MemStream: Memory-Based Streaming Anomaly Detection

Siddharth Bhatia
National University of Singapore
Singapore
siddharth@comp.nus.edu.sg

Arjit Jain
IIT Bombay
India
arjit@cse.iitb.ac.in

Shivin Srivastava
National University of Singapore
Singapore
shivin@comp.nus.edu.sg

Kenji Kawaguchi
Harvard University
United States
kkawaguchi@fas.harvard.edu

Bryan Hooi
National University of Singapore
Singapore
bhooi@comp.nus.edu.sg

ABSTRACT

Given a stream of entries over time in a multi-dimensional data setting where concept drift is present, how can we detect anomalous activities? Most of the existing unsupervised anomaly detection approaches seek to detect anomalous events in an offline fashion and require a large amount of data for training. This is not practical in real-life scenarios where we receive the data in a streaming manner and do not know the size of the stream beforehand. Thus, we need a data-efficient method that can detect and adapt to changing data trends, or *concept drift*, in an online manner. In this work, we propose **MemStream**, a streaming anomaly detection framework, allowing us to detect unusual events as they occur while being resilient to concept drift. We leverage the power of a denoising autoencoder to learn representations and a memory module to learn the dynamically changing trend in data without the need for labels. We prove the optimum memory size required for effective drift handling. Furthermore, MemStream makes use of two architecture design choices to be robust to memory poisoning. Experimental results show the effectiveness of our approach compared to state-of-the-art streaming baselines using 2 synthetic datasets and 11 real-world datasets.

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**; **Online learning settings**.

KEYWORDS

Anomaly Detection, Streams, Concept Drift

ACM Reference Format:
Siddharth Bhatia, Arjit Jain, Shivin Srivastava, Kenji Kawaguchi, and Bryan Hooi. 2022. MemStream: Memory-Based Streaming Anomaly Detection. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3485447.3512221

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9096-5/22/04.
https://doi.org/10.1145/3485447.3512221

1 INTRODUCTION

Anomaly detection is a fundamental and well-studied problem in many areas, such as cybersecurity [11, 69], video surveillance [42, 54], financial fraud [67] and healthcare [61]. Traditional classifiers trained in a supervised learning setting do not work well in anomaly detection because of the cold-start problem, i.e., the amount of anomalous data is usually not sufficient to train the model. Therefore, anomaly detectors are trained in an unsupervised setting where the normal data distribution is learned and instances that appear unlikely under this distribution are identified as anomalous.

Developing effective methods for handling *multi-aspect data* (i.e. data having multiple features or dimensions) still remains a challenge. This is especially true in an unsupervised setting, where traditional anomaly detection algorithms, such as One-Class SVM, tend to perform poorly because of the curse of dimensionality. Deep architectures such as Autoencoders [31], because of their ability to learn multiple levels of representation, are able to achieve better performance compared to their shallow counterparts [10]. For anomaly detection, existing deep learning based techniques include deep belief networks [23], variational autoencoders [3, 76], adversarial autoencoders [8, 37, 80], and deep one-class networks [16, 57].

The problem of anomaly detection becomes even more challenging when the data arrives in a streaming/online manner and we want to detect anomalies in real-time. For example, intrusions in cybersecurity need to be detected as soon as they arrive to minimize the harm caused. Moreover, in streaming data, there can be a drift in the distribution over time which the existing approaches [12, 27, 30, 43, 45, 47] are unable to fully handle.

To handle concept drift in a streaming setting, our approach uses an explicit memory module. For anomaly detection, this memory can be used to store the trends of normal data that act as a baseline with which to judge incoming records. A read-only memory, in a drifting setting, is of limited use and thus should be accompanied by an appropriate memory update strategy. The records arrive over time; thus, older records in the memory might no longer be relevant to the current trends suggesting a First-In-First-Out memory replacement strategy. The introduction of memory, with an appropriate update strategy, seems to tackle some of the issues in streaming anomaly detection with concept drift. However, the system described so far does not provide a fail-safe for when an anomalous sample enters the memory and is thus susceptible to memory poisoning.

We, therefore, propose MemStream, which uses a denoising autoencoder [73] to extract features, and a memory module to learn the dynamically changing trend, thereby avoiding the over-generalization of autoencoders (i.e. the problem of autoencoders reconstructing anomalous samples well). Our streaming framework is resilient to concept drift and we prove a theoretical bound on the size of memory for effective drift handling. Moreover, we allow quick retraining when the arriving stream becomes sufficiently different from the training data.

We also discuss two architectural design choices to make MemStream robust to memory poisoning. The first modification prevents anomalous elements from entering the memory, and the second modification deals with how the memory can be self-corrected and recovered even if it harbors anomalous elements. Finally, we discuss the effectiveness of MemStream compared to state-of-the-art streaming baselines.

In summary, the main contributions of our paper are:

1. **Streaming Anomaly Detection:** We propose a novel streaming approach using a denoising autoencoder and a memory module, for detecting anomalies. MemStream is resilient to concept drift and allows quick retraining.
2. **Theoretical Guarantees:** In Proposition 1, we discuss the optimum memory size for effective concept drift handling. In Proposition 2, we discuss the motivation behind our architecture design.
3. **Robustness to Memory Poisoning:** MemStream prevents anomalies from entering the memory and can self-correct and recover from bad memory states.
4. **Effectiveness:** Our experimental results show that MemStream convincingly outperforms 11 state-of-the-art baselines using 2 synthetic datasets (that we release as open-source) and 11 popular real-world datasets.

Reproducibility: Our code and datasets are available on https://github.com/Stream-AD/MemStream.

2 RELATED WORK

[17] surveys traditional anomaly detection methods including reconstruction-based approaches [14, 28, 32, 33, 39, 40, 81], clustering-based [1, 6, 34, 75, 83], one class classification-based [62, 63, 74]. Several deep learning based methods have also been proposed for anomaly detection such as GAN-based approaches [2, 7, 20, 48, 61, 77, 79], Energy-based [36, 80], Autoencoder-based [3, 24, 66, 68, 76, 82, 84], and RNN-based [60]; see [15, 49] for extensive surveys. However, deep learning based approaches such as MemAE [24] do not process the data in a streaming manner and typically require a large amount of training data in an offline setting, whereas we process the data in an online manner. Additionally, we provide theoretical analysis and are robust to memory poisoning. Anomaly detection is a vast topic by itself and cannot be fully covered in this manuscript; in this section, our review mainly focuses on methods that can detect anomalies in streams containing concept drift; see [29, 41] for concept drift literature and [5, 9, 18, 50, 64] for different ways to detect concept drift in streams.

As for density-based approaches, Local Outlier Factor (LOF) [13] estimates the local density at each point, then identifies anomalies as points with much lower local density than their neighbors. DILOF [47] improves upon LOF and LOF variants [53, 58] by adopting a novel density-based sampling scheme to summarize the data, without prior assumptions on the data distribution. LUNAR [26] is a hybrid approach combining deep learning and LOF. However, LOF-based approaches are suitable only for lower-dimensional data due to the curse of dimensionality.

Isolation Forest (IF) [38] constructs trees by randomly selecting features and splitting them at random split points, and then defines anomalies as points that are separated from the rest of the data at low depth values. HS-Tree [70] uses an ensemble of randomly constructed half-space trees with a sliding window to detect anomalies in evolving streaming data. iForestASD [21] uses a sliding window frame scheme to handle abnormal data. Random Cut Forest (RCF) [27] tries to further improve upon IF by creating multiple random cuts (trees) of data and constructing a forest of such trees to determine whether a point is anomalous or not. Recently, [30] shows that splitting by only one variable at a time introduces some biases in IF which can be overcome by using hyperplane cuts instead. They propose Extended Isolation Forest (Ex. IF) [30] where the split criterion is based on a threshold set on a linear combination of randomly chosen variables instead of a threshold on a single variables value at a time. However, these approaches compute an anomaly score by traversing a tree structure that is bounded by the maximum depth parameter and the size of the sliding window, therefore they do not capture long-range dependence.

Popular streaming approaches include STORM [4], which uses a sliding window to detect global distance-based outliers in data streams with respect to the current window. RS-Hash [59] uses subspace grids and randomized hashing in an ensemble to detect anomalies. For each model in the ensemble, a grid is constructed using subsets of features and data, random hashing is used to record data counts in grid cells, and the anomaly score of a data point is the log of the frequency in its hashed bins. LODA [52] generates several weak anomaly detectors by producing many random projections of the data and then computing a density estimation histogram for each projection. The outlier scores produced are the mean negative log-likelihood according to each histogram for each point. xStream [43] detects anomalies in feature-evolving data streams through the use of a streaming random projection scheme and ensemble of half-space chains. MStream [12] performs feature extraction and then detects group anomalies in multi-aspect streams. Kitsune [45] is an ensemble of light-weight autoencoders for real-time anomaly detection. We compare with all these methods in Section 5.

3 PROBLEM

Let $\mathcal{X} = \{x_1, x_2, \cdots\}$ be records arriving in a streaming manner. Each entry $x_i = (x_{i1}, \cdots, x_{id})$ consisting of *d attributes* or dimensions, where each dimension can either be categorical (e.g. IP address) or real-valued (e.g. average packet length).

Our goal is to detect anomalies in streaming data. A common phenomenon in real-world data is that the nature of the stream changes over time. These changes are generally described in terms of the statistical properties of the stream, such as the mean changes across some or all features. As the definition of the "concept" of normal behavior changes, so does the definition of an anomaly. Thus, we need a model that is able to adapt to the dynamic trend and thereby recognize anomalous records.

4 ALGORITHM

4.1 Motivation

Table 1: Simple toy example, consisting of a stream of records over time with a trend shift at $t = 6$.

Time	Feature 1	Feature 2	Feature 3	...
1	8.39	1.44	4.16	⋯
2	6.72	4.55	3.49	⋯
3	3.49	2.10	1.56	⋯
4	4.28	0.64	1.22	⋯
5	5.54	2.40	6.55	⋯
6	183.75	132.03	9.86	⋯
7	146.47	128.49	16.52	⋯
8	197.96	97.16	15.05	⋯
9	192.50	89.95	12.46	⋯
10	158.32	10.37	15.76	⋯

Consider an attacker who hacks a particular IP address and uses it to launch denial of service attacks on a server. Modern cybersecurity systems are trained to detect and block such attacks, but this is made more challenging by changes over time, e.g. in the identification of attacking machines. This is a "concept" drift and the security system must learn to identify such changing trends to mitigate the attacks. Consider the toy example in Table 1, comprising of a multi-dimensional temporal data stream. There is a sudden distribution change and concept drift in all attributes from time $t = 5$ to $t = 6$.

The main challenge for the algorithm is to detect these types of patterns in a **streaming** manner within a suitable timeframe. That is, the algorithm should not give an impulsive reaction to a short-lived change in the base distribution, but also should not take too long to adapt to the dynamic trend. Note that we do not want to set any limits a priori on the duration of the anomalous activity we want to detect, or the window size after which the model should be updated to account for the concept drift.

4.2 Overview

As shown in Figure 1, the proposed MemStream algorithm addresses these problems through the use of a memory augmented feature extractor that is initially trained on a small subset of normal data. The memory acts as a reserve of encodings of normal data. At a high level, the role of the feature extractor is to capture the structure of normal data. An incoming record is then scored by *calculating the discounted score* based on the similarity of its encoding as evaluated against those in memory. Based on this score, if the record is deemed normal, then it is used to *update the memory.* To adapt to the changing data trend, memory is required to keep track of the data drift from the original distribution. Since concept drift is generally a gradual process, the memory should maintain the temporal contiguity of records. This is achieved by following a First-In-First-Out (FIFO) memory replacement policy.

4.3 Feature Extraction

Neural Networks can learn representations using an autoencoder consisting of two parts - an encoder and a decoder [25]. The encoder forms an intermediate representation of the input samples and the decoder is trained to reconstruct the input samples from their intermediate representations. Denoising autoencoders [73] partially corrupt the input data before passing it through the encoder. Intuitively, this "forces" the network to capture the useful structure in the input distribution, pushing it to learn more robust features of the input. In our implementation, we use an additive isotropic Gaussian noise model.

MemStream allows flexibility in the choice of the feature extraction backbone. We consider Principal Component Analysis (PCA) and Information Bottleneck (IB) [35, 72] as alternatives to autoencoders for feature extraction [12]. PCA-based methods are effective for off-the-shelf learning, with little to no hyperparameter tuning. Information Bottleneck can be used for learning useful features by posing the following optimization problem:

$$\min_{p(t|x)} I(X;T) - \beta I(T;Y)$$

where X, Y, and T are random variables. T is the compressed representation of X, $I(X;T)$ and $I(T;Y)$ are the mutual information of X and T, and of T and Y, respectively, and β is a Lagrange multiplier. The problem configuration and the available data greatly influence the choice of the feature extraction algorithm. We evaluate the methods to extract features in Section 5.5.

4.4 Memory

Memory-based Representation: The memory $\boldsymbol{M}$ is a collection of N real-valued D dimensional vectors where D is the dimension of the encodings $\mathbf{z}$. Given a representation $\mathbf{z}$, the memory is queried to retrieve the K-nearest neighbours $\{\hat{\mathbf{z}}_1^t, \hat{\mathbf{z}}_2^t...\hat{\mathbf{z}}_K^t\}$ of $\mathbf{z}$ in $\boldsymbol{M}$ under the ℓ_1 norm such that:

$$||\hat{\mathbf{z}}_1^t - \mathbf{z}||_1 \leq ... \leq ||\hat{\mathbf{z}}_K^t - \mathbf{z}||_1$$

The hyper-parameter N denotes the memory size. Performance of the algorithm varies depending on the value of N; very large or small values of N would hinder the performance.

Memory Update: Fixed memory trained on limited samples of streaming data will not be able to handle concept drift; therefore, continuous memory update is necessary. Different memory update strategies can be used such as Least Recently Used (LRU), Random Replacement (RR), and First-In-First-Out (FIFO). We observe that the FIFO memory update policy wherein the new element to be added replaces the earliest added element in the memory works well in practice. It can easily handle concept drift in streaming data as the memory retains the most recent non-anomalous samples from the distribution. We compare FIFO with LRU and RR strategies in more detail in Section 5.5. It is also interesting to note that MemStream can easily handle periodic patterns by adjusting the memory size: a memory of size greater than the product of the period and the sampling frequency should be sufficient to avoid flagging periodic

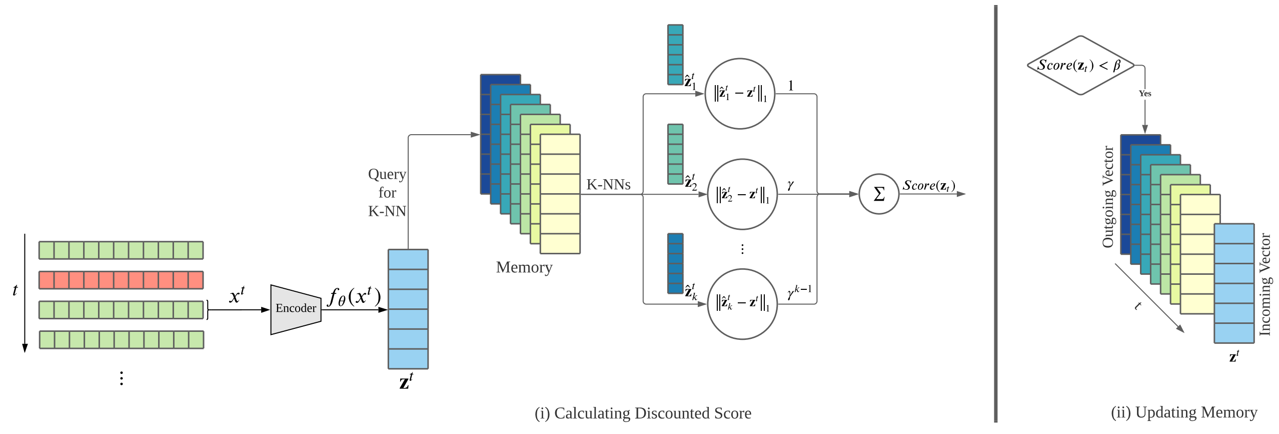

Figure 1: After an initial training of the feature extractor on a small subset of normal data, MemStream processes records in two steps: (i) It outputs anomaly scores for each record by querying the memory for K-nearest neighbours to the record encoding and calculating a discounted distance and (ii) It updates the memory, in a FIFO manner, if the anomaly score is within an update threshold β.

changes as anomalies. Section 5.2 evaluates MemStream's ability to detect anomalies in a periodic setting.

As shown in Algorithm 1, the autoencoder is initially trained with a small amount of data $\mathcal{D}$ to learn how to generate data embeddings (line 2). The memory is initialized with the same training dataset (line 3). We also store the mean and standard deviation of this small training dataset. As new records arrive, the encoder performs normalization using the stored mean and standard deviation and computes the compressed representation $\mathbf{z}^t$ (line 6). It then computes the K-nearest neighbours $(\hat{\mathbf{z}}_1^t, \cdots, \hat{\mathbf{z}}_K^t)$ by querying the memory (line 8), and calculates their ℓ_1 distance with $\mathbf{z}^t$ (line 10). The final discounted score is calculated as an exponentially weighted average (weighting factor γ) (line 12). This helps in making the autoencoder more robust. The discounted score is then compared against a user-defined threshold β (line 14) and the new record is updated into the memory in a FIFO manner if the score falls within β (line 15). This step ensures that anomalous records do not enter the memory. If the memory is updated, then the stored mean and standard deviation are also updated accordingly. The discounted score is returned as the anomaly score for the record $\mathbf{x}^t$ (line 17).

Algorithm 1: MemStream

```
Input: Stream of data records
Output: Anomaly scores for each record
1  ▷ Initialization
2  Feature Extractor, $f_\theta$, trained using small subset of data $\mathcal{D}$
3  Memory, $M$, initialized as $f_\theta(\mathcal{D})$
4  while new sample $\mathbf{x}^t$ is received: do
5      ▷ Extract features:
6      $\mathbf{z}^t = f_\theta(\mathbf{x}^t)$
7      ▷ Query memory:
8      $\{\hat{\mathbf{z}}_1^t, \hat{\mathbf{z}}_2^t ... \hat{\mathbf{z}}_K^t\}$ = $K$-nearest neighbours of $\mathbf{z}^t$ in $M$
9      ▷ Calculate distance:
10     $R(\mathbf{z}^t, \hat{\mathbf{z}}_i^t) = ||\mathbf{z}^t - \hat{\mathbf{z}}_i^t||_1$ for all $i \in 1..K$
11     ▷ Assign discounted score:
12     $Score(\mathbf{z}^t) = \frac{\sum_{i=1}^{K} \gamma^{i-1} R(\mathbf{z}^t, \hat{\mathbf{z}}_i^t)}{\sum_{i=1}^{K} \gamma^{i-1}}$
13     ▷ Update Memory:
14     if $Score(\mathbf{z}^t) < \beta$ then
15         Replace earliest added element in $\boldsymbol{M}$ with $\mathbf{z}^t$
16     ▷ Anomaly Score:
17     output $Score(\mathbf{z}^t)$
```

4.5 Theoretical Analysis

4.5.1 Relation between Memory Size and Concept Drift. Our analysis on the relation of the memory size and concept drifts suggests that the memory size should be proportional to (the spread of data distributions) / (the speed of concept drifts).

As we increase the size of memory, we can decrease the possibility of a false positive (falsely classifying a normal sample as an anomaly). This is because it is more likely for a new data point to have a close point in a larger memory. Therefore, on the one hand, in order to decrease the false *positive* rate, we want to increase the memory size. On the other hand, in order to minimize a false *negative* rate (i.e., failing to raise an alarm when an anomaly did happen), Proposition 1 suggests that the memory size should be smaller than some quantity proportional to (standard deviations of distributions) / (the speed of distributional drifts). That is, it suggests that the memory size should be smaller than $2\sigma\sqrt{d(1+\epsilon)}/\alpha$, where d is the input dimension, α measures the speed of distributional drifts, σ is the standard deviation of distributions, and $\epsilon \in (0, 1)$. More concretely, under drifting normal distributions, the proposition shows that a new distribution after τ drifts and an original distribution before the τ drifts are sufficiently dissimilar whenever $\tau > 2\sigma\sqrt{d(1+\epsilon)}/\alpha$, so that the memory should forget about the original distribution to minimize a false-negative rate. We also discuss this effect of increasing the memory size in Section 5.5.

Proposition 1. *(Proof in Appendix 1) Define $S_{t,\epsilon} = \{x \in \mathbb{R}^d : \|x - \mu_t\|_2 \leq \sigma\sqrt{d(1+\epsilon)}\}$. Let $(\mu_t)_t$ be the sequence such that there exits a positive real number α for which $\|\mu_t - \mu_{t'}\|_2 \geq (t'-t)\alpha$ for any $t < t'$. Let $\tau > \frac{2\sigma\sqrt{d(1+\epsilon)}}{\alpha}$ and $x_t \sim \mathcal{N}(\mu_t, \sigma I)$ for all $t \in \mathbb{N}^+$. Then, for any $\epsilon > 0$ and $t \in \mathbb{N}^+$, with probability at least $1 - 2\exp(-d\epsilon^2/8)$, the following holds: $x_t \in S_{t,\epsilon}$ and $x_{t+\tau} \notin S_{t,\epsilon}$.*

4.5.2 Architecture Choice. In the following, we provide one reason why we use an architecture with $d \leq D$, where d is the input dimension and D is the embedding dimension. Namely, Proposition (2) shows that if $d > D$, then there exists an anomaly constructed through perturbation of a normal sample such that the anomaly is not detectable. The construction of an anomaly in the proof is indeed unique to the case of $d > D$, and is not applicable to the case of $d \leq D$. This provides the motivation of why we may want to use the architecture of $d \leq D$, to avoid such an undetectable anomaly.

Let θ be fixed. Let f_θ be a deep neural network $f_\theta : \mathbb{R}^d \to \mathbb{R}^D$ with ReLU and/or max-pooling as: $f_\theta(x) = \sigma^{[L]}(z^{[L]}(x,\theta)), z^{[l]}(x,\theta) = W^{[l]}\sigma^{(l-1)}\left(z^{[l-1]}(x,\theta)\right)$, for $l = 1, 2, \dots, L$, where $\sigma^{(0)}\left(z^{[0]}(x,\theta)\right) = x$, σ represents nonlinear function due to ReLU and/or max-pooling, and $W^{[l]} \in \mathbb{R}^{N_l \times N_{l-1}}$ is a matrix of weight parameters connecting the $(l-1)$-th layer to the l-th layer. For the nonlinear function σ due to ReLU and/or max-pooling, we can define $\dot{\sigma}^{[l]}(x,\theta)$ such that $\dot{\sigma}^{[l]}(x,\theta)$ is a diagonal matrix with each element being 0 or 1, and $\sigma^{[l]}\left(z^{[l]}(x,\theta)\right) = \dot{\sigma}^{[l]}(x,\theta)z^{[l]}(x,\theta)$. For any differentiable point x of f_θ, define $\Omega(x) = \{x' \in \mathbb{R}^d : \forall l,\ \dot{\sigma}^{[l]}(x',\theta) = \dot{\sigma}^{[l]}(x,\theta)\}$ and $\mathcal{B}_r(x) = \{x' \in \mathbb{R}^d : \|x - x'\|_2 \leq r\}$.

Proposition 2. *(Proof in Appendix 2) Let x be a differentiable point of f_θ such that $\mathcal{B}_r(x) \subseteq \Omega(x)$ for some $r > 0$. If $d > D$, then there exists a $\delta \in \mathbb{R}^d$ such that for any $\hat{x} \in \mathbb{R}^d$ and $\bar{\beta} > 0$, the following holds: $\|\delta\|_2 = r$ and*

$$R(x,\hat{x}) < \bar{\beta} \implies R(x+\delta,\hat{x}) < \bar{\beta}.$$

5 EXPERIMENTS

In this section, we aim to answer the following questions:

(1) **Comparison to Streaming Methods:** How accurately does MemStream detect real-world anomalies as compared to state-of-the-art streaming baseline methods?
(2) **Concept Drift:** How fast can MemStream adapt under concept drift?
(3) **Retraining:** What effect does retraining MemStream have on the accuracy and time?
(4) **Self-Correction and Recovery:** Does MemStream provide a self-correction mechanism to recover from "bad" memory states?

Datasets: KDDCUP99 [19] is a popular multi-aspect anomaly detection dataset. *NSL-KDD* [71] solves some of the inherent problems of *KDDCUP99* such as redundant and duplicate records. Recently, [56] recommends to use *UNSW-NB15* [46] and *CICIDS*-DoS [65] after surveying more than 30 datasets. In addition, we use seven standard ODDS [55] datasets: Ionosphere, Cardio, Satellite, Satimage-2, Mammograph, Pima, and ForestCover. Datasets are discussed in detail in Appendix B.

Apart from these standard datasets, we also create and use a synthetic dataset (that we plan to release publicly), *Syn* with 10% anomalies and $T = 10000$ samples. This dataset is constructed as a superposition of a linear wave with slope 2×10^{-3}, two sinusoidal waves with time periods $0.2T$ and $0.3T$ and amplitudes 8 and 4, altogether with an additive Gaussian noise from a standard normal distribution. 10% of the samples are chosen at random and are perturbed with uniform random noise from the interval $[3, 6]$ to simulate anomalous data. Figure 2 shows a scatterplot of the synthetic data. Anomalous samples constitute 10% of the data and are represented by red dots in the scatter plot.

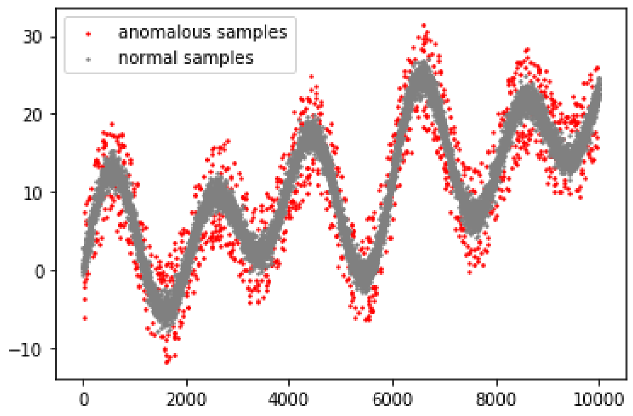

Figure 2: Scatterplot of the Synthetic Dataset.

By construction, the synthetic data distribution changes significantly over time. The presence of this concept drift makes the task challenging resulting in poor performance by baseline approaches, as seen in the Experiments. However, MemStream, through the use of explicit memory, can adapt to the drift in the distribution, proving its effectiveness in concept drift settings.

Experimental Setup. All methods output an anomaly score for every record (higher is more anomalous). We report the ROC-AUC (Area under the Receiver Operating Characteristic curve). All experiments, unless explicitly specified, are performed 5 times for each parameter group, and the mean values are reported. All experiments are carried out on a 2.6*GHz* Intel Core *i*7 system with 16*GB* RAM and running Mac OS Catalina 10.15.5. Following MStream, we take the output dimension as 8 for PCA and IB. For MemStream-PCA, we use the open-source implementation available in the scikit-learn [51] library of Principal Component Analysis. For MemStream-IB, we used an online implementation [1] for the underlying Information Bottleneck algorithm with $\beta = 0.5$ and the variance parameter set to 1. The network was implemented as a 2 layer binary classifier. For

[1]https://github.com/burklight/nonlinear-IB-PyTorch

Table 2: AUC of MemStream and Streaming Baselines. Averaged over 5 runs.

Method	KDD99	NSL	UNSW	DoS	Syn.	Ion.	Cardio	Sat.	Sat.-2	Mamm.	Pima	Cover
STORM (CIKM'07)	0.914	0.504	0.810	0.511	0.910	0.637	0.507	0.662	0.514	0.650	0.528	0.778
HS-Tree (IJCAI'11)	0.912	0.845	0.769	0.707	0.800	0.764	0.673	0.519	0.929	0.832	0.667	0.731
iForestASD (ICONS'13)	0.575	0.500	0.557	0.529	0.501	0.694	0.515	0.504	0.554	0.574	0.525	0.603
RS-Hash (ICDM'16)	0.859	0.701	0.778	0.527	0.921	0.772	0.532	0.675	0.685	0.773	0.562	0.640
RCF (ICML'16)	0.791	0.745	0.512	0.514	0.774	0.675	0.617	0.552	0.738	0.755	0.571	0.586
LODA (ML'16)	0.500	0.500	− − −	0.500	0.506	0.503	0.501	0.500	0.500	0.500	0.502	0.500
Kitsune (NDSS'18)	0.525	0.659	0.794	0.907	− − −	0.514	0.966	0.665	0.973	0.592	0.511	0.888
DILOF (KDD'18)	0.535	0.821	0.737	0.613	0.703	**0.928**	0.570	0.561	0.563	0.733	0.543	0.688
xStream (KDD'18)	0.957	0.552	0.804	0.800	0.539	0.847	0.918	0.677	**0.996**	0.856	0.663	0.894
MStream (WWW'21)	0.844	0.544	0.860	0.930	0.505	0.670	**0.986**	0.563	0.958	0.567	0.529	0.874
Ex. IF (TKDE'21)	0.874	0.767	0.541	0.734	− − −	0.872	0.921	0.716	0.995	0.867	0.672	0.902
MemStream	**0.980**	**0.978**	**0.972**	**0.938**	**0.955**	0.821	0.884	**0.727**	0.991	**0.894**	**0.742**	**0.952**

MemStream, the encoder and decoder were implemented as single layer Neural Nets with ReLU activation. We used Adam Optimizer to train both these networks with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Grid Search was used for hyperparameter tuning: Learning Rate was set to $1e-2$, and the number of epochs was set to 5000. The memory size N, and the value of the threshold β, can be found in Table 8 in the Appendix. Memory size for each intrusion detection dataset was searched in $\{256, 512, 1024, 2048\}$. For multi-dimensional point datasets, if the size of the dataset was less than 2000, N was searched in $\{4, 8, 16, 32, 64\}$, and if it was greater than 2000, then N was searched in $\{128, 256, 512, 1024, 2048\}$. The threshold β, is an important parameter in our algorithm, and hence we adopt a finer search strategy. For each dataset, and method, β was searched in $\{10, 1, 0.1, 0.001, 0.0001\}$. Unless stated otherwise, AE was used for feature extraction with output dimension $D = 2d$, and with a FIFO memory update policy. The KNN coefficient γ was set to 0 for all experiments. For the synthetic dataset, we use a memory size of $N = 16$. For all methods, across all datasets, the number of training samples used is equal to the memory size.

5.1 Comparison to Streaming Methods

Table 2 shows the AUC of MemStream and state-of-the-art streaming baselines. We use open-sourced implementations of DILOF [47], xStream [43], MStream [12], Extended Isolation Forest (Ex. IF) [30], provided by the authors, following parameter settings as suggested in the original papers. For STORM [4], HS-Tree [70], iForestASD [21], RS-Hash [59], Random Cut Forest (RCF) [27], LODA [52], Kitsune [45], we use the open-source library PySAD [78] implementation, following original parameters. Baseline parameters are listed in Appendix D. LODA could not process the large *UNSW* dataset. Ex. IF and Kitsune are unable to run on datasets with just one field, therefore their results with *Syn* are not reported.

Random subspace generation in RS-Hash includes many irrelevant features into subspaces while omitting relevant features in high-dimensional data. The objective of random projection in LODA retains the pairwise distances of the original space, therefore it fails to provide accurate outlier estimation. xStream performs well in *KDD99*, MStream performs well in *DoS*, however, note that MemStream achieves statistically significant improvements in AUC scores over baseline methods. Moreover, baselines are unable to catch complicated drift scenarios in *NSL*, *UNSW* and *Syn.*

Table 3: AUC-PR and Time required to run MemStream and Streaming Baselines on *NSL-KDD*. MemStream provides statistically significant (p value $<$ 0.001) improvements over baseline methods.

Method	AUC-PR	Time (s)
STORM	0.681 ± 0.000	754
HS-Tree	0.709 ± 0.063	306
iForestASD	0.534 ± 0.000	19876
RS-Hash	0.500 ± 0.140	892
RCF	0.664 ± 0.006	665
LODA	0.734 ± 0.067	2617
Kitsune	0.673 ± 0.000	821
DILOF	0.822 ± 0.000	260
xStream	0.541 ± 0.070	34
MStream	0.510 ± 0.000	0.08
Ex. IF	0.659 ± 0.014	889
MemStream	**0.959** ± 0.002	55

Table 3 reports the running AUC-PR scores of MemStream and baseline methods on the *NSL-KDD* dataset, as well as their corresponding running times. Note that not only does MemStream greatly outperform baselines on AUC-PR, but also does so in a time-efficient manner.

5.2 Concept Drift

We next investigate MemStream's performance under concept drift, particularly how fast it can adapt. As shown in Figure 3 (top), we create a synthetic data set which covers a wide variety of drifts scenarios: (a) point anomalies: $T = 19000$ (b) sudden frequency change:

$T \in [5000, 10000]$ (c) continuous concept drift: $T \in [15000, 17500]$ (d) sudden concept drift due to mean change: $T \in [12500, 15000]$. Anomaly scores are clipped at $T = 12500$ and $T = 19000$ for better visibility.

MemStream is able to handle all the above-mentioned concept drift scenarios as is evident in Figure 3 (bottom). We observe that MemStream assigns high scores corresponding to trend-changing events (e.g. $T = 1000, 5000, 10000$ etc.) which produce anomalies, then with a gradual decrease in scores thereafter as it *adapts* successfully to the new distribution. Note that MemStream can also adapt to periodic streams. For the first cycle of the sine wave $T \in [1000, 2000]$, the anomalous scores are relatively high. However, as more and more normal samples are seen from the sine distribution, MemStream adapts to it.

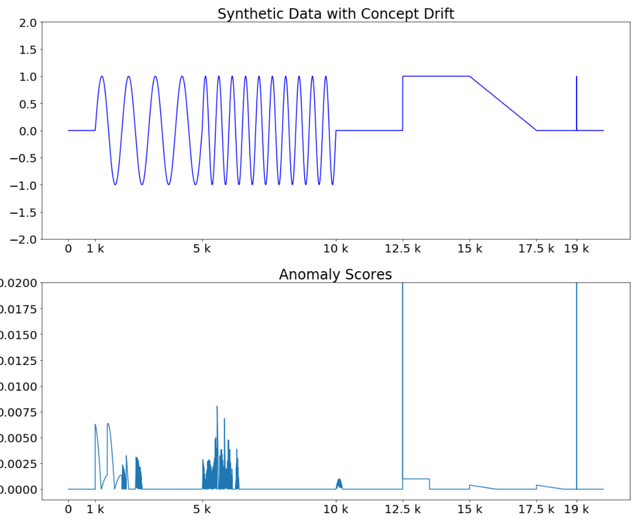

Figure 3: (Top): Synthetic data with drift. (Bottom): Anomaly Scores output by MemStream demonstrating resilience to drift.

5.3 Retraining

The need for re-training is especially prevalent in very long drifting streams where the feature extractor, trained on the small subset of the initial normal data $\mathcal{D}$, starts facing record data sufficiently different from its training data. In this experiment, we test the ability of MemStream to accommodate this more challenging setting by periodically retraining its feature extractor. Fine-tuning is performed at regular intervals distributed uniformly across the stream, i.e. to implement k fine-tunings on a stream of size S, the first fine-tuning occurs at $\lfloor \frac{S}{k+1} \rfloor$. Figure 4 shows the AUC and time taken to fine-tune MemStream on *CICIDS-DoS* with a stream size greater than $1M$ records. Note that as we increase the number of times MemStream is fine-tuned, we observe large gains in AUC with the negligible time difference.

5.4 Self-Correction and Recovery

Consider the scenario where an anomalous element enters into the memory. A particularly catastrophic outcome of this event could

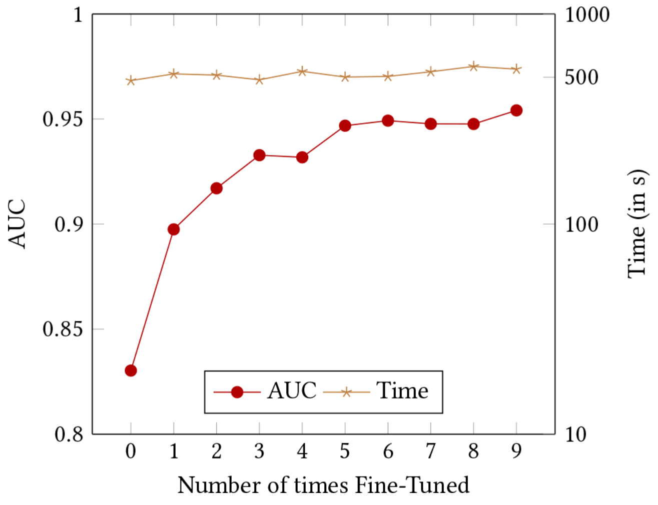

Figure 4: Retraining effect on the AUC and time for *CICIDS-DOS.*

be the cascading effect where more and more anomalous samples replace the normal elements in the memory due to their similarity. This can ultimately lead to a situation where the memory solely consists of anomalous samples. These "Group Anomaly" events are fairly common in intrusion detection settings. We show that this issue is mitigated by the use of K-nearest neighbours in our approach. We simulate the above setting by adding the first labeled anomalous element in memory during the initialization.

In Table 5, a high β allows anomalous elements to also enter the memory. In the absence of K-nearest neighbour discounting (i.e. $\gamma = 0$), a high β value algorithm succumbs to the above-described scenario resulting in poor performance. On the other hand, with discounting (i.e. $\gamma \neq 0$), the algorithm is able to "recover" itself, and as a result, the performance does not suffer considerably. Note that when the threshold β is in its appropriate range, the algorithm is robust to the choice of discount factor γ.

5.5 Ablations

(a) Memory Update: Taking inspiration from the work done in cache replacement policies in computer architecture, we replace the FIFO memory update policy with Least Recently Used (LRU) and Random Replacement (RR) policies. Table 6(a) reports results with these three and when no memory update is performed on the *KDDCUP99* dataset. Note that FIFO outperforms other policies. This is due to the temporal locality preserving property of the FIFO policy to keep track of the current trend. LRU and RR policies do not maintain a true snapshot of the stream in the memory and are thus unable to learn the changing trend.

(b) Feature Extraction: Table 6(b) shows experiments with different methods for feature extraction discussed in Section 4.3. Autoencoder outperforms both PCA and Information Bottleneck approaches.

(c) Memory Length (N): As we noted in Section 4.5.1, increasing N can decrease the false positive rate, but also increase the false negative rate. We observe this effect empirically in Table 6(c),

Table 4: Effect of Memory Size on the AUC in MemStream on *NSL-KDD* dataset.

Memory Size	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
AUC	0.670	0.649	0.932	0.936	0.923	0.950	0.972	0.976	0.985	0.989	0.991

Table 5: Performance of MemStream on *NSL-KDD* dataset after adding an anomalous element in memory when $K = 3$ and for different values of discount factor γ.

γ	High $\beta(=1)$	Appropriate $\beta(=0.001)$
0	0.771	0.933
0.25	0.828	0.966
0.5	0.848	0.967
1	0.888	0.965

Table 6: Ablation study for different components of MemStream on *KDDCUP99*.

	Component	Ablations			
(a)	Memory Update	None 0.938	LRU 0.946	RR 0.946	FIFO 0.980
(b)	Feature Extraction	Identity 0.822	PCA 0.863	IB 0.959	AE 0.980
(c)	Memory Length (N)	128 0.950	256 0.980	512 0.946	1024 0.811
(d)	Output Dimension (D)	$d/2$ 0.951	d 0.928	$2d$ 0.980	$5d$ 0.983
(e)	Update Threshold (β)	1 0.980	0.1 0.938	0.01 0.938	0.001 0.938
(f)	KNN coefficient (γ)	0 0.980	0.25 0.939	0.5 0.937	1 0.936

where the sweet spot is found at $N = 256$, and increasing memory length further degrades performance. An additional experiment demonstrating the effect of memory size is discussed in Table 4. We note that very large or very small values of N would hinder the algorithm performance as the memory will not be able to capture the current trend properly. A very large 'N' will not ensure that the current trend is learned exclusively and the memory would always be contaminated by representatives of the previous trend. On the other hand, a very small 'N' will not allow enough representatives from the current trend and thus in both cases, the performance of the algorithm will be sub-optimal.

(d) Output Dimension (D): In Section 4.5.2, we motivate why we use an architecture with $D >= d$. In Table 6(d), we compare architectures with different output dimension D as a function of the input dimension d. We find that $D = d/2$ outperforms an architecture with $D = d$, owing to the features learning by dimensionality reduction. Note that MemStream performs well for large D.

(e) Update Threshold (β): The update threshold is used to judge records based on their anomaly scores and determine whether they should update the memory. A high β corresponds to frequent updates to the memory, whereas a low β seldom allows memory updates. Thus, β can capture our belief about how frequently the memory should be updated, or how close is the stream to the initial data distribution. From Table 6(e), we notice that for *KDDCUP99*, a drifting dataset, a more flexible threshold ($\beta = 1$) performs well, and more stringent thresholds perform similar to no memory updates (Table 6(a)).

(f) KNN coefficient (γ): In Section 5.4, we discussed the importance of the KNN coefficient γ in the Self-Recovery Mechanism. Table 6(f) compares different settings of γ, without memory poisoning.

6 CONCLUSION

We propose MemStream, a novel memory augmented feature extractor framework for streaming anomaly detection in multi-dimensional data and concept drift settings. MemStream uses a denoising autoencoder to extract features and a memory module with a FIFO replacement policy to learn the dynamically changing trends. Moreover, MemStream allows quick retraining when the arriving stream becomes sufficiently different from the training data. We give a theoretical guarantee on the relation between the memory size and the concept drift. Furthermore, MemStream prevents memory poisoning by using (1) a discounting K-nearest neighbour memory leading to a unique self-correcting and recovering mechanism; (2) a theoretically motivated architecture design choice. MemStream outperforms 11 state-of-the-art streaming methods. Future work could consider more tailored memory replacement policies, e.g. by assigning different weights to the memory elements.

REFERENCES

[1] Charu C Aggarwal. 2015. Outlier analysis. In *Data mining*.

[2] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. 2018. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *ACCV*.

[3] Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* (2015).

[4] Fabrizio Angiulli and Fabio Fassetti. 2007. Detecting distance-based outliers in streams of data. In *CIKM*.

[5] Liang Bai, Xueqi Cheng, Jiye Liang, and Huawei Shen. 2016. An Optimization Model for Clustering Categorical Data Streams with Drifting Concepts. *TKDE* (2016).

[6] Vic Barnett and Toby Lewis. 1984. Outliers in statistical data. (1984).

[7] Md Abul Bashar and Richi Nayak. 2020. TAnoGAN: Time Series Anomaly Detection with Generative Adversarial Networks. *SSCI* (2020).

[8] Laura Beggel, Michael Pfeiffer, and Bernd Bischl. 2019. Robust anomaly detection in images using adversarial autoencoders. In *ECMLPKDD*.

[9] András A. Benczúr, Levente Kocsis, and Róbert Pálovics. 2019. Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning. In *Encyclopedia of Big Data Technologies*.

[10] Yoshua Bengio. 2009. *Learning Deep Architectures for AI*. Now Publishers Inc.

[11] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. MIDAS: Microcluster-Based Detector of Anomalies in Edge Streams. In *AAAI*.

[12] Siddharth Bhatia, Arjit Jain, Pan Li, Ritesh Kumar, and Bryan Hooi. 2021. MSTREAM: Fast Anomaly Detection in Multi-Aspect Streams. *TheWebConf (WWW)* (2021).

[13] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *SIGMOD*.

[14] E Candès, X Li, Y Ma, and J Wright. 2011. Robust principal component analysis? *JACM* (2011).

[15] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep Learning for Anomaly Detection: A Survey. *ArXiv* abs/1901.03407 (2019).

[16] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. 2018. Anomaly Detection using One-Class Neural Networks. *ArXiv* abs/1802.06360 (2018).

[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Computing Survey* (2009).

[18] Lianhua Chi, Bin Li, Xingquan Zhu, Shirui Pan, and Ling Chen. 2018. Hashing for Adaptive Real-Time Graph Stream Classification With Concept Drifts. *IEEE Transactions on Cybernetics* (2018).

[19] KDD Cup Dataset. 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[20] L. Deecke, R. Vandermeulen, L. Ruff, S. Mandt, and M. Kloft. 2018. Image Anomaly Detection with Generative Adversarial Networks. In *ECMLPKDD*.

[21] Zhiguo Ding and Minrui Fei. 2013. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data Using Sliding Window. In *ICONS*.

[22] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[23] Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. 2016. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognition* (2016).

[24] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. van den Hengel. 2019. Memorizing Normality to Detect Anomaly: Memory-Augmented Deep Autoencoder for Unsupervised Anomaly Detection. *ICCV* (2019).

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press Cambridge.

[26] Adam Goodge, Bryan Hooi, See Kiong Ng, and Wee Siong Ng. 2021. LUNAR: Unifying Local Outlier Detection Methods via Graph Neural Networks. *arXiv preprint arXiv:2112.05355* (2021).

[27] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. 2016. Robust Random Cut Forest Based Anomaly Detection on Streams. In *ICML*.

[28] S. Günter, N. N. Schraudolph, and S. Vishwanathan. 2007. Fast Iterative Kernel Principal Component Analysis. *JMLR* (2007).

[29] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. 2014. Outlier Detection for Temporal Data: A Survey. *TKDE* (2014).

[30] S. Hariri, M. Kind, and R. Brunner. 2021. Extended Isolation Forest. *TKDE* (2021).

[31] Geoffrey E Hinton and Richard S Zemel. 1994. Autoencoders, minimum description length and Helmholtz free energy. In *NIPS*.

[32] Ian T Jolliffe. 1986. Principal components in regression analysis. In *Principal component analysis*.

[33] Jaechul Kim and Kristen Grauman. 2009. Observe locally, infer globally: A space-time MRF for detecting abnormal activities with incremental updates. In *CVPR*.

[34] J. Kim and C. D. Scott. 2012. Robust kernel density estimation. *JMLR* (2012).

[35] Artemy Kolchinsky, Brendan D. Tracey, and David H. Wolpert. 2019. Nonlinear Information Bottleneck.

[36] Rithesh Kumar, Anirudh Goyal, Aaron C Courville, and Yoshua Bengio. 2019. Maximum Entropy Generators for Energy-Based Models. *ArXiv* abs/1901.08508 (2019).

[37] Swee Kiat Lim, Yi Loo, Ngoc-Trung Tran, Ngai-Man Cheung, Gemma Roig, and Yuval Elovici. 2018. DOPING: Generative Data Augmentation for Unsupervised Anomaly Detection with GAN. *ICDM* (2018).

[38] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. *ICDM* (2008).

[39] Miodrag Lovric. 2011. *International Encyclopedia of Statistical Science.* Springer.

[40] Cewu Lu, Jianping Shi, and Jiaya Jia. 2013. Abnormal Event Detection at 150 FPS in MATLAB. *ICCV* (2013).

[41] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *TKDE* (2019).

[42] Vijay Mahadevan, Weixin Li, Viral Bhalodia, and Nuno Vasconcelos. 2010. Anomaly detection in crowded scenes. *CVPR* (2010).

[43] Emaad Manzoor, Hemank Lamba, and Leman Akoglu. 2018. xStream: Outlier Detection in Feature-Evolving Data Streams. *KDD* (2018).

[44] Leandro L Minku and Xin Yao. 2011. DDD: A new ensemble approach for dealing with concept drift. *TKDE* (2011).

[45] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. *NDSS* (2018).

[46] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *MilCIS*.

[47] Gyoung S Na, Donghyun Kim, and Hwanjo Yu. 2018. DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams. *KDD* (2018).

[48] P. C. Ngo, A. A. Winarto, C. K. L. Kou, S. Park, F. Akram, and H. K. Lee. 2019. Fence GAN: Towards Better Anomaly Detection. *ICTAI* (2019).

[49] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel. 2020. Deep learning for anomaly detection: A review. *arXiv preprint arXiv:2007.02500* (2020).

[50] Ravdeep Pasricha, Ekta Gujral, and Evangelos E. Papalexakis. 2018. Identifying and Alleviating Concept Drift in Streaming Tensor Decomposition. In *ECML/PKDD*.

[51] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine Learning in Python. *JMLR* (2011).

[52] T. Pevný. 2015. Loda: Lightweight on-line detector of anomalies. *Machine Learning* (2015).

[53] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Latecki. 2007. Incremental Local Outlier Detection for Data Streams. *CIDM* (2007).

[54] M. Ravanbakhsh, E. Sangineto, M. Nabi, and N. S. 2019. Training Adversarial Discriminators for Cross-Channel Abnormal Event Detection in Crowds. *WACV* (2019).

[55] Shebuti Rayana. 2016. ODDS Library. http://odds.cs.stonybrook.edu

[56] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. 2019. A survey of network-based intrusion detection data sets. *Computers & Security* (2019).

[57] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep One-Class Classification. In *ICML*.

[58] M. Salehi, C. Leckie, J. Bezdek, T. Vaithianathan, and X. Zhang. 2016. Fast Memory Efficient Local Outlier Detection in Data Streams. *TKDE* (2016).

[59] Saket K. Sathe and Charu Aggarwal. 2016. Subspace Outlier Detection in Linear Time with Randomized Hashing. *ICDM* (2016).

[60] S. Saurav, P. Malhotra, V. TV, N. Gugulothu, L. Vig, P. Agarwal, and G. Shroff. 2018. Online anomaly detection with concept drift adaptation using recurrent neural networks. *CODS-COMAD* (2018).

[61] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *IPMI*.

[62] B. Schölkopf, J. C Platt, J. Shawe-Taylor, A. J Smola, and R. C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* (2001).

[63] B. Schölkopf, R. C Williamson, A. J Smola, J. Shawe-Taylor, John C Platt, et al. 2000. Support vector method for novelty detection. In *NIPS*.

[64] Junming Shao, Zahra Ahmadi, and Stefan Kramer. 2014. Prototype-based learning on concept-drifting data streams. *KDD* (2014).

[65] I. Sharafaldin, A. H. Lashkari, and A. A Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSP*.

[66] M. Sölch, J. Bayer, M. Ludersdorfer, and P. van der Smagt. 2016. Variational Inference for Online Anomaly Detection in High-Dimensional Time Series. *ArXiv* abs/1602.07109 (2016).

[67] Abhinav Srivastava, Amlan Kundu, Shamik Sural, and Arun Majumdar. 2008. Credit card fraud detection using hidden Markov model. *TDSC* (2008).

[68] Y Su, Y Zhao, C Niu, R Liu, and et al. 2019. Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network. *KDD* (2019).

[69] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast Anomaly Detection for Streaming Data. In *IJCAI*.

[70] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast Anomaly Detection for Streaming Data. In *IJCAI*.

[71] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. *CISDA* (2009).

[72] Naftali Tishby, Fernando C Pereira, and William Bialek. 2000. The information bottleneck method. *arXiv preprint physics/0004057* (2000).

[73] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. 2008. Extracting and Composing Robust Features with Denoising Autoencoders. In *ICML*.

[74] Graham Williams, Rohan Baxter, Hongxing He, Simon Hawkins, and Lifang Gu. 2002. A comparative study of RNN for outlier detection in data mining. In *ICDM*.

[75] Liang Xiong, Barnabás Póczos, and Jeff Schneider. 2011. Group Anomaly Detection using Flexible Genre Models. In *NIPS*.

[76] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. *WWW* (2018).

[77] Z. Yang, T. Zhang, I. S. Bozchalooi, and E. Darve. 2020. Memory Augmented Generative Adversarial Networks for Anomaly Detection. *ArXiv* abs/2002.02669 (2020).

[78] Selim F Yilmaz and Suleyman S Kozat. 2020. PySAD: A Streaming Anomaly Detection Framework in Python. *ArXiv* abs/2009.02572 (2020).

[79] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. 2018. Adversarially Learned Anomaly Detection. *ICDM* (2018).

[80] Shuangfei Zhai, Yu Cheng, Weining Lu, and Zhongfei Zhang. 2016. Deep structured energy based models for anomaly detection. In *ICML*.
[81] Yiru Zhao, Bing Deng, Chen Shen, Yao Liu, Hongtao Lu, and Xian-Sheng Hua. 2017. Spatio-temporal autoencoder for video anomaly detection. *ACM MM* (2017).
[82] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *KDD*.
[83] A. Zimek, E. Schubert, and H.-P. Kriegel. 2012. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining* (2012).
[84] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection. In *ICLR*.

APPENDIX

A PROOFS

Proposition 1. *Define $S_{t,\epsilon} = \{x \in \mathbb{R}^d : \|x - \mu_t\|_2 \leq \sigma\sqrt{d(1+\epsilon)}\}$. Let $(\mu_t)_t$ be the sequence such that there exits a positive real number α for which $\|\mu_t - \mu_{t'}\|_2 \geq (t'-t)\alpha$ for any $t < t'$. Let $\tau > \frac{2\sigma\sqrt{d(1+\epsilon)}}{\alpha}$ and $x_t \sim \mathcal{N}(\mu_t, \sigma I)$ for all $t \in \mathbb{N}^+$. Then, for any $\epsilon > 0$ and $t \in \mathbb{N}^+$, with probability at least $1 - 2\exp(-d\epsilon^2/8)$, the following holds: $x_t \in S_{t,\epsilon}$ and $x_{t+\tau} \notin S_{t,\epsilon}$.*

Proof. Let us write $\bar{d}(x, x') = \|x - x'\|_2$. Then, by the triangle inequality,

$$\bar{d}(\mu_t, \mu_{t+\tau}) \leq \bar{d}(\mu_t, x_{t+\tau}) + \bar{d}(x_{t+\tau}, \mu_{t+\tau}). \tag{1}$$

By using the property of the Gaussian distribution with $z_{t+\tau} \sim \mathcal{N}(0, I)$, we have that

$$\begin{aligned}&\Pr(\|x_{t+\tau} - \mu_{t+\tau}\|_2 < \sigma\sqrt{d(1+\epsilon)})\\ &= \Pr(\|\sigma z_{t+\tau} + \mu_{t+\tau} - \mu_{t+\tau}\|_2 < \sigma\sqrt{d(1+\epsilon)})\\ &= \Pr(\|z_{t+\tau}\|_2^2 < d(1+\epsilon)).\end{aligned}$$

Thus, using the Chernoff bound for the Standard normal distribution for $z_{t+\tau} \sim \mathcal{N}(0, I)$, we have that

$$\Pr(\|x_{t+\tau} - \mu_{t+\tau}\|_2 > \sigma\sqrt{d(1+\epsilon)}) \leq \exp\left(-\frac{d\epsilon^2}{8}\right).$$

Similarly,

$$\Pr(\|x_t - \mu_t\|_2 > \sigma\sqrt{d(1+\epsilon)}) \leq \exp\left(-\frac{d\epsilon^2}{8}\right).$$

By tanking union hounds, we have that with probability at least $1 - 2\exp(-d\epsilon^2/8)$,

$$\|x_{t+\tau} - \mu_{t+\tau}\|_2 \leq \sigma\sqrt{d(1+\epsilon)}, \tag{2}$$

and

$$\|x_t - \mu_t\|_2 \leq \sigma\sqrt{d(1+\epsilon)}. \tag{3}$$

By using the upper bound of (2) in (1), we have that $\bar{d}(\mu_t, \mu_{t+\tau}) \leq \bar{d}(\mu_t, x_{t+\tau}) + \sigma\sqrt{d(1+\epsilon)}$, which implies that

$$\bar{d}(\mu_t, \mu_{t+\tau}) - \sigma\sqrt{d(1+\epsilon)} \leq \bar{d}(\mu_t, x_{t+\tau}).$$

Using the assumption on $(\mu_t)_t$,

$$\tau\alpha - \sigma\sqrt{d(1+\epsilon)} \leq \bar{d}(\mu_t, x_{t+\tau}).$$

Using the definition of τ,

$$\sigma\sqrt{d(1+\epsilon)} < \bar{d}(\mu_t, x_{t+\tau}).$$

This means that $x_{t+\tau} \notin S_{t,\epsilon}$. On the other hand, equation (3) shows that $x_t \in S_{t,\epsilon}$. □

Proposition 2. *Let x be a differentiable point of f_θ such that $\mathcal{B}_r(x) \subseteq \Omega(x)$ for some $r > 0$. If $d > D$, then there exists a $\delta \in \mathbb{R}^d$ such that for any $\hat{x} \in \mathbb{R}^d$ and $\bar{\beta} > 0$, the following holds: $\|\delta\|_2 = r$ and*

$$R(x, \hat{x}) < \bar{\beta} \implies R(x+\delta, \hat{x}) < \bar{\beta}.$$

Proof. We can rewrite the output of the function as $f_\theta(x) = \dot{\sigma}^{[L]}(x,\theta)W^{[L]}\dot{\sigma}^{[L-1]}(x,\theta)W^{[L-1]}\cdots W^{[2]}\dot{\sigma}^{[1]}(x,\theta)W^{[1]}x$. Thus, for any δ such that $(x+\delta) \in \mathcal{B}_r(x) \subseteq \Omega(x)$, we have

$$\begin{aligned}f_\theta(x+\delta) &= \dot{\sigma}^{[L]}(x+\delta,\theta)W^{[L]}\dot{\sigma}^{[L-1]}(x+\delta,\theta)W^{[L-1]}\cdots\\ &\quad W^{[2]}\dot{\sigma}^{[1]}(x+\delta,\theta)W^{[1]}(x+\delta)\\ &= \sigma^{[L]}(x,\theta)W^{[L]}\dot{\sigma}^{[L-1]}(x,\theta)W^{[L-1]}\cdots\\ &\quad W^{[2]}\dot{\sigma}^{[1]}(x,\theta)W^{[1]}(x+\delta)\\ &= Mx + M\delta\end{aligned}$$

where $M = \sigma^{[L]}(x,\theta)W^{[L]}\dot{\sigma}^{[L-1]}(x,\theta)W^{[L-1]}$ $\cdots W^{[2]}\dot{\sigma}^{[1]}(x,\theta)W^{[1]}$. Notice that M is a matrix of size D by d. Thus, ff $d > D$, there the nulls space (or the kernel space) of M is not $\{0\}$ and there exists $\delta' \in \mathbb{R}^d$ in the null space of M such that $\|\delta'\| \neq 0$ and $M(r'\delta') = 0$ for all $r' > 0$. Thus, there exists a $\delta \in \mathbb{R}^d$ such that $(x+\delta) \in \mathcal{B}_r(x) \subseteq \Omega(x)$, $\|\delta\|_2 = r$, and $M\delta = 0$, yielding

$$f_\theta(x+\delta) = Mx = f_\theta(x).$$

This implies the statement of this proposition. □

B DATASETS

Table 7 contains the datasets that we use for evaluation. We briefly describe how these datasets are prepared for anomaly detection.

(1) *KDDCUP99* [19] is based on the DARPA data set and is amongst the most extensively used data sets for multi-aspect anomaly detection. The original dataset contains samples of 41 dimensions, 34 of which are continuous and 7 are categorical, and also displays concept drift [44]. We use one-hot representation to encode the categorical features, and eventually, we obtain a dataset of 121 dimensions. For the *KDDCUP99* dataset, we follow the settings in [84]. As 20% of data samples are labeled as "normal" and 80% are labeled as "attack", normal samples are in a minority group; therefore, we treat normal ones as anomalous in this experiment and the 80% samples labeled as attack in the original dataset are treated as normal samples.
(2) *NSL-KDD* [71] solves some of the inherent problems of the *KDDCUP99* dataset such as redundant and duplicate records and is considered more enhanced as compared to *KDDCUP99*.
(3) *CICIDS-DoS* [65] was created by the Canadian Institute of Cybersecurity. Each record is a flow containing features such as Source IP Address, Source Port, Destination IP Address, Bytes, Packets. These flows were captured from a real-time simulation of normal network traffic and synthetic attack simulators. This consists of the *CICIDS-DoS* dataset (1.05 million records). *CICIDS-DoS* has 5% anomalies and contains samples of 95 dimensions with a mixture of numeric and

Table 7: Statistics of the datasets.

	KDD99	NSL	UNSW	DoS	Syn.	Ion.	Cardio	Sat.	Sat.-2	Mamm.	Pima	Cover
Records	494, 021	125, 973	2, 540, 044	1, 048, 575	10, 000	351	1831	6435	5803	11183	768	286048
Dimensions	121	126	122	95	1	33	21	36	36	6	8	10

Table 8: Memory Length and Update Threshold used for the different datasets

Method	KDD99	NSL	UNSW	DoS	Syn.	Ion.	Cardio	Sat.	Sat.-2	Mamm.	Pima	Cover
N	256	2048	2048	2048	16	4	64	32	256	128	64	2048
β	1	0.1	0.1	0.1	1	0.001	1	0.01	10	0.1	0.001	0.0001

categorical features. For categorical features, we further used binary encoding to represent them because of the high cardinality.

(4) *UNSW-NB15* [46] was created by the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviors. This dataset has nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. It has 13% anomalies.

(5) Ionosphere [55] is derived using the ionosphere dataset from the UCI ML repository [22] which is a binary classification dataset with dimensionality 34. There is one attribute having values of all zeros, which is discarded. So the total number of dimensions is 33. The 'bad' class is considered as outliers class and the 'good' class as inliers.

(6) Cardio [55] is derived using the Cardiotocography (Cardio) dataset from the UCI ML repository [22] which consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians. This is a classification dataset, where the classes are normal, suspect, and pathologic. For outlier detection, the normal class formed the inliers, while the pathologic (outlier) class is downsampled to 176 points. The suspect class is discarded.

(7) Satellite [55] is derived using the Statlog (Landsat Satellite) dataset from the UCI ML repository [22] which is a multi-class classification dataset. Here, the training and test data are combined. The smallest three classes, i.e. 2, 4, 5 are combined to form the outliers class, while all the other classes are combined to form an inlier class.

(8) Satimage-2 [55] is derived using the Statlog (Landsat Satellite) dataset from the UCI ML repository [22] which is also a multi-class classification dataset. Here, the training and test data are combined. Class 2 is down-sampled to 71 outliers, while all the other classes are combined to form an inlier class. The modified dataset is referred to as Satimage-2.

(9) Mammography [55] is derived from openML[2]. The publicly available openML dataset has 11, 183 samples with 260 calcifications. If we look at predictive accuracy as a measure of goodness of the classifier for this case, the default accuracy would be 97.68% when every sample is labeled non-calcification. But, it is desirable for the classifier to predict most of the calcifications correctly. For outlier detection, the minority class of calcification is considered as the outlier class and the non-calcification class as inliers.

(10) Pima [55] is the same as Pima Indians diabetes dataset of the UCI ML repository [22] which is a binary classification dataset. Several constraints were placed on the selection of instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

(11) ForestCover [55] is the ForestCover/Covertype dataset from the UCI ML repository [22] which is a multiclass classification dataset. It is used in predicting forest cover type from cartographic variables only. This dataset has 54 attributes (10 quantitative variables, 4 binary wilderness areas, and 40 binary soil type variables). Here, an outlier detection dataset is created using only 10 quantitative attributes. Instances from class 2 are considered as normal points and instances from class 4 are anomalies. The anomalies ratio is 0.9%. Instances from the other classes are omitted.

C MEMORY SIZE AND UPDATE THRESHOLDS

Table 8 shows the memory size N, and the value of the threshold β.

D BASELINE PARAMETERS

STORM: window_size=10000, max_radius=0.1
HS-Tree: window_size=100, num_trees=25, max_depth=15, initial_window_X=None
iForestASD: window_size=100, n_estimators=25, anomaly_threshold=0.5, drift_threshold=0.5
RS-Hash: sampling_points=1000, decay=0.015, num_components=100, num_hash_fns=1
RCF: num_trees=4, shingle_size=4, tree_size=256

[2]https://www.openml.org/

LODA: num_bins=10, num_random_cuts=100
Kitsune: max_size_ae=10, learning_rate=0.1, hidden_ratio=0.75, grace_feature_mapping=grace_anomaly_detector=10% of data
DILOF: window size = 400, thresholds = [0.1f, 1.0f, 1.1f, 1.15f, 1.2f, 1.3f, 1.4f, 1.6f, 2.0f, 3.0f] , K = 8
xStream: projection size=50, number of chains=50, depth=10, rowstream=0, nwindows=0, initial sample size=# rows in data, scoring batch size=100000
MStream: alpha = 0.85
Ex. IF: ntrees=200, sample_size=256, limit=None, ExtensionLevel=1