enable the detection of anomalous nodes in computer networks, where the node can not easily be represented by a single vector. Instead, a computer network node would more naturally be characterized by a hierarchy of vectors easily extracted from its various connections to other nodes or servers, or from various types of internal system activity. A generic MIL-based anomaly detection technique would then enable the straightforward application of anomaly detection on the network node level, which is practical for administering large-scale networks, even on a worldwide scale. However, the MIL for AD is in the early stage of development and the published research is clearly lacking in comparison to standard anomaly detection.

4.3.1 Prior Art

The most related approach is described in [15], where the authors utilize MIL paradigm to detect steganographers such as users with anomalous behavior. They perform distance measure for pairs of users with Maximum Mean Discrepancy (see Sect. 4.2.2). Consequently, the outlier is observed as a more distant object from the others. An application to network security is demonstrated in [14], where the authors propose to model anomalies in the network traffic with a MIL paradigm such that the packets are converted to instances and related packets are grouped into bags. The authors propose a trivial averaging metric to measure the bag similarity that is used for decision-making.

Group anomaly detection using flexible genre models was introduced in [144]. The method utilizes both single and group-level approaches to detect anomalous observations of various types. In Hierarchical probabilistic models for group anomaly detection [145], two different models are utilized. Authors start with a discrete model based on LDA [16] that is originally a probabilistic generative model for the purpose of text document comparison. Then introduce its Gaussian modification GLDA that handles continuous variables. In addition to that, they modified GLDA to MGMM for supporting multi-modal distributions. Group anomaly detection is also applied to social media analysis in [17].

Another approach is demonstrated in One-class support measure machines for group anomaly detection (2013) [146] where the anomalous objects are detected with analyses of distribution and moments of higher orders.

Finally, MIL anomaly detection was also successfully applied to image processing in medicine [147] and similarly to video processing [148, 149, 150, 151]. The main contribution of those papers is typically defining features and performing segmentation or partitioning of the input data in order to transfer the data into the MIL form. Then the MIL algorithms vary from AD detection via similarity measure to classification trained with positive bags.

Limitations of Prior Art

Despite some of the principles in the prior art being suitable to a wide range of problems including the problem we intend to solve, they suffer from prohibitive computational complexity even for medium-scale data. Other methods typically suffer from limited expressiveness and possibly also prohibitive computational complexity. Moreover, the evaluation of the published methods is not standardized yet. This is because the prior works are mainly focused on the context-dependent application to specific fields. Thus the utilized data sets are different and moreover are composed of different types of data across the methods. To give a few examples, some of the methods operate on images to find an anomalous visible object, [15] operates on images as well but is looking for steganographic behavior thus the features and data are absolutely different. Other examples are social media analyses, the collection of documents, and telemetry data. On contrary, there are several standardized data sets [152] used in MIL classification literature but there seems to be no prior art in

general MIL-based AD applicated to them. The specifics of our problem that we do not see solved in Prior Art are as follows: we seek a generally applicable method, fast at the inference phase, easy to operate, with good scalability enabling application in the extreme industrial setting we face in cyber-security.

4.4 Adapting SNN for MIL Anomaly Detection

In MIL setting the computational advantage of a surrogate neural model over k-Nearest Neighbor would be expectably even more striking than in the vector data case. Known MIL neural models are computationally relatively cheap [153] (although more expensive than standard vector models), while the nearest neighbor search on MIL bag space can be expected extremely costly (cf. Section 4.2.2; MMD or similar measure would have to be evaluated pairwise between bags. Compare to simple L2 on vector space). Another factor that rapidly increases the performance gap is that the kNN operating on vector (non-MIL) data is routinely accelerated with search trees that cannot be utilized in the MIL setting. See the crucial difference of kNN inference speed with and without search trees in Fig. 2.11.

In the previous chapters, SNN (Sect. 2.2) has been successfully utilized for anomaly detection on vector data with benchmarks on publicly available data sets (Sect. 2.3) and also on an industrial data set from the field of computer security (Sect. 2.4).

We aim at replicating the idea of SNN for MIL and take benefit of the well-performing kNN for MIL AD [15, 14]. The opportunity to reduce the inference time is even more astonishing regarding the computational complexity of MIL distance metrics and the unavailability of supporting structures such as search trees in the space of MIL. As a result, the MIL SNN could be an enabler for many, as of now prohibitive, applications. However, the adaption to MIL is not straightforward as there are two main challenges:

1. Construction of MIL auxiliary data set

2. Adopting multiple instance neural network structure

The construction of the MIL AUX data set is the most critical and complicated element of the method. Generating the space of bags (MIL space) has lacking prior art and is more complex in the sense of missing baseline full coverage contrary to a vector space. To illustrate that, let us discuss the task with non-MIL construction. The baseline uniform approach (Sect. 2.2.1) simply generate vectors within a hyper-block to provide full coverage of the space. However, such an approach is mostly impossible to carry out for MIL space coverage. Instead, we have to focus on more sophisticated and generative approaches.

4.4.1 MIL Auxiliary Data Set Construction Strategies

The construction of the auxiliary set (AUX) is an essential part of the proposed algorithm. We have to follow the nature and properties of the dataset to achieve optimal coverage of the bag space. It means not only creating such bags that are very similar to the bags in the training set but also generating bags that cover the neighborhood. In some cases, it might be even beneficial to cover more distant neighborhood where anomalous bags could be located. This all is way more difficult since we operate on bag space where the definition of the neighborhood is ambiguous.

We consider the following properties of the data set when constructing AUX:

1. Distribution in the instance space in general
2. Bag-related distribution in instance space
3. Distribution of the cardinality among bags

It would be possible to generate such AUX that has all three properties identical to the training set but that would suffer from insufficient diversity to operate well. The goal is to generate possibly all bags that describe the same problem and the samples could even belong to yet unknown classes. Intuitively, this is an analogy to the generative process introduced in Chapt. 2 where the samples cover the area of the training set and also its close and more distant neighborhood (see Fig. 2.2). As the neighborhood of the training set in the bag space is complex, we utilize various strategies to generate the neighborhood in multiple scenarios. We will discuss how each method preserves and generate the properties mentioned above.

Notation

Let us define the following notations for the explanation of the generative methods given below:

Training Set $\mathbb{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_M\}$ consists of bags $\mathbf{X}_i$ such that each bag consists of instances $\mathbf{X}_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,N_i}\}$ where instances $\mathbf{x}_{i,j} \in \mathbb{R}^d$; $i = \{1, \ldots, M\}, j = \{1, \ldots, N_i\}$, and where M is number of bags and cardinality N_i varies across the bags.

Thus analogically, the training set can be defined also as a set of all instances:
$\mathbb{X} = \{\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \ldots, \mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3}, \ldots, \mathbf{x}_{M,N_M}\}$

Analogically AUX Set $\mathbb{A} = \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_L\}$, $\mathbf{A}_i = \{\mathbf{a}_{i,1}, \mathbf{a}_{i,2}, \ldots, \mathbf{a}_{i,N_i}\}$ where instances $\mathbf{a}_{i,j} \in \mathbb{R}^d$; $i = \{1, \ldots, L\}, j = \{1, \ldots, N_i\}$, and where L is number of bags and cardinality N_i varies across the bags.

Thus analogically: $\mathbb{A} = \{\mathbf{a}_{1,1}, \mathbf{a}_{1,2}, \mathbf{a}_{1,3}, \ldots, \mathbf{a}_{2,1}, \mathbf{a}_{2,2}, \mathbf{a}_{2,3}, \ldots, \mathbf{a}_{L,N_L}\}$

Subsidiary Sets are used in the description of AUX construction. Let us denote them with tildes organized in the following order (e.g. for instaces): $\mathbf{x} \to \tilde{\mathbf{x}} \to \tilde{\tilde{\mathbf{a}}} \to \tilde{\mathbf{a}} \to \mathbf{a}$.

Hyper-parameters are used to control the generative process. Generally explained, the *multiplication constant* typically determines the size of the AUX set in dependence on the size of the training set and the *scale of the generator* defines the magnitude of uncertainty or change applied to the data. The specific setup and range of the hyper-parameters are discussed in Sect. 4.5.2. For better clarity, the functions of the hyper-parameters are explained for each strategy separately.

4.4.2 Strategy 1 – Single Instance Bags

Generating the single instance bags is the simplest algorithm for obtaining the AUX data set that operates rather on the instance level. In other words, the idea is to generate instances based on the instance-level information and then wrap each single instance with a bag.

We use modified Parzen window estimation to generate new instances $\mathbf{a}_i$ in the neighborhood of all instances in the training set. More exactly, we adopt the algorithm described in Sect. 2.2.1. Note that the generative process is also controlled by hyper-parameters *multiplication constant* and *scale of the generator*.

The approach follows and generates instance space distribution while the bag-related distribution and cardinality are irrelevant.

4.4.3 Strategy 2 – Noise to Instances

The idea of this algorithm is to generate bags that are very similar to the training data. It actually replicates the original bags by adding specific noise to each instance. The algorithm repetitively ($L > M$) iterates over the training bags while each bag is used to create a new noise-modified AUX bag. Note that $L = M \cdot$ *multiplication constant*.

Formally:

$$\tilde{\mathbf{A}}_i = \mathbf{X}_{(i \mod M)} \ ; \ i = \{1, \ldots, L\}$$

$$\mathbf{A}_i = \{\tilde{\mathbf{a}}_{i,1} + \mathcal{N}(0,\, h), \tilde{\mathbf{a}}_{i,2} + \mathcal{N}(0,\, h), \ldots, \tilde{\mathbf{a}}_{i,N} + \mathcal{N}(0,\, h)\} \ ; \ i = \{1, \ldots, L\} \quad (4.1)$$

where h is a hyper-parameter of the algorithm controlled with *scale of the generator*.

As a result, the distribution of cardinality is preserved as well as the bag-related distribution in the instance space while the instance distribution is generated.

4.4.4 Strategy 3 – Genetic Algorithm

We can also use a generative approach inspired by genetic algorithms to generate various bags across the bag space. First of all, we need to the analogy to the basic genetic operations that are crossover and mutation:

Crossover: Let us assume bags $\mathbf{Y}$, $\mathbf{Z}$ and $\alpha \in (0, 1)$ then we define cross$(\mathbf{Y}, \mathbf{Z})$ as random selection of $(1 - \alpha) \cdot N_Y$ instances from $\mathbf{Y}$ and $\alpha \cdot N_Z$ from $\mathbf{Z}$. By default we utilize $\alpha = 0.5$

Mutation is carried out as the addition of Gaussian noise to instances.

Procedure description: The algorithm repetitively iterates over the training bags to create L new candidates. Note that $L = M \cdot$ *multiplication constant.*

$$\tilde{\tilde{\mathbf{A}}}_i = \mathbf{X}_{(i \mod M)} \; ; \; i = \{1, \ldots, L\}$$

Then each candidate is crossed with a randomly selected bag from the training set:

$$\tilde{\mathbf{A}}_i = \text{cross}(\tilde{\tilde{\mathbf{A}}}_i, \mathbf{X}_j) \; ; \quad i = \{1, \ldots, L\} \; , \; j = \text{rand}\{1, 2, \ldots, M\}, j \neq (i \mod M)$$

Finally, the mutation is performed as $\mathbf{A}_i = \text{mutation}(\tilde{\mathbf{A}}_i)$ identically to equation 4.1.

This algorithm preserves the overall distribution on the instance level but variates the instance distribution with respect to the bags. It also preserves the number of instances in bags on average but the crossover could also create a bag of cardinality that was not present in the training set.

Our genetic approach fulfills all three properties of interest. Since we mix bags to create new ones, new cardinality could originate as a mixture of two cardinalities. However, the mean cardinality of AUX and the training set will be statistically comparable. The Bag related distribution is also generated when mixing the bags and the instance distribution is generated with mutation.

4.4.5 Strategy 4 – Vocabulary-based MIL Generation

To create a complex generator, we address the following concerns: generative model on the instance space, generative model of the bag size, and generative model on the bag level. The high-level idea is to transfer the MIL dataset to standard vector representation with a vocabulary embedding, apply the generative algorithm of bags on the representations, and subsequently, create bags with generating instances using the vocabulary which is a reverse operation of the first logical step.

Creating vocabulary and representation: In most of the MIL data sets, the instances are an ensemble of various distributions. We propose to use cluster analysis to utilize vocabulary embedding and to obtain a bag space vector representation. The algorithm works as follows:

1. Extract all training instances $\mathbb{X} = \{\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \ldots, \mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3}, \ldots, \mathbf{x}_{M,N_M}\}$

2. Find the optimal number of clusters k:
   The trade-off between quality criterion and the number of clusters is a common issue for clustering. For DBSCAN we utilize a simple non-parametric heuristic and locate the elbow as a maximum of the second difference of the trade-off smoothed function (see example in Fig. 4.5). Moreover, we have experimentally discovered that the parametrization of the clustering has only a minimal impact on the procedure. Therefore, our approach preserves the algorithm hyper-parameter-lightweight without a loss of quality.

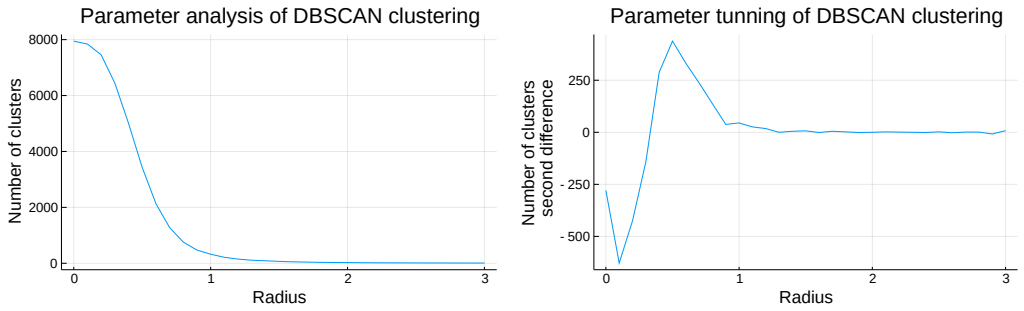


Figure 4.5: The optimal radius for clustering (so-called elbow) is determined as a maximum of the second difference of the smoothed criterion function. Example for DBSCAN and CorelBeach data set.

   For the k-means clustering, the above-mentioned heuristic can be applied with the number of clusters vs e.g. Ward's criterium [154]. However, it does not deliver sufficient performance and thus we performed empirical analysis and approximated the optimal number of clusters with high precision across all data sets as $\sqrt{\text{Nr. of instances}}$.

3. Perform clustering $\mathbb{C} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k\}$ (by default, we use DBSCAN and k-means)

4. Convert each bag $\mathbf{X}_i$ to k-dimensional vector representation $\tilde{\mathbf{x}}_i$ such that:

$$\tilde{\mathbf{x}}_i^{(l)} = \sum_{j=1}^{N_i} \begin{cases} 1, & \text{if } \mathbf{x}_{i,j} \in \mathbf{C}_l \\ 0, & \text{otherwise} \end{cases} \quad ; \quad \begin{matrix} i = \{1, \ldots, M\} \\ l = \{1, \ldots, k\} \end{matrix} \tag{4.2}$$

   In other words, each element of the representation vector indicates how many instances of the bag belong to the relevant cluster. Note that the ℓ_1 norm of the representation vector is equal to the number of instances in the original bag.

Generating bags in vector representation: First, the representations are normalized such that $\forall \tilde{\mathbf{x}} \in \tilde{\mathbf{X}} : \|\tilde{\mathbf{x}}\|_{L_1} \in (0, 1]$. This is carried out by dividing all elements of each vector by a constant *maxCardinality* which is obtained as the cardinality of the largest bag from $\mathbf{X}$. Then, we utilize the modified Parzen window estimation (see Sect. 2.2.1) to generate new samples $\{\tilde{\tilde{\mathbf{a}}}_1, \tilde{\tilde{\mathbf{a}}}_2, \ldots, \tilde{\tilde{\mathbf{a}}}_L\}$.

Some of the generated samples do not follow the assumptions of the input in two ways. First, some elements in the vectors could result in negative numbers because of the nature of the generator that is not bounded. This occurs very often and the interpretation of such a vector is not defined thus we solve this issue by clamping the negative elements to 0. Second, some vectors could overflow the interval $\|\tilde{\mathbf{x}}\|_{L_1} \in (0, 1]$. This behavior is rare and its representation is clearly defined. This propriety allows the generator to create a larger bag than the largest in the training set.

Generating MIL representation: Before we generate instances, we need to determine how many instances belong to each cluster for each bag. We construct $\{\tilde{\mathbf{a}}_1, \tilde{\mathbf{a}}_2, \ldots, \tilde{\mathbf{a}}_L\}$ where $\tilde{\mathbf{a}}_i^{(l)}$ represents number of instances from l-th cluster of i-th bag.

$$\tilde{\mathbf{a}}_i^{(l)} = \sum_{1}^{maxCardinality} \begin{cases} 1, & \text{if } \tilde{\tilde{\mathbf{a}}}_i > \text{rand}(0,1) \\ 0, & \text{otherwise} \end{cases} \quad ; \quad \begin{matrix} i = \{1, \ldots, L\} \\ l = \{1, \ldots, k\} \end{matrix} \tag{4.3}$$

This generator has a twofold effect. It determines the distribution of the instances into the clusters and it also randomly generates the bag size while preserving the average size of the training set.

Generating instances and bag representation: The auxiliary set can be finally generated in its final bag representation $\mathbb{A} = \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_L\}$ with generating instances from the distributions of the corresponding cluster. We utilize modified Parzen window estimation to generate samples in the distribution of the corresponding cluster. The bag level AUX is ready once the instances are generated.

Summary of the MIL generative strategy: The strategy follows well both instance-level and bag-level distribution in the instance space while providing a generative function on both levels. Similarly, it follows and generates cardinality.

4.4.6 Combining Generative Strategies

We have proposed four diverse strategies, that can either be utilized standalone or can be combined. The combination is simply carried out by executing more strategies standalone and aggregating the outputs. The combinations are expected to provide more diverse coverage of the MIL space and achieve better performance. The selection of strategies is more discussed in the experimental section 4.5.2.

4.4.7 MIL Neural Network Structure

We select to adopt the MIL NN paradigm proposed in [138] from all the available models (see Sect 4.2.5), because it delivers the best fit to our requirements. The

model is sophisticated enough to provide instance-level and bag-level modeling and also it is the best parallel to NN utilized for non-MIL AD.

The authors of [138] present an approach to multi-instance learning using neural networks to transform both instance-level and bag-level representations. A deep neural network $h : \mathbb{R}^d \to \mathbb{R}^u$ is used to transform instances, followed by an aggregation function $g : \mathbb{R}^u \to \mathbb{R}^u$ that is selected as one of the functions described in Sect. 4.2.3. Finally, a second deep neural network $f : \mathbb{R}^u \to \mathbb{R}^o$ is used to transform the representation of each bag.

Combining these functions gives the embedding function:

$$\mathcal{M}(\mathbf{X}) = f\left(g\left(\{h\left(\mathbf{x}_i\right) | \mathbf{x}_i \in \mathbf{X}\}\right)\right)$$

where d is dimensionality of the instance space, u is dimensionality of the latent space and o is dimensionality of the output (see the NN structure in Fig. 4.6).

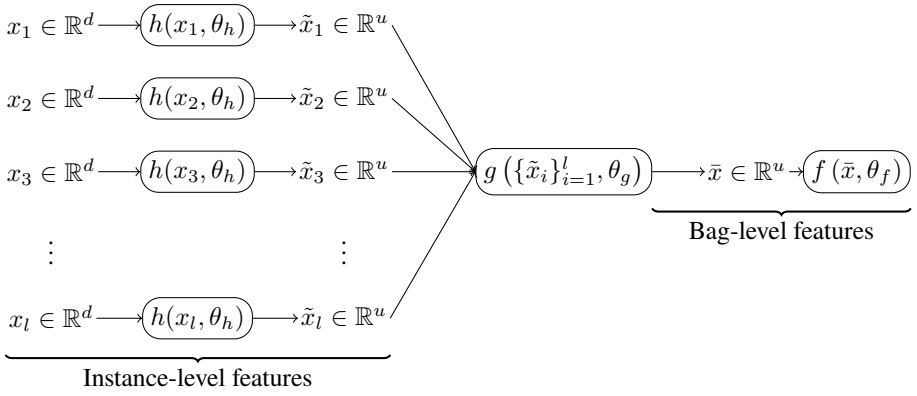


Figure 4.6: Structure of the MIL neural network proposed in [138]

The original design presented in [138] assumes (probably for the sake of simplicity) the aggregation function $g : \mathbb{R}^u \to \mathbb{R}^u$ operating with constant dimensionality. However, variable dimensionality is used in practice for example for the *meanmax* aggregation. Then analogically $g : \mathbb{R}^u \to \mathbb{R}^{\bar{u}}$ and $f : \mathbb{R}^{\bar{u}} \to \mathbb{R}^o$.

In our experiments, the number of neurons in each layer and the number of layers is a subject of hyper-parameter optimization (see Sect. 4.5.2) and the embedding is carried with the *meanmax* embedding (see Sect. 4.2.3). We utilize implementation from "Mill.jl framework: a flexible library for (hierarchical) multi-instance learning" [155].

4.4.8 Proposed Method Summary

In section 4.4 so far we have proposed and discussed in detail a methodology to train a neural model for multiple instance anomaly detection. Let us now summarize the resulting algorithm at a glance:

1. Input: training data set consisting of non-anomalous samples only (anomalous samples are not used for the training).

2. The auxiliary set is constructed from the training set with one of the four proposed MIL generative strategies (see Sect. 4.4.1) or their combinations where the selection of the strategy is a hyper-parameter. Note that the combinations are expectably more effective as we will confirm that fact in the experimental section 4.8.

3. For each auxiliary sample, its label (anomaly score) is computed with respect to the training set. To compute the anomaly score, we utilize kNN and one of the following MIL distance metrics: Maximum Mean Discrepancy, Minimal Hausdorff distance or instance-based distance (see experimental setup in Sect. 4.5.2)

4. Finally, the MIL neural network (see Sect. 4.4.7) is trained with the auxiliary set to predict its labels.

5. As a result, the trained network is able to predict the anomaly score for any input MIL sample.

4.5 Experimental Evaluation Setup

The evaluation of MIL anomaly detection is not standardized yet in the literature and the cross-method comparison does not exist due to the lack of prior art . The typical comparison scenario in the MIL AD prior art is to evaluate the method on a single data set. We provide the evaluation of the proposed method against the most relevant kNN on eight publicly available data sets and with three various MIL distance metrics.

4.5.1 Data Sets

The evaluation was done on a set of 8 data sets, made publicly available in [152]. These are:

- The "BrownCreeper" and "WinterWren" data sets. See [156]. A data set of bird songs where each bag represents a recording of one or multiple birds. Originally a 13-class data set, converted to binary classification data sets by selecting a target class.

- The "CorelAfrican" and "CorelBeach" data sets. See [157]. A data set of object images where each bag is an image, consisting of segments described by 4×4 patch features. Originally a 20-class data set, converted to binary classification data sets by selecting a target class.

- The "Musk1" data sets. See [158]. A data set of molecules where each bag is a set of the different shapes the molecule can fold into (so-called *conformers*). The goal is to predict whether a molecule has a musky smell or not. If at least one of the conformers of a molecule can cause it to smell musky, the molecule is positive.

- The "Mutagenesis1" and "Mutagenesis2" datasets. See [159]. A data set consisting of a drug activity prediction problem. There is an easy ("Mutagenesis1") and a hard ("Mutagenesis2") version.

- The "Protein" data set. See [160] and [161]. A data set consisting of protein annotations makes it a text categorization problem. The task is to decide whether a given pair should be annotated by a Gene Ontology (GO) code.

4.5.2 Experimental Setup

Sampling

The MIL data sets are two-class thus we consider the larger class (with respect to nr. of bags) as normal (non-anomalous) and the other as anomalous. Then we utilize 6x repetitive random sampling such that 75% of normal bags are used for training, 12.5% for validation, and 12.5% for testing. The anomalous bags are sampled 50% for validation and 50% for testing. We remind that anomalous samples are not used for training in anomaly detection.

Evaluation Metric

The accuracy is measured with the AUC of ROC as it is common for anomaly detection in literature. The advantage of this metric is the independence on specific thresholding and also the robustness for imbalanced data sets in contrast to e.g. PR-curve [162]. The selection of the metric for AD is more addressed in detail in Sect. 1.3.4.

MIL Metric

When considering conventional k-NN for MIL, the selection of the distance function (MIL metric) itself is a problem. Many such metrics exist and some could be more powerful for some problems and other way round. We include the three most common and frequently utilized metrics in MIL relevant literature that are Maximum Mean Discrepancy (MMD), Minimal Hausdorff distance (MHD) (see Sect. 4.2.2), and instance-based metric which is a direct application of kNN and the instance level paradigm (see Sect. 4.2.4). In our comparison, we utilize the metrics both for generating auxiliary data and also for competitive comparison.

Data Normalization

All data sets are normalized with respect to the training set such that the instances have the mean equal to 0 and std. dev. equal to 1.

Hyper-parameters and Tuning

In the experiment, some of the parameters are static either for their nature or because their optimal values could be found independently and some of the hyper-parameters are a subject of meta-optimization.

Let us start with the static. All of our experiments are based on k-NN thus the choice of k is essential. We observed $k = 3$ empirically as the best performing on average for the conventional approach thus we use it for both the conventional (competitive) and the proposed method. For the MMD, we use the Gaussian kernel with the size of 1 (data set is normalized) as this setup is mostly used in literature. For the instance-based metric we use the maximum as the aggregation function. We also have a static setup for the neural model training that is minibatch size = 180, maximum number of epochs = 10, and leaky ReLU activation function.

The hyper-parameter optimization is based on random search and validation and test data set such that the optimal hyper-parameters are selected with respect to the validation data set and the test data set is only used for final evaluation. Regarding the AUX generators, for each of the four strategies, it is randomly generated number $\{0, 1\}$ whether the strategy is used or not. As a result, both pure and mixed strategies are involved in the generative process. The joining of strategies is simple; each strategy runs independently and the resulting sets are joined. The generators also use hyper-parameters (see Sect. 4.4.1) *multiplication constant* as a random integer between 1 and 20 and *scale of the generator* as random between 0 and 1.1. The MIL generative strategy uses randomly DBSCAN or k-means clustering while determining the number of clusters is described in Sect. 4.4.1). The NN number of instance layers is between 1 and 3 and bag layers between 1 and 4. We consider all instance layers of the same size that is equal to the instance space dimension multiplied by a hyper-parameter between 1 and 3 and similarly for the bag layer size between 1 and 3.

Features of Validation Sata - Early Stopping, NN Initialization, and Efficient Random Search

In addition to the hyper-parameter search, we utilize the validation set to perform early stopping when training NN and also to avoid distorted models caused by incompetent initialization of NN by training three identical models with various initialization and selecting the best performing with respect to the validation data.

To achieve higher computational efficiency of the hyper-parameter search, we utilize evaluation skipping of the significantly poor hyper-parameter setups as follows. The performance of the model is estimated after the first evaluation round (out of six) and the experiment is stopped if the current performance is significantly lower than expected. More specifically, the AUC of the current setup is compared with the leading setup for the corresponding data set and metric and it must achieve at least 90% of the leading score to be fully evaluated with multiple evaluations.

4.6 Experimental Evaluation of Accuracy

In this work, we evaluate the proposed method against the one-class kNN over 8 data sets and 3 MIL metrics. Evaluating methods over multiple data sets can be done in many ways and thus we provide a comparison from various points of view in accordance to [97].

In Sect. 4.6.1, we provide the complete results (see Tab. 4.1) where we evaluate the count of wins, the count of significant wins, and averaging across data sets. The significance of wins is analyzed with confidence intervals (see Fig 4.7). Note that we particularly provide a pairwise comparison for each data set and metric.

In Sect. 4.6.2 we consider the metric as a hyper-parameter and thus we select the optimal metric for each method and perform the comparison under the such an assumption. In addition to the above-mentioned statistics, we also provide the Wilcoxon signed-rank test over the entire experiment.

4.6.1 Evaluation Over All Metrics

The full experimental results are provided in table 4.1. We compare the methods for each data set and metric separately. First of all, let us comment on the count of wins where the proposed method outperforms the one-class kNN in 19 cases while the kNN wins in 5 cases. The table also provides the count of significant wins from the confidence interval analysis at 0.95 level (see Fig 4.7). The neural method wins significantly in 10 cases while the kNN does not have any significant win. We also provide the averaged scores across all data sets in order to exhaust all the possibilities of comparison despite the fact it is the less reliable one. Anyway, the proposed method achieved average AUC of 80.5 while the kNN earn 71.2.

To summarize, the proposed method outperforms the kNN in all available assessments. An aggregated statistical test is not provided for the evaluation across various metrics because the single experiments (rows of the table) are not independent. More specifically, each data set is used for three experiments.

4.6.2 Evaluation for Best Metrics

Another point of view on the evaluation of the experimental results is to consider the MIL metric to be a hyper-parameter. In other words, each data set is now assessed once such that each method is represented by the score achieved by the best-performing MIL metric. Theoretically, such an approach corresponds to the real-world scenario where all the conditions can be fine-tuned according to the needs of the algorithm (neglecting the complexity of MIL kNN).

The results are provided in Tab. 4.2 which measures similar statistics as Tab. 4.1 but is more compact. In general, the neural method is better in all provided measures. However, this comparison is more benevolent for kNN such that the difference between the methods is less significant. Especially the count of significant wins is

Set	Metric	Neural	kNN.	Win	Signif.
BrownCreeper	Instance dist.	90.8	73.1	1	1
BrownCreeper	Min. Hausdorff	87.2	79.8	1	0
BrownCreeper	MMD	90.9	91.7	-1	0
CorelAfrican	Instance dist.	80.9	49.4	1	1
CorelAfrican	Min. Hausdorff	82.1	59.3	1	1
CorelAfrican	MMD	80.1	61.9	1	1
CorelBeach	Instance dist.	97.1	88.1	1	1
CorelBeach	Min. Hausdorff	93.7	76.3	1	1
CorelBeach	MMD	96.6	90.5	1	1
Musk1	Instance dist.	84.3	91.4	-1	0
Musk1	Min. Hausdorff	94.9	89.1	1	0
Musk1	MMD	98.5	59.1	1	1
Mutagenesis1	Instance dist.	72.9	72.8	1	0
Mutagenesis1	Min. Hausdorff	73.7	68.6	1	0
Mutagenesis1	MMD	72.9	50.7	1	0
Mutagenesis2	Instance dist.	61.1	73.6	-1	0
Mutagenesis2	Min. Hausdorff	61.1	51.0	1	0
Mutagenesis2	MMD	52.8	92.4	-1	0
Protein	Instance dist.	65.5	51.3	1	0
Protein	Min. Hausdorff	61.6	58.0	1	0
Protein	MMD	54.1	55.9	-1	0
WinterWren	Instance dist.	92.8	92.1	1	0
WinterWren	Min. Hausdorff	93.9	65.0	1	1
WinterWren	MMD	92.1	66.6	1	1
Aggregation		AVG	AVG	$19-5$	$10-0$
		80.5	71.2	14	10

Table 4.1: Accuracy comparison of the proposed method and kNN both operating under three metrics. Columns *Neural* and *kNN* provide AUC ROC scaled between [0,100], *Win* represents winning method (1 for Neural and -1 for kNN), *Signif* represents significant winning method (1 for Neural and -1 for kNN) according to the confidence intervals at 0.95 level (see Fig 4.7).

2 (25% of all cases) while it is 10 (42% of all cases) in the full comparison in Sect. 4.6.1. Similarly, the difference between the averaged scores is 1.5 while it is 9.3 in Sect. 4.6.1.

The Wilcoxon signed-rank test [163] can be utilized when the experiments (rows of the table) are independent. As a result, the test does not prove the statistical significance at the level of 0.95.

To conclude, despite the assessment for the best-performing MIL metric being more profitable for kNN than the previous comparison, the kNN is outperformed by the proposed model in accordance with all provided statistics. The difference is statistically significant for two of eight data sets and not significant overall.

Set	Neural	kNN	Win	Signif.
BrownCreeper	90.9	91.7	-1	0
CorelAfrican	82.1	61.9	1	1
CorelBeach	97.1	90.5	1	1
Musk1	98.5	91.4	1	0
Mutagenesis1	73.7	72.8	1	0
Mutagenesis2	61.1	92.4	-1	0
Protein	65.5	58	1	0
WinterWren	93.9	92.1	1	0
Aggregation	Avg	Avg	$6-2$	$2-0$
	82.85	81.35	4	2

Table 4.2: Accuracy comparison of the proposed method and kNN, both with the best-performing MIL metric. Columns *Neural* and *kNN* provide AUC ROC scaled between [0,100], *Win* represents the winning method (1 for Neural and -1 for kNN), *Signif* represents significant winning method (1 for Neural and -1 for kNN) according to the confidence intervals at 0.95 level (see Fig 4.7).

4.6.3 Conclusion on Accuracy

We evaluate the experiments from two points of view (see Sect. 4.6) and provide a number various of statistics. In both cases, all statistics clearly show that the proposed neural method outperforms kNN. The confidence intervals (see Fig 4.7) indicate statistically significant dominance of the proposed method for some of the data sets. In the second assessment, we also utilize the Wilcoxon signed-rank test which does not prove statistical significance over all data sets.

The neural model demonstrated superior performance over the kNN to a considerable extent which is interesting when taking into account that the kNN is a source-detector to be replicated. There are several explanations of this phenomenon. First of all, we remind similar behavior for experiments on vector data in Chapt. 2 where the approximation has a denoising effect resulting in a more robust detector. In the context of MIL, the regularization feature of the neural model becomes even more important in the face of the complexity of the source data and can prevent overfitting of the model. Another regularization factor is the computation of the AUX set using strategies. The analysis of the strategy's importance (see Sect. 4.8) evaluates the simplest *single instance bag* strategy as the most valuable and thus utilized in the algorithm. This strategy itself brings a noticeable regularization and simplification due to its nature. Furthermore, MIL data sets benefit from regularization, particularly due to the lower ratios of sample numbers to dimensionality, compared to vector data sets.

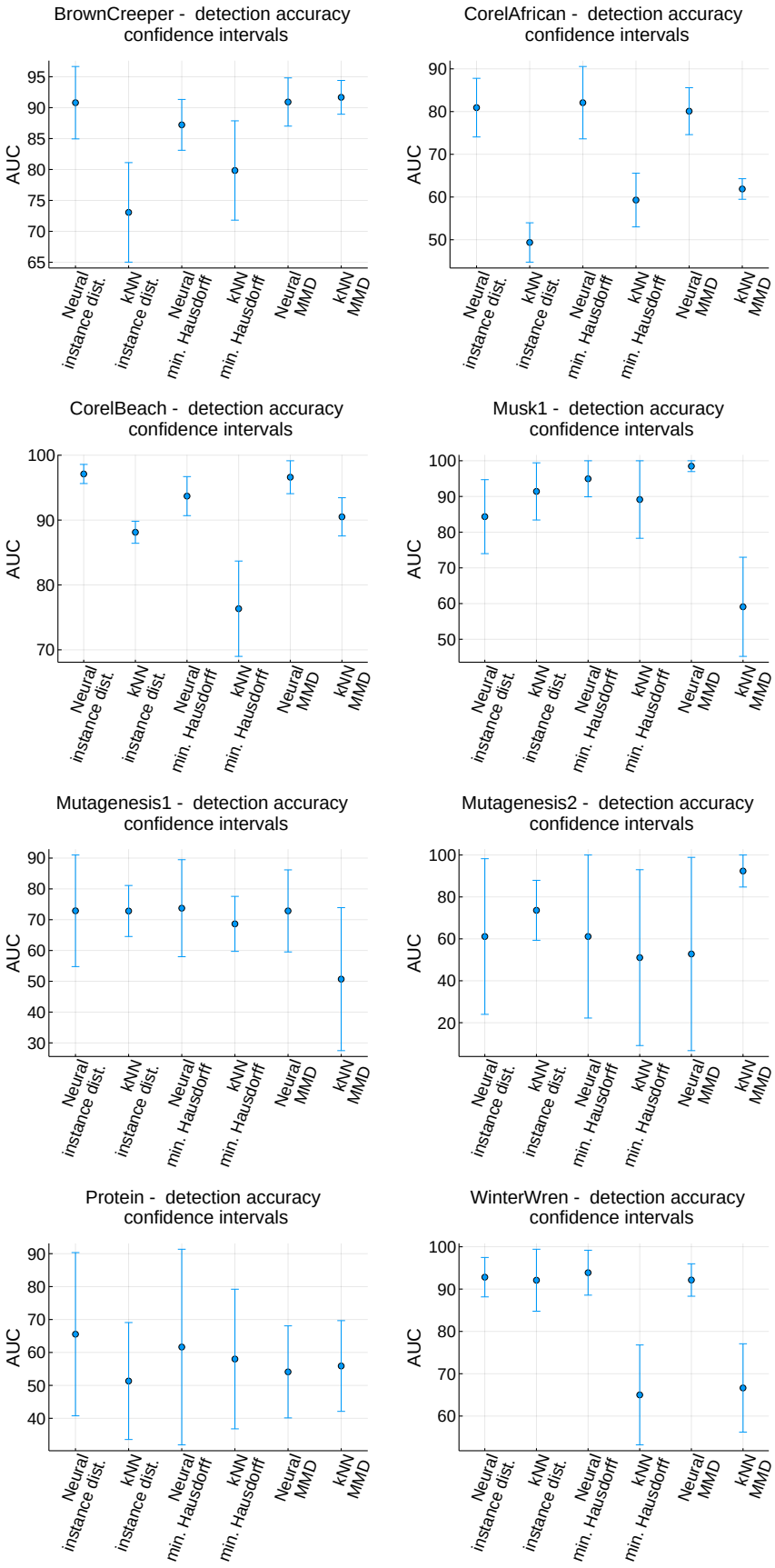


Figure 4.7: Confidence intervals at 0.95 level. AUC scaled to [0,100] correspondingly to Tab. 4.1.

4.7 Experimental Evaluation of Inference Speed

Inference speed is an essential advantage of the proposed algorithm since the main goal is to achieve comparable accuracy while reducing the inference time. In this section, we provide inference speed analysis with respect to dimensionality, size of the training set, and cardinality for three data sets with various properties.

4.7.1 Evaluation Analysis

The inference speed analysis is provided in Figs. 4.8, 4.9, and 4.10. The plots, in addition to providing resulting inference speed for the analyzed data sets, depict the time complexity with respect to mentioned properties. More specifically the right edge of the plots corresponds to the inference time of the full data set and the other inference speed measure is carried out with a down sampled or simplified data accordingly to the analysis. The dimensionality is sampled with the step of size 1 and the other statistics are sampled to create a 10-step plot.

The time measurement is carried out in a controlled environment and each measure is executed multiple times and averaged to achieve more reliable statistics.

Let us discuss the inference speed provided in Figs. 4.8, 4.9, and 4.10. First of all, the instances ratio has an insignificant impact on the inference speed for all presented data sets and methods. The size of the training set (nr. of bags) analysis shows that the neural model has constant inference time with a growing training set while the kNN-based models have a growing trend. All of the presented methods have a growing trend with dimensionality.

The proposed methodology has resulted in a significant reduction in inference time for the original data sets, as measured on the right-hand edge of the plots (Figs. 4.8, 4.9, and 4.10). The reduction magnitude is a problem and metric dependent, ranging from 42 to 6300 times faster inference in the provided experiments. The most significant reduction occurs for the Corel Beach data set and MMD metric from 12,6 to 0.002 seconds and the less significant occurs for the Brown Creeper dataset from 1.06 to 0.025 seconds.

4.7.2 Discussion

The proposed method mainly benefits from the constant inference speed with respect to the number of training samples (bags) and thus is suitable for larger data sets typically used in industrial and real-world applications. This is consistent with the fact that the biggest time saving is achieved on the Corel Beach data set, which is also the biggest in terms of number of samples. Note that the Corel Beach data set was trained with 1425 samples, which is only a small fraction of how many samples can be used in an industrial environment. As a result, the expected inference reduction is even more significant for industrial applications.

While we acknowledge that our analysis is limited to specific cases and it is important to note that extrapolating from our plots could be misleading. For example, while our results show that the neural inference time grows faster with dimensionality than the kNN MMD method, this does not necessarily imply that kNN MMD is advantageous for data with larger dimensions. In practice, datasets with larger dimensions often come with larger training sets, which can confer an advantage to the neural method. On the other hand, it is theoretically possible for datasets with high dimensionality and a small number of samples to exist, in which case kNN MMD would be more time-efficient. However, such datasets are generally problematic in machine learning, as they are prone to overfitting and can lead to unreliable results.

4.7.3 Conslusion on Inference Speed

In all the simulated cases, the order of methods according to the speed remains consistent. The neural detector is the fastest by orders of magnitude while the choice of the source-detector for the neural model does not affect the inference speed (see the identical plots for neural models). The instance-based metric is the fastest of the conventional kNN approaches followed by the minimal Hausdorff distance and lastly with MMD. Note that for each data set, the right edges of the three plots are identical and represent the inference time on the original data set. The plots have logarithmic scaling and show a striking difference in the inference speed between the proposed method and the baseline. Depending on the data and metric, the proposed method achieved up to 6300 times faster inference in our experiments, and in the industrial setup, the speed-up is expected even more remarkable.

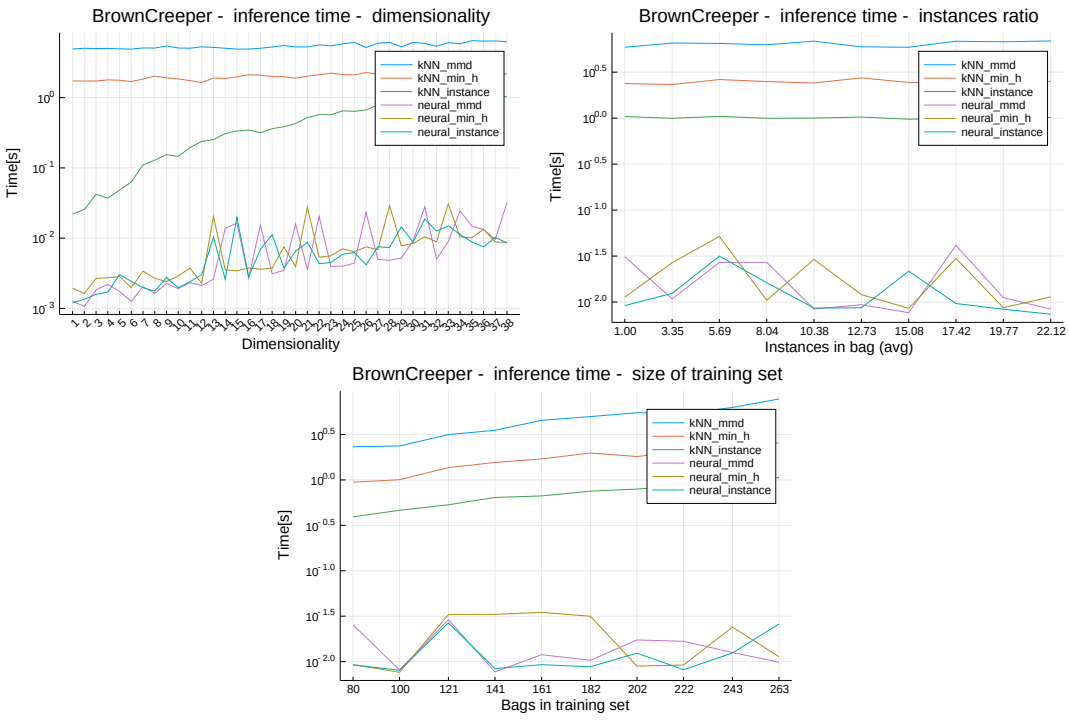


Figure 4.8: Inference time for BrownCreeper (log. scaling)

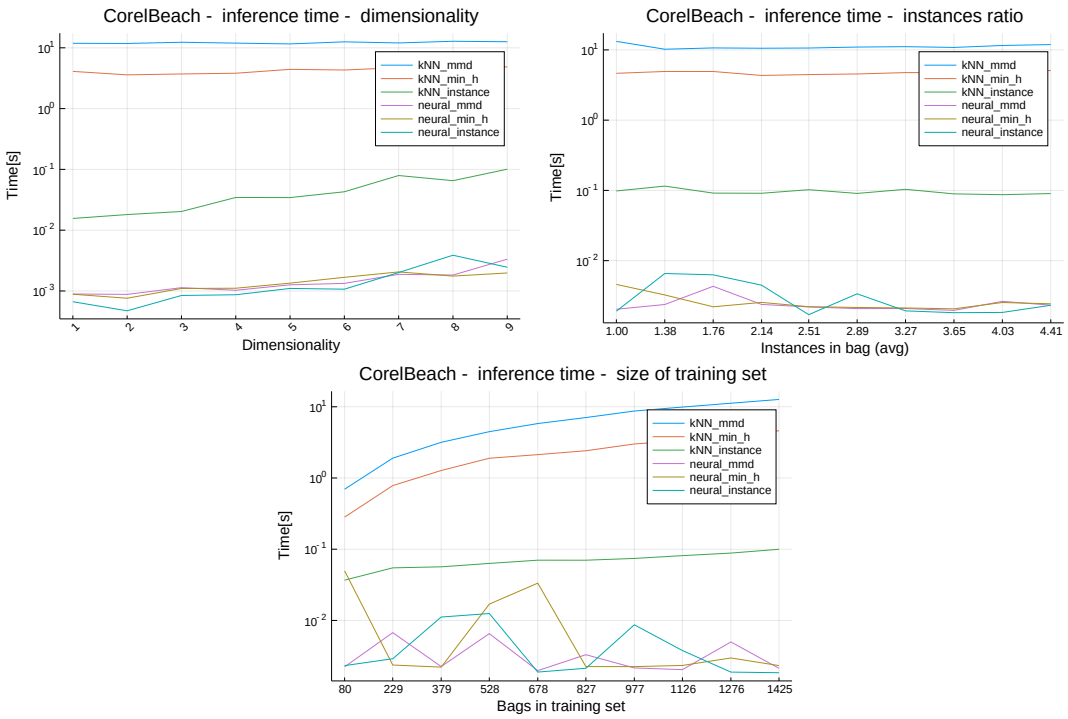


Figure 4.9: Inference time for CorelBeach (log. scaling)

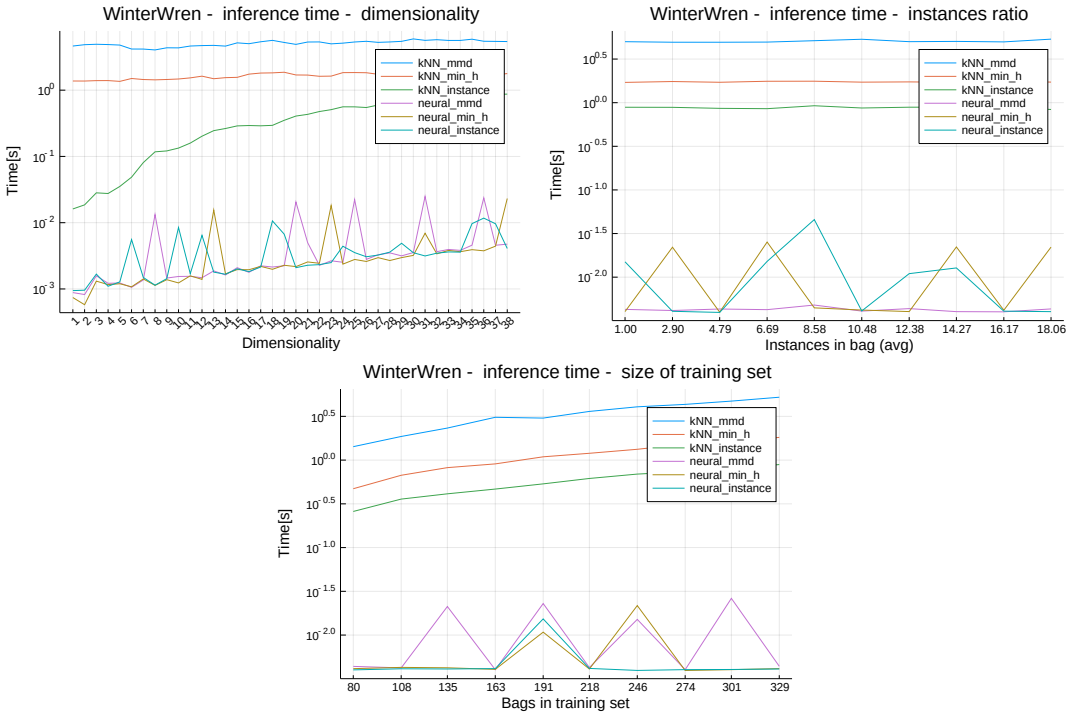


Figure 4.10: Inference time for WinterWren (log. scaling)

4.8 Evaluating Auxiliary Set Generative Strategies

We have proposed four diverse strategies to generate the AUX set (see Sect. 4.4.1) and the strategies are heuristically combined in our experiments. In this section, we statistically analyze the efficiency of the strategies from three perspectives. First, we discover what strategy combinations are heuristically found as the optimum in our experiments. Secondly, we investigate the value of each strategy in the combined strategy scenario, which is the experimental setup. Finally, we evaluate the strategies in a pure scenario where the strategies are not combined. Note that the pure strategy setup is also a member of the strategy combination space.

Clarifying the Data Source for the Plots

Numerous plots are provided in this section to demonstrate the score distribution of our experimental shots depending on the selected strategy and data set. Our experimental design naturally utilizes multiple validation procedures (see Sect. 4.5.2) to evaluate the model and its performance. However, the hyper-parameter search heuristic stops the experiment after the first evaluation if the performance is significantly lower than expected. This saves a lot of computational power without the loss of generality. However, full multiple evaluations are computed only for better hyper-parameter setups and thus such a group of experiments is strongly biased for overall analysis. To resolve this problem, we use the score of a single evaluation instead of averaging multiple evaluations and thus we can include all experimental shots ever computed to obtain the true score distribution across all experiments. Moreover, this approach is not negatively affected by the averaging of evaluation scores and reveals the raw distributions of AUC scores. Remark: this approach is only used for creating the distribution plots in this section.

Optimal Combination of Strategies

The optimal combination of strategies for each data set and metric is given in Tab. 4.3 according to our random search heuristic. Expectably, the optimal strategy combinations are data and metric-dependent while some of the strategies are more beneficial than others in general. The most popular *single instance bag* strategy is ideally utilized in all cases except two, both for *Corel Beach* data set. The second favorite *MIL generative* strategy is widely utilized with partial exceptions mainly for *MMD* metric or *Winter Wren* data set. *Noise to instances* are utilized for over half of the problems and *genetics* for less than half without a significant relation to any specific data set or metric.

4.8.1 Overall Analysis of Mixed Strategies

We performed a number of experiments in order to cover the hyper-parameter space. As a result, we can benefit from the experimental data to create a comparison for

Set	Metric	Noise to instances	Genetics	Single-inst. bags	MIL gener.
BrownCreeper	InstanceDistance	1	0	1	1
BrownCreeper	MinimalHausdorff	0	0	1	1
BrownCreeper	Mmd	0	1	1	1
CorelAfrican	InstanceDistance	1	0	1	1
CorelAfrican	MinimalHausdorff	0	0	1	1
CorelAfrican	Mmd	1	0	1	0
CorelBeach	InstanceDistance	1	0	1	1
CorelBeach	MinimalHausdorff	1	1	0	1
CorelBeach	Mmd	1	1	0	1
Musk1	InstanceDistance	1	0	1	1
Musk1	MinimalHausdorff	1	0	1	1
Musk1	Mmd	1	0	1	0
Mutagenesis1	InstanceDistance	1	0	1	1
Mutagenesis1	MinimalHausdorff	0	1	1	1
Mutagenesis1	Mmd	1	1	1	1
Mutagenesis2	InstanceDistance	0	1	1	1
Mutagenesis2	MinimalHausdorff	0	0	1	1
Mutagenesis2	Mmd	0	1	1	0
Protein	InstanceDistance	0	1	1	1
Protein	MinimalHausdorff	1	1	1	1
Protein	Mmd	1	1	1	1
WinterWren	InstanceDistance	1	0	1	0
WinterWren	MinimalHausdorff	0	0	1	0
WinterWren	Mmd	1	0	1	1
SUM		15	10	22	19

Table 4.3: Optimal combination of strategies according to the proposed experiments. The numbers 1 and 0 represent whether the strategy is included (1) or not (0) into the optimal combination.

each strategy. The strategy combination mechanism involves each strategy with a 0.5 probability in the experimental shot. In other words, each strategy roughly covers half of the experimental shots.

The concept is to split the experimental statistics in two groups with respect to the analyzed strategy and compare the distribution of the scores for both groups. Note that the distribution of other (non-analyzed) strategies will remain similar in both groups.

Let us start with a more general comparison given in Fig. 4.11. The comparison for each strategy will be referred to as *plot* each consisting of two *distributions*. First of all, it is apparent that the distribution is multi-modal because the data are aggregated from all the data sets that have various score distributions. Moreover, the plots show only a minimal difference in the distributions for most of the strategies except *single instance bag*. In general, the figure depicts a slight advantage of all strategies, based on the fact that the distribution for *used* have higher density by the

right edge of the plot. This is consistent with the expectation that the best results are achieved with a mixture of more strategies. The figure also shows the noticeable benefit of the *single instance bag* strategy because both of the distribution peaks reach higher scores for the *used* distribution. This is consistent with the fact that this strategy is recommended for 22 of 24 problems according to Tab. 4.3. Anyway, the analysis for each data set separately will provide a more reliable comparison.

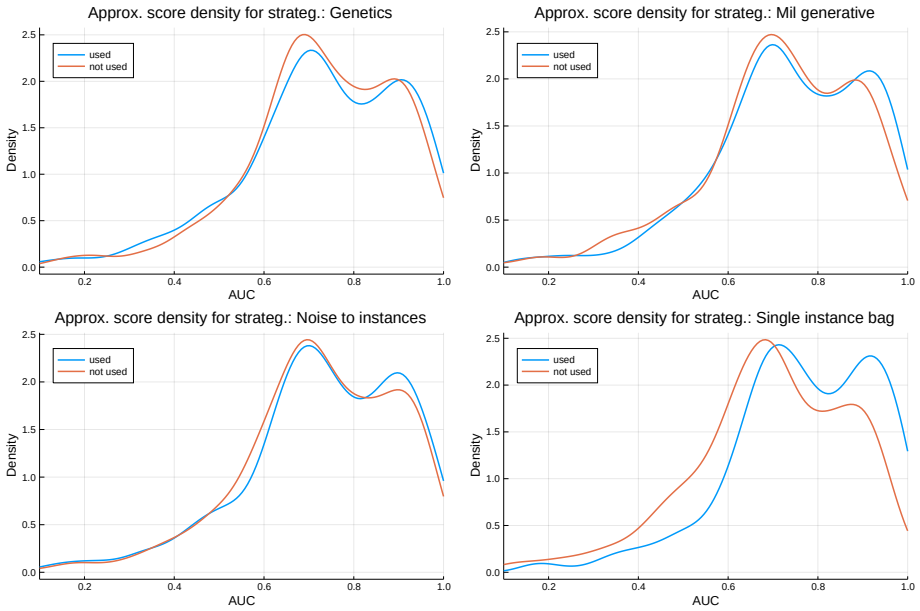


Figure 4.11: Overall strategy performance comparison across all data sets. For each strategy, we compare the distribution of performance of the experimental shots where the strategy was utilized with those that were not.

Analysis of the distributions for each data set separately is provided in Fig. 4.12 and delivers probably the most detailed insight into the differences among the strategies. Besides that, it also reveals the score distribution characteristics for each of the data sets. These differ across the problems and among others, it also illustrates the robustness of the algorithm (Compare e.g. compact distribution of Corel Beach with Mutagenesis2).

Let us highlight the most interesting observations from Fig. 4.12. The figure clearly shows the positive impact of *single instance bag* strategy for Brown Creeper. In addition, when the *single instance bag* and/or *MIL generative* strategy is utilized, the AUC is typically between 0.7 and 1.0. Another way round, the scores are between 0.4 and 0.9 without *single instance bag* strategy. Less significant but similar behavior show *single instance bag* strategy also for Musk1.

A very nice demonstration of combination power is provided for Protein where the combination of *genetics*, *MIL generative*, and *single instance bag* is mandatory to break the 0.7 AUC threshold. The plot for *noise to instances* shows a negligible effect of this strategy at the highest score level (see the similar distributions around 0.85 AUC). These observations are consistent with Tab. 4.3 such that the combination of the three strategies is recommended for all metrics for Protein and *noise to instances* is recommended two times (out of three).

Clarifying the interpretation of the plots: When comparing e.g. maximum AUC for Protein in Fig. 4.12 which is, as mentioned above, roughly about 0.85 with the main results in Tab 4.1 which are 0.54, 0.61, and 0.65 (depending on the utilized metric), we observe that it does not match. This is caused by using data from experiments without multiple validations and averaging as described in Sect. 4.8. For the sake of completeness we also provide the plots computed on multiple validated experiments, that correspond to the main experimental results in Sect. 4.6 and Tab. 4.1; see the appendix (Sect. A.1). The difference between the plots computed with a single evaluation and multiple evaluations typically corresponds to the size of the confidence intervals presented in Fig. 4.7.

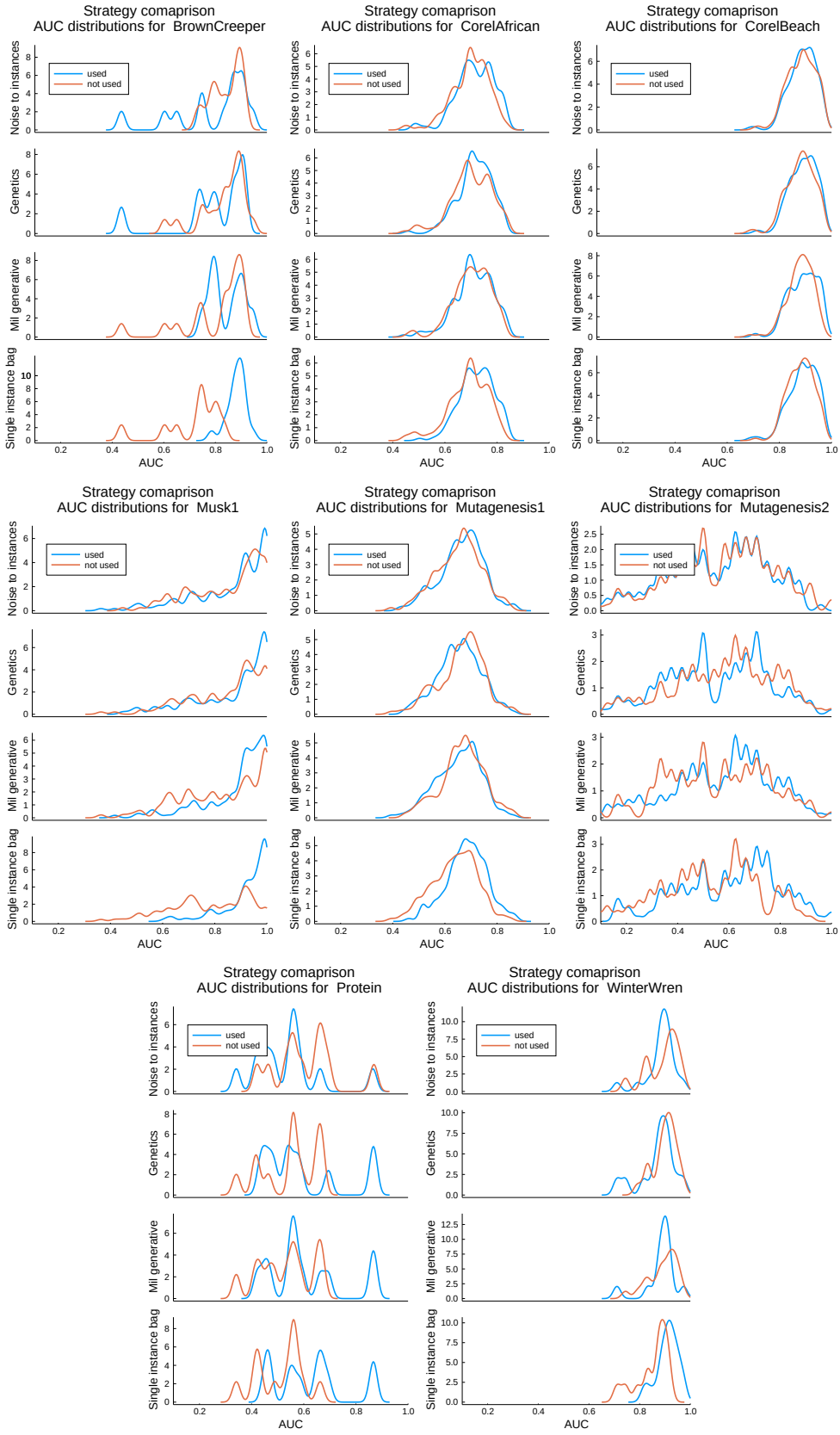


Figure 4.12: Overall strategy performance analyzed separately for each data set.

4.8.2 Overall Analysis of Pure Strategies

In this section, we discuss the performance of the stand-alone strategies, despite the fact that the best-performing setup is achieved by mixing the strategies.

The approximated score distributions across all data sets are given in Fig. 4.13 where *single instance bag* strategy shows considerably better potential than other methods. However, as discussed in the previous section, analysis across all data sets might be misleading and too uncertain to make any conclusions.

Analysis of the AUC distributions separately for each data set (see Fig 4.14) is more relevant. Let us comment on some interesting observations. The Brown Creeper and Winter Wren data sets have very similar distributions where *single instance bag* outperforms others. The strategy also seems to be most beneficial for Protein, Corel Beach, Mutagenesis1, and partly Musk1. For Musk1, *MIL generative* is the best-performing strategy. All the interesting observations are consistent with the analysis of mixed strategies (see Sect. 4.8.1).

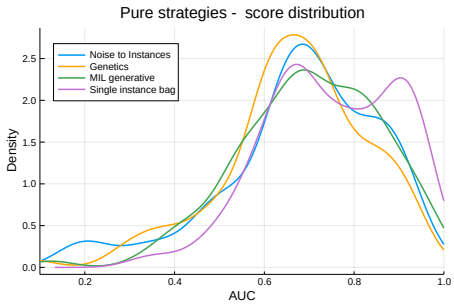


Figure 4.13: Overall pure strategy performance across all data sets

4.8.3 Conclusion on Strategy Selection

We analyzed the performance of the strategies in two perspectives, pure and mixed and we also compare the observations with the results of the experimental heuristic that search the space of both pure and mixed strategies.

In general, the optimal combination of strategies is problem-dependent. However, it has been shown that the fusion of strategies provides better performance than a strategy operating alone. The score distribution analysis provided conclusions equivalent to the quality estimate given in Tab. 4.3 so we can assess the strategies accordingly. The most important role plays the *single instance bag* strategy which is by far the most excellent then follows *MIL generative*, *Noise to instances*, and last *genetics*.

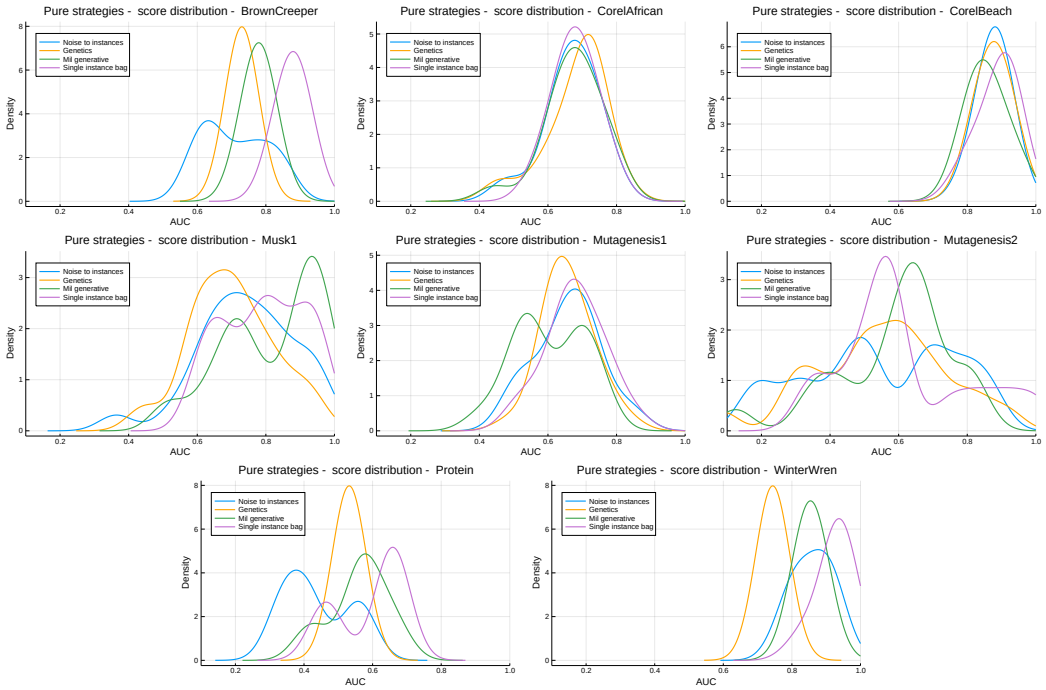


Figure 4.14: Overall pure strategy performance analyzed separately for each data set.

4.9 Summary

We address the problem of efficient anomaly detection on structural data in Multiple Instance Learning setting. Starting with a baseline one-class kNN; a direct application of the bag space paradigm which is commonly utilized in the literature, we emphasize its extremely high computational complexity.

For non-MIL learning, SNN has proved its ability to replicate anomaly detectors via neural networks with better or comparable accuracy while reducing computational complexity. We have successfully adapted SNN to the MIL problem by addressing two challenges. First, generating AUX data set in the MIL space; second, adopting MIL neural network.

Generating the AUX data set is an essential step of the SNN algorithm and we recognize it as the major contribution. The generative task in MIL space is incomparably harder than in standard vector space. To solve this, we propose four different strategies to generate the MIL samples and we show that their combinations are diverse enough to successfully generate the space.

We provide broad evaluation across eight various data sets and three MIL distance metrics. The comparison of the proposed method and the baseline detector, which is also utilized as a source-detector, is carried out from multiple perspectives and with all available assessment metrics and mechanisms. Depending on the point of view, the proposed method has comparable performance or even outperforms the baseline. Roughly explained, the proposed method delivers better accuracy for the majority of the data sets and setups while the statistical significance is reached only

in some cases and the overall significance is not proved. Anyway, the accuracy goal was to achieve at least comparable performance which is fulfilled.

The neural model enables a reduction of inference time in comparison to the baseline by orders of magnitude. We provide extensive experimental measures to depict the complexity with respect to training set size, dimension, and average cardinality. We demonstrate that the proposed method is significantly faster under all circumstances and, furthermore, in the original setup of the data sets, the proposed method is approximately $100\times$ faster. Note that the difference would be even more striking for larger industrial data sets.

We also perform the analysis to reveal the benefit of each proposed auxiliary data set generation strategy. The strategies are analyzed with two methodologies and we consider their combinations and also the strategies standalone. All the analyses confirm that some strategies are more valuable than others, however, it is shown that combining the strategies is the ultimate approach that leads to success. This is not surprising as the mixtures provide more diverse output and thus cover the space better.

Conclusion and Outlook

This work provides research in three interrelated areas featuring innovative paradigms and approaches. First, we addressed the problem of efficient anomaly detection by designing a novel neural-based model capable of approximating high-accuracy density-based detectors. The resulting model benefits from the synergy between the high-accuracy density-based detectors and efficient neural-based models. Second, we proposed an efficient schema for continuously updating the model with high efficiency and accuracy. Third, we focused on its application to the field of multiple instance learning.

Summary

We proposed surrogate neural network model (SNN) for neural anomaly detection in Chapt. 2 while aiming to the industrial application where the neural networks are preferred due to the inference speed and constantly improving HW and SW support. We demonstrated that the proposed model provides comparable performance to the best-performing models while reducing inference speed by orders of magnitude. We successfully evaluated the concept with publicly available data sets from the UCI repository (evaluated over 64 problems) and consequently, we performed the experiment with an industrial-based data set from the field of computer security where the proposed method outperforms the state-of-the-art models (AE, GAN, GANomaly, IF). In addition, the method provides easy and memory-efficient deployment in the big-data industrial setup, deployment in embedded systems, and even simplifies the use of detector ensembles.

Motivated by the needs of the real-world application, we developed an efficient update procedure for the SNN model in Chapt. 3. We took the advantage of the algorithm's auxiliary data set properties and utilized compression via Gaussian mixture models to deliver a memory-efficient updating algorithm without significant loss of accuracy. Such an efficiency would be difficult to achieve with e.g. kNN. The method is experimentally compared with relevant competitors in two scenarios; with and without concept drift with satisfactory results. In addition, we demonstrated that the sequentially trained model has comparable performance to the offline variant of the model evaluated in Chapt. 2.

Finally, we adopted SNN to the field of multiple instance learning (MIL) based anomaly detection where its inference time advantage is even more striking, as illus-

trated on comparison to MIL kNN. To do so, we addressed the problem of adopting the MIL neural network and more importantly, generating the space of MIL when creating the MIL auxiliary set. Despite the fact that generating MIL space remains an open problem, we have developed four distinct strategies and shown that their combination is capable of providing sufficient and relevant coverage of the space. We provided extensive evaluation across MIL metrics and publicly available data sets while the proposed method outperforms the competitors and more importantly provides up to 6300 times faster inference for the largest evaluated data set. As a result, we delivered a novel neural-based model, opening new, yet computationally prohibited, opportunities to apply reliable anomaly detection to structured and large-scale data.

Future Challenges

Although the training of SNN via auxiliary data set is relatively efficient, it would be useful to further explore other options to train SNN. Would it be possible to find synergy between SNN and GANs as both models consist of similar logical parts, generator, and decision-maker? For example, would it be possible to boost GANs with another AD source-detector (e.g. kNN) via combination with SNN while using the original GANs generator? Would it be possible to boost or replace the existing auxiliary Parzen-based generator with a GAN-based generator? Would the employment of such generators lead to better results, especially in concert with existing generators, given that ensemble diversity has been found useful in Chapt. 4?

The problem of generating auxiliary set for MIL problems may offer even more research opportunity. Despite the fact that we have proposed four distinct generative heuristics the mixture of which provides sufficient coverage of the MIL space in our experiments, the fundamental problem of generating the entire MIL space remains open. Future research thus might aim to move beyond the heuristical approach and search for a comprehensive solution that can solve the problem more accurately.

As we have registered that the resulting SNN detector is able to outperform the source-detector, it might be beneficial to explore this phenomenon in depth. The hypothesis is that the accuracy improvement over the source-detector is caused by the regularization effect of the SNN model. For instance, it would be beneficial to estimate the potential accuracy gain of SNN without training SNN - from the training data and/or the source-detector. As a result, it would be useful to estimate for which problems it is convenient to train SNN even for small-scale problems, where the inference time is not a crucial factor. In other words, there might be an opportunity to consider the SNN as a regularizer on top of any anomaly detector.

For the updating procedure proposed in Chapt. 3, future research might focus on alternative, even more memory-efficient approaches to preserve information about historical data. Although the use of *Gaussian Mixture Model* has proven to be efficient, there might be a possibility to reconstruct the necessary information from the SNN model itself. The idea is to analyze the anomaly function provided by the SNN model to estimate the location of relevant areas in the space. In other

words, the task would be to approximate the training data set based on the anomaly score function. In case of success, this would result in further reduction of space complexity.

Main Contributions

The main contribution is providing verification to the hypotheses mentioned in the introduction by proposing a theoretical and practical solution to the three relevant problems, which resulted in the design of three completely novel algorithms:

1. We proved hypothesis No. 1 by proposing an entirely novel neural paradigm for anomaly detection, which combines density-based models with neural models and that outperforms existing neural models for applications in cyber-security.
2. We proved hypothesis No. 2 by implementing a solution for updating the proposed model to reflect the changes in the modeled environment while achieving extreme memory efficiency and preserving a high degree of model accuracy.
3. We proved hypothesis No. 3 by designing a novel neural-based model for anomaly detection in structural data. With up to 6300 times faster inference than prior art on the benchmark data and expectably more striking time saved in a real environment, we covered the gap in efficient and reliable AD for large-scale data. The method now enables applications that have yet been prohibited due to the extreme computational complexity.

Bibliography

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[2] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural Network Design*. Martin Hagan, 2014.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019.

[5] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Computer Vision – ACCV 2018*, C. V. Jawahar, H. Li, G. Mori, and K. Schindler, Eds. Cham: Springer International Publishing, 2019, pp. 622–637.

[6] V. Škvára, T. Pevný, and V. Šmídl, "Are generative deep models for novelty detection truly better?" 2018.

[7] X. Gu, L. Akoglu, and A. Rinaldo, "Statistical analysis of nearest neighbor methods for anomaly detection," *arXiv preprint arXiv:1907.03813*, 2019.

[8] L. Bergman, N. Cohen, and Y. Hoshen, "Deep nearest neighbor anomaly detection," *arXiv preprint arXiv:2002.10445*, 2020.

[9] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[10] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[11] S. Garcia, J. Derrac, J. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 417–435, 2012.

[12] S. Mittal, "A survey on optimized implementation of deep learning models on the nvidia jetson platform," *Journal of Systems Architecture*, vol. 97, pp.

428–442, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762118306404

[13] J. Amores, "Multiple instance classification: Review, taxonomy and comparative study," *Artificial Intelligence*, vol. 201, pp. 81–105, 2013.

[14] I.-Y. Weon, D.-H. Song, S.-B. Ko, and C.-H. Lee, "A multiple instance learning problem approach model to anomaly network intrusion detection," *Journal of Information Processing Systems*, vol. 1, no. 1, pp. 14–21, 2005.

[15] A. D. Ker and T. Pevnỳ, "The steganographer is the outlier: realistic large-scale steganalysis," *IEEE Transactions on information forensics and security*, vol. 9, no. 9, pp. 1424–1435, 2014.

[16] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[17] R. Yu, X. He, and Y. Liu, "Glad: group anomaly detection in social media analysis," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 2, p. 18, 2015.

[18] D. A. Reynolds, "Gaussian mixture models." *Encyclopedia of biometrics*, vol. 741, no. 659-663, 2009.

[19] "MS Azure machine learning documentation," https://docs.microsoft.com/en-us/azure/machine-learning/media/machine-learning-algorithm-choice/image8.png, accessed: 2018-10-10.

[20] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.

[21] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," accessed: 2017-08-30. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] "Medium - online publishing platform," https://medium.com/analytics-vidhya/unsupervised-learning-and-convolutional-autoencoder-for-image-anomaly-detection-b783706eb59e, accessed: 2022-12-09.

[23] T. Nakao, S. Hanaoka, Y. Nomura, M. Murata, T. Takenaga, S. Miki, T. Watadani, T. Yoshikawa, N. Hayashi, and O. Abe, "Unsupervised deep anomaly detection in chest radiographs," *Journal of Digital Imaging*, vol. 34, no. 2, pp. 418–427, 2021.

[24] "Wikipedia receiver operating characteristic," https://en.wikipedia.org/wiki/Receiver_operating_characteristic, accessed: 2017-03-10.

[25] J. Beck and E. Shultz, "The use of relative operating characteristic (ROC) curves in test performance evaluation," *Archives of Pathology and Laboratory Medicine*, vol. 110, no. 1, p. 13—20, January 1986. [Online]. Available: http://europepmc.org/abstract/MED/3753562

[26] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145 – 1159, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320396001422

[27] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006, ROC Analysis in Pattern Recognition. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786550500303X

[28] C. E. Metz, "Basic principles of ROC analysis," *Seminars in Nuclear Medicine*, vol. 8, no. 4, pp. 283 – 298, 1978. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0001299878800142

[29] C. Marrocco, R. Duin, and F. Tortorella, "Maximizing the area under the ROC curve by pairwise feature combination," *Pattern Recognition*, vol. 41, no. 6, pp. 1961 – 1974, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320307005067

[30] "Wikipedia precision and recall," https://en.wikipedia.org/wiki/ Precision_and_recall#/media/File:Precisionrecall.svg, accessed: 2022-12-12.

[31] C. D. Brown and H. T. Davis, "Receiver operating characteristics curves and related decision measures: A tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 80, no. 1, pp. 24–38, 2006.

[32] V. Raghavan, P. Bollmann, and G. S. Jung, "A critical investigation of recall and precision as measures of retrieval system performance," *ACM Transactions on Information Systems (TOIS)*, vol. 7, no. 3, pp. 205–229, 1989.

[33] T. Pevný, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, vol. 102, no. 2, pp. 275–304, 2016.

[34] V. Škvára, J. Franců, M. Zorek, T. Pevný, and V. Šmídl, "Comparison of anomaly detectors: Context matters," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[35] A. F. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong, "Systematic construction of anomaly detection benchmarks from real data," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, ser. ODD '13. New York, NY, USA: ACM, 2013, pp. 16–21. [Online]. Available: http://doi.acm.org/10.1145/2500853.2500858

[36] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study," *Data mining and knowledge discovery*, vol. 30, no. 4, pp. 891–927, 2016.

[37] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS one*, vol. 11, no. 4, p. e0152173, 2016.

[38] J. Brabec and L. Machlica, "Bad practices in evaluation methodology relevant to class-imbalanced problems," *arXiv preprint arXiv:1812.01388*, 2018.

[39] D. Fourure, M. U. Javaid, N. Posocco, and S. Tihon, "Anomaly detection: how to artificially increase your f1-score with a biased evaluation protocol," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2021, pp. 3–18.

[40] "KDD Cup 1999 Data," accessed: 2017-08-30. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[41] "MIT Lincoln Laboratory: DARPA Intrusion Detection Evaluation," accessed: 2017-08-30. [Online]. Available: https://ll.mit.edu/ideval/data/1999data.html

[42] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2Nd Workshop on Machine Learning for Sensory Data Analysis*, ser. MLSDA'14. New York, NY, USA: ACM, 2014, pp. 4:4–4:11. [Online]. Available: http://doi.acm.org/10.1145/2689746.2689747

[43] S. T. Sarasamma, Q. A. Zhu, and J. Huff, "Hierarchical kohonenen net for anomaly detection in network security," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 2, pp. 302–312, 2005.

[44] H. A. Dau, V. Ciesielski, and A. Song, "Anomaly detection using replicator neural networks trained on examples of one class," in *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 2014, pp. 311–322.

[45] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal processing*, vol. 99, pp. 215–249, 2014.

[46] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[47] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," *arXiv preprint arXiv:2004.00433*, 2020.

[48] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[49] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: algorithms and applications," *The VLDB Journal*, vol. 8, no. 3, pp. 237–253, Feb 2000. [Online]. Available: https://doi.org/10.1007/s007780050006

[50] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *European conference on principles of data mining and knowledge discovery*. Springer, 2002, pp. 15–27.

[51] M. Amer and M. Goldstein, "Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer," in *Proc. of the 3rd RapidMiner Community Meeting and Conference (RCOMM 2012)*, 2012, pp. 1–12.

[52] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[53] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[54] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 97–104.

[55] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Information processing letters*, vol. 40, no. 4, pp. 175–179, 1991.

[56] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Loop: local outlier probabilities," in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, pp. 1649–1652.

[57] S. Hariri, M. Carrasco Kind, and R. J. Brunner, "Extended isolation forest," *IEEE Transactions on Knowledge and Data Engineering*, p. 1–1, 2019. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2019.2947676

[58] G. Staerman, P. Mozharovskyi, S. Clémençon, and F. d'Alché Buc, "Functional isolation forest," 2019.

[59] K. M. Ting, Y. Zhu, and Z.-H. Zhou, "Isolation kernel and its effect on svm," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2329–2337.

[60] J. Grim, P. Somol, M. Haindl, and J. Danes, "Computer-aided evaluation of screening mammograms based on local texture models," *IEEE Transactions on Image Processing*, vol. 18, no. 4, pp. 765–773, 2009.

[61] R. Laxhammar, G. Falkman, and E. Sviestins, "Anomaly detection in sea traffic-a comparison of the gaussian mixture model and the kernel density estimator," in *2009 12th International Conference on Information Fusion*. IEEE, 2009, pp. 756–763.

[62] L. Li, R. J. Hansman, R. Palacios, and R. Welsch, "Anomaly detection via a gaussian mixture model for flight operation and safety monitoring," *Transportation Research Part C: Emerging Technologies*, vol. 64, pp. 45–57, 2016.

[63] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International conference on learning representations*, 2018.

[64] D.-Y. Yeung and C. Chow, "Parzen-window network intrusion detectors," in *Object recognition supported by user interaction for service robots*, vol. 4. IEEE, 2002, pp. 385–388.

[65] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch, "Kernel PCA and de-noising in feature spaces," in *Advances in neural information processing systems*, 1999, pp. 536–542.

[66] J. Kim and C. D. Scott, "Robust kernel density estimation," *Journal of Machine Learning Research*, vol. 13, no. Sep, pp. 2529–2565, 2012.

[67] M. Goldstein and A. Dengel, "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm," *KI-2012: poster and demo track*, vol. 1, pp. 59–63, 2012.

[68] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[69] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 1100–1109. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045390.3045507

[70] E. Aleskerov, B. Freisleben, and B. Rao, "Cardwatch: a neural network based database mining system for credit card fraud detection," in *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, Mar 1997, pp. 220–226.

[71] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," Tech. Rep., 2015.

[72] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[73] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[74] J. Cannady, "Artificial neural networks for misuse detection," in *National Information Systems Security Conference*, 1998, pp. 368–81.

[75] J. Ryan, M.-J. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," in *Advances in Neural Information Processing Systems*, 1998, pp. 943–949.

[76] A. K. Ghosh, A. Schwartzbard, and M. Schatz, "Learning program behavior profiles for intrusion detection." in *Workshop on Intrusion Detection and Network Monitoring*, vol. 51462, 1999, pp. 1–13.

[77] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2. IEEE, 2002, pp. 1702–1707.

[78] W. Jiang, Y. Hong, B. Zhou, X. He, and C. Cheng, "A gan-based anomaly detection approach for imbalanced industrial time series," *IEEE Access*, vol. 7, pp. 143 608–143 619, 2019.

[79] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-anogan: Fast unsupervised anomaly detection with generative adversarial networks," *Medical Image Analysis*, vol. 54, pp. 30–44, 2019.

[80] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient gan-based anomaly detection," *CoRR*, vol. abs/1802.06222, 2018. [Online]. Available: http://arxiv.org/abs/1802.06222

[81] S. Goyal, A. Raghunathan, M. Jain, H. V. Simhadri, and P. Jain, "Drocc: Deep robust one-class classification," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3711–3721.

[82] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep anomaly detection with outlier exposure," *arXiv preprint arXiv:1812.04606*, 2018.

[83] L. Shoemaker and L. O. Hall, "Anomaly detection using ensembles," in *International Workshop on Multiple Classifier Systems*. Springer, 2011, pp. 6–15.

[84] A. Chiang and Y.-R. Yeh, "Anomaly detection ensembles: In defense of the average," in *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 3. IEEE, 2015, pp. 207–210.

[85] Z. Zhao, K. G. Mehrotra, and C. K. Mohan, "Ensemble algorithms for unsupervised anomaly detection," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2015, pp. 514–525.

[86] B. A. Tama, L. Nkenyereye, S. R. Islam, and K.-S. Kwak, "An enhanced anomaly detection in web traffic using a stack of classifier ensemble," *IEEE Access*, vol. 8, pp. 24 120–24 134, 2020.

[87] M. Grill and T. Pevný, "Learning combination of anomaly detectors for security domain," *Computer Networks*, vol. 107, pp. 55–63, 2016.

[88] J. Vanerio and P. Casas, "Ensemble-learning approaches for network security and anomaly detection," in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 2017, pp. 1–6.

[89] M. Flusser and P. Somol, "Adaptive approach for density-approximating neural network models for anomaly detection," in *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, Á. Herrero, C. Cambra, D. Urda, J. Sedano, H. Quintián, and E. Corchado, Eds. Cham: Springer International Publishing, 2021, pp. 415–425.

[90] M. Flusser, T. Pevný, and P. Somol, "Density-approximating neural network models for anomaly detection," *ACM SIGKDD workshop on outlier detection de-constructed*, 8 2018, london, United Kingdom.

[91] M. Flusser and P. Somol, "Efficient anomaly detection through surrogate neural networks," *Neural Computing and Applications*, vol. 34, no. 23, pp. 20 491–20 505, 2022. [Online]. Available: https://doi.org/10.1007/s00521-022-07506-9

[92] M. Zhao and V. Saligrama, "Anomaly detection with score functions based on nearest neighbor graphs," in *Advances in neural information processing systems*, 2009, pp. 2250–2258.

[93] C. R. Loader, "Local likelihood density estimation," *Ann. Statist.*, vol. 24, no. 4, pp. 1602–1618, 08 1996. [Online]. Available: https://doi.org/10.1214/aos/1032298287

[94] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[95] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[96] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[97] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548

[98] J. Kohout *et al.*, "Detection of malicious network connections," May 17 2016, cisco Technology, Inc., San Jose, CA (US), US Patent 9,344,441 B2. [Online]. Available: https://patents.google.com/patent/US9344441B2/

[99] D. Altman, D. Machin, T. Bryant, and M. Gardner, *Statistics with confidence: confidence intervals and statistical guidelines*. John Wiley & Sons, 2013.

[100] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, UK, 2014.

[101] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[102] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[103] L. Perini, V. Vercruyssen, and J. Davis, "Quantifying the confidence of anomaly detectors in their example-wise predictions," in *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. Springer Verlag*, 2020.

[104] M. Kusner, S. Tyree, K. Weinberger, and K. Agrawal, "Stochastic neighbor compression," in *International conference on machine learning*. PMLR, 2014, pp. 622–630.

[105] A. Zimek, M. Gaudet, R. J. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 428–436.

[106] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[107] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[108] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information fusion*, vol. 58, pp. 52–68, 2020.

[109] B. Pfülb and A. Gepperth, "A comprehensive, application-oriented study of catastrophic forgetting in dnns," *arXiv preprint arXiv:1905.08101*, 2019.

[110] S. Farquhar and Y. Gal, "Towards robust evaluations of continual learning," *arXiv preprint arXiv:1805.09733*, 2018.

[111] E. J. Spinosa, A. P. d. L. F. de Carvalho, and J. Gama, "Novelty detection with application to data streams," *Intelligent Data Analysis*, vol. 13, no. 3, pp. 405–422, 2009.

[112] M. Kloft and P. Laskov, "Online anomaly detection under adversarial impact," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 405–412.

[113] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[114] Y. Yao, A. Sharma, L. Golubchik, and R. Govindan, "Online anomaly detection for sensor systems: A simple and efficient approach," *Performance Evaluation*, vol. 67, no. 11, pp. 1059–1075, 2010.

[115] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[116] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 385–392.

[117] S. Han, Q. Wu, H. Zhang, B. Qin, J. Hu, X. Shi, L. Liu, and X. Yin, "Log-based anomaly detection with robust feature extraction and online learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2300–2311, 2021.

[118] J. Feng, C. Zhang, and P. Hao, "Online learning with self-organizing maps for anomaly detection in crowd scenes," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 3599–3602.

[119] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[120] D. van Leeuwen *et al.*, "A julia package for gaussian mixture models (gmms)," https://github.com/davidavdav/GaussianMixtures.jl, 2022.

[121] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *arXiv preprint arXiv:1901.01588*, 2019.

[122] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly, "The gan landscape: Losses, architectures, regularization, and normalization," 2018.

[123] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1801.00631*, 2018.

[124] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th international conference on World Wide Web*, 2016, pp. 263–273.

[125] M.-A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon, "Multiple instance learning: A survey of problem characteristics and applications," *Pattern Recognition*, vol. 77, pp. 329–353, 2018.

[126] Z.-H. Zhou, "Multi-instance learning: A survey," *Department of Computer Science & Technology, Nanjing University, Tech. Rep*, vol. 1, 2004.

[127] B. Babenko, "Multiple instance learning: algorithms and applications," *View Article PubMed/NCBI Google Scholar*, pp. 1–19, 2008.

[128] J. Wang and J.-D. Zucker, "Solving multiple-instance problem: A lazy learning approach," 2000.

[129] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International journal of computer vision*, vol. 73, no. 2, pp. 213–238, 2007.

[130] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," CALIFORNIA UNIV SAN DIEGO LA JOLLA DEPT OF COMPUTER SCIENCE AND ENGINEERING, Tech. Rep., 2002.

[131] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola, "Multi-instance kernels," in *ICML*, vol. 2, 2002, pp. 179–186.

[132] G. Edgar, *Measure, topology, and fractal geometry*. Springer Science & Business Media, 2007.

[133] V. Cheplygina, D. M. Tax, and M. Loog, "Multiple instance learning with bag dissimilarities," *Pattern Recognition*, vol. 48, no. 1, pp. 264–275, 2015.

[134] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample-problem," in *Advances in neural information processing systems*, 2007, pp. 513–520.

[135] L. Dong, "A comparison of multi-instance learning algorithms," Ph.D. dissertation, The University of Waikato, 2006.

[136] Z.-H. Zhou and M.-L. Zhang, "Neural networks for multi-instance learning," in *Proceedings of the International Conference on Intelligent Information Technology, Beijing, China*, 2002, pp. 455–459.

[137] X. Wang, Y. Yan, P. Tang, X. Bai, and W. Liu, "Revisiting multiple instance neural networks," *Pattern Recognition*, vol. 74, pp. 15–24, 2018.

[138] T. Pevný and M. Dědič, "Nested Multiple Instance Learning in Modelling of HTTP network traffic," *arXiv:2002.04059 [cs]*, Feb. 2020, arXiv: 2002.04059. [Online]. Available: http://arxiv.org/abs/2002.04059

[139] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, 2017, pp. 3391–3401.

[140] M. Ilse, J. Tomczak, and M. Welling, "Attention-based deep multiple instance learning," in *International conference on machine learning*. PMLR, 2018, pp. 2127–2136.

[141] M. Sun, T. X. Han, M.-C. Liu, and A. Khodayari-Rostamabad, "Multiple instance learning convolutional neural networks for object recognition," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 3270–3275.

[142] J. Wu, Y. Yu, C. Huang, and K. Yu, "Deep multiple instance learning for image classification and auto-annotation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3460–3469.

[143] K. Sirinukunwattana, S. E. A. Raza, Y.-W. Tsang, D. R. Snead, I. A. Cree, and N. M. Rajpoot, "Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1196–1206, 2016.

[144] L. Xiong, B. Póczos, and J. G. Schneider, "Group anomaly detection using flexible genre models," in *Advances in neural information processing systems*, 2011, pp. 1071–1079.

[145] L. Xiong, B. Póczos, J. Schneider, A. Connolly, and J. VanderPlas, "Hierarchical probabilistic models for group anomaly detection," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 789–797.

[146] K. Muandet and B. Schölkopf, "One-class support measure machines for group anomaly detection," *arXiv preprint arXiv:1303.0309*, 2013.

[147] G. Quellec, M. Lamard, M. Cozic, G. Coatrieux, and G. Cazuguel, "Multiple-instance learning for anomaly detection in digital mammography," *Ieee transactions on medical imaging*, vol. 35, no. 7, pp. 1604–1614, 2016.

[148] W. Yang, Y. Gao, and L. Cao, "Trasmil: A local anomaly detection framework based on trajectory segmentation and multi-instance learning," *Computer Vision and Image Understanding*, vol. 117, no. 10, pp. 1273–1286, 2013.

[149] W. Sultani, C. Chen, and M. Shah, "Real-world anomaly detection in surveillance videos," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6479–6488.

[150] A. M. Kamoona, A. K. Gostar, A. Bab-Hadiashar, and R. Hoseinnezhad, "Multiple instance-based video anomaly detection using deep temporal encoding-decoding," *Expert Systems with Applications*, p. 119079, 2022.

[151] J.-C. Feng, F.-T. Hong, and W.-S. Zheng, "Mist: Multiple instance self-training framework for video anomaly detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 14 009–14 018.

[152] M. Dědič, "MilDatasets.jl." [Online]. Available: https://github.com/marekdedic/MilDatasets.jl

[153] T. Pevný and P. Somol, "Using neural network formalism to solve multiple-instance problems," in *International Symposium on Neural Networks*. Springer, 2017, pp. 135–142.

[154] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[155] T. Pevný and Šimon Mandlík, "Mill.jl framework: a flexible library for (hierarchical) multi-instance learning," https://github.com/pevnak/Mill.jl, 2018.

[156] F. Briggs, X. Fern, and R. Raich, "Rank-loss support instance machines for MIML instance annotation."

[157] Y. Chen, J. Bi, and J. Z. Wang, "MILES: Multiple-instance learning via embedded instance selection," vol. 28, no. 12, pp. 1931–1947.

[158] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," vol. 89, no. 1, pp. 31–71. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370296000343

[159] A. Srinivasan, S. Muggleton, and R. King, "Comparing the use of background knowledge by inductive logic programming systems."

[160] S. Ray and M. Craven, "Learning statistical models for annotating proteins with function information using biomedical text," vol. 6, no. 1, p. S18. [Online]. Available: https://doi.org/10.1186/1471-2105-6-S1-S18

[161] ——, "Supervised versus multiple instance learning: An empirical comparison," pp. 697–704.

[162] J. Brabec, T. Komárek, V. Franc, and L. Machlica, "On model evaluation under non-constant class imbalance," in *International Conference on Computational Science*. Springer, 2020, pp. 74–87.

[163] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://www.jstor.org/stable/3001968

Appendixes

Appendix A

Additional figures

A.1 Supporting material to Sect. 4.8

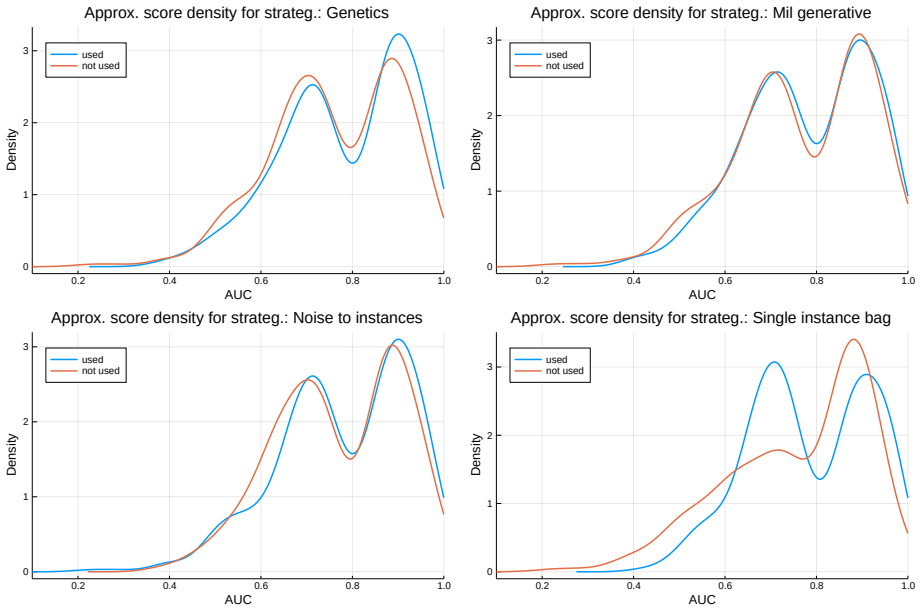


Figure A.1: Parallel figure to Fig. 4.11, computed on multiple validated experimental data. Overall strategy performance comparison across all data sets. For each strategy, we compare distribution of performance of the experimental shots where the strategy was utilized with these where was not.

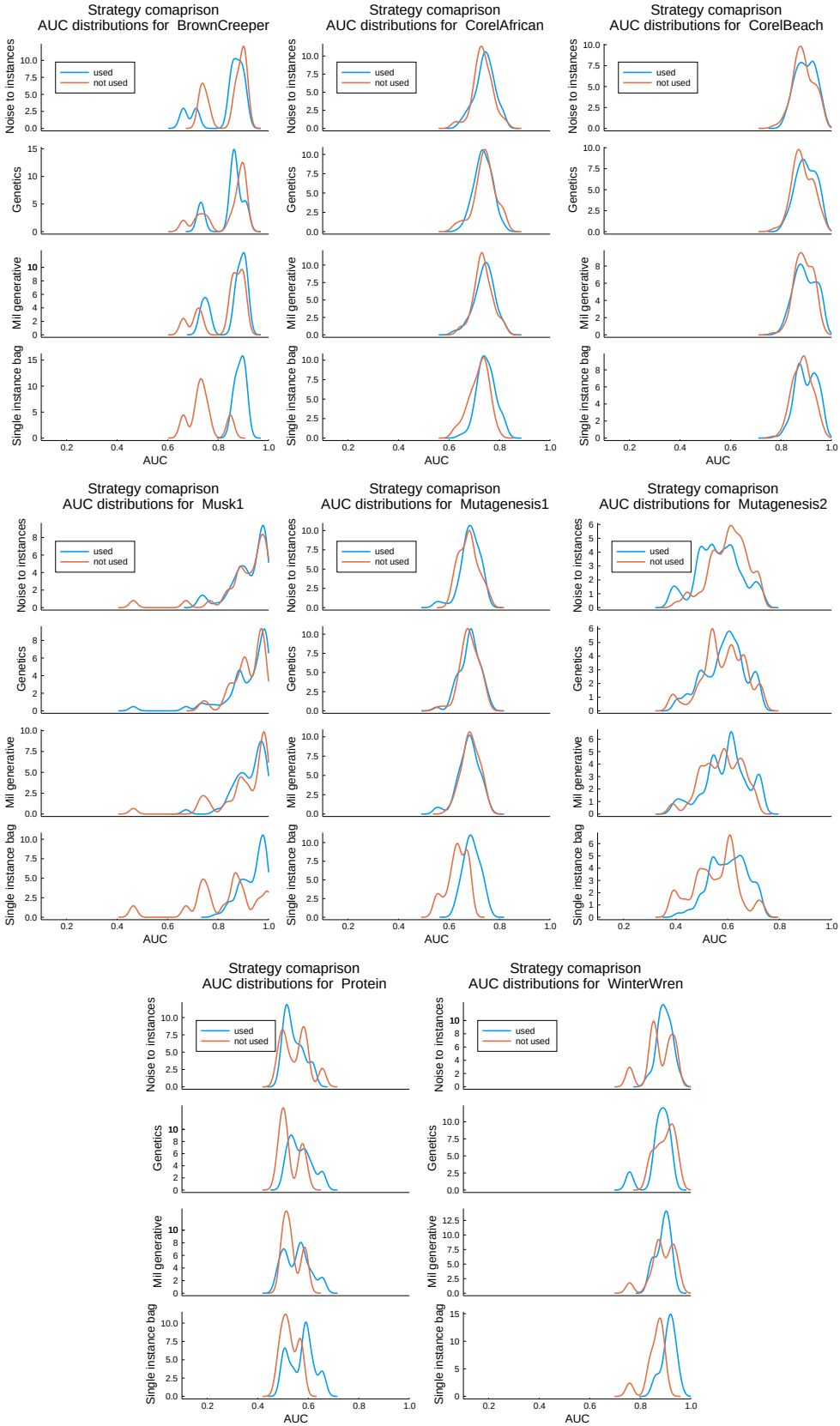


Figure A.2: Parallel figure to Fig. 4.12, computed on multiple validated experimental data. Overall strategy performance analyzed separately for each data set.

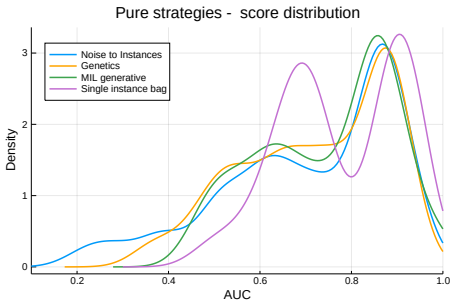


Figure A.3: Parallel figure to Fig. 4.13, computed on multiple validated experimental data. Overall pure strategy performance across all data sets

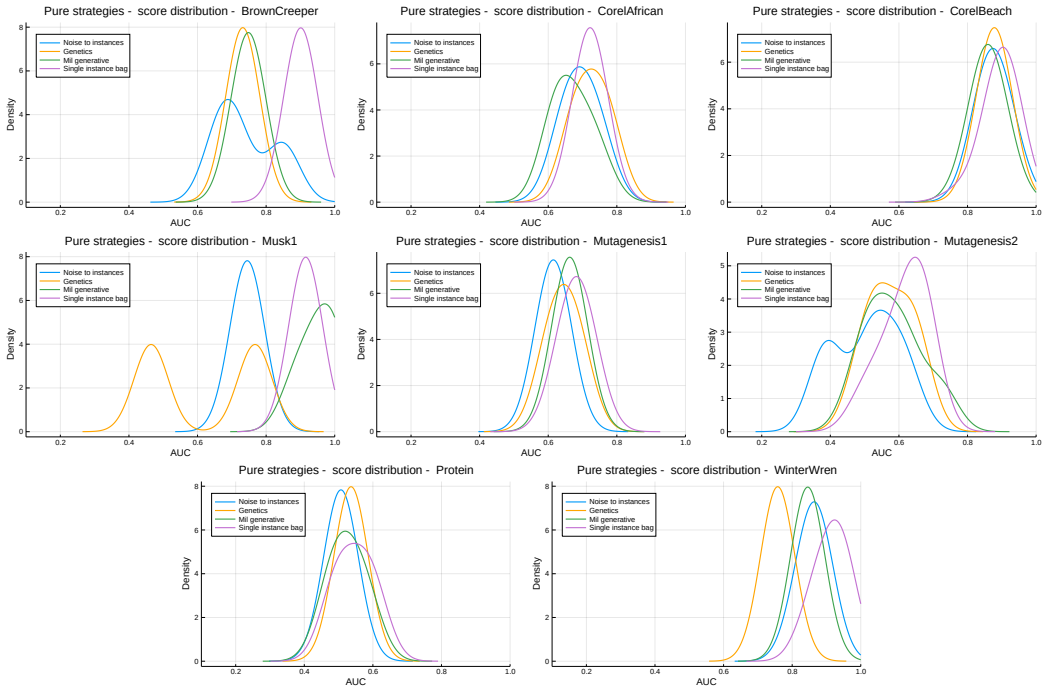


Figure A.4: Parallel figure to Fig. 4.14, computed on multiple validated experimental data. Overall pure strategy performance analyzed separately for each data set.