Intrusion detection based on concept drift detection and online incremental learning

Farah Jemili, Khaled Jouini and Ouajdi Korbaa
ISITCom, Mars Research Laboratory, LR17ES05, University of Sousse, Sousse, Tunisia

Received 26 December 2023
Revised 14 July 2024
Accepted 29 August 2024

Abstract

Purpose – The primary purpose of this paper is to introduce the drift detection method-online random forest (DDM-ORF) model for intrusion detection, combining DDM for detecting concept drift and ORF for incremental learning. The paper addresses the challenges of dynamic and nonstationary data, offering a solution that continuously adapts to changes in the data distribution. The goal is to provide effective intrusion detection in real-world scenarios, demonstrated through comprehensive experiments and evaluations using Apache Spark.

Design/methodology/approach – The paper uses an experimental approach to evaluate the DDM-ORF model. The design involves assessing classification performance metrics, including accuracy, precision, recall and F-measure. The methodology integrates Apache Spark for distributed computing, using metrics such as processed records per second and input rows per second. The evaluation extends to the analysis of IP addresses, ports and taxonomies in the MAWILab data set. This comprehensive design and methodology showcase the model's effectiveness in detecting intrusions through concept drift detection and online incremental learning on large-scale, heterogeneous data.

Findings – The paper's findings reveal that the DDM-ORF model achieves outstanding classification results with 99.96% accuracy, demonstrating its efficacy in intrusion detection. Comparative analysis against a convolutional neural network-based model indicates superior performance in anomalous and suspicious detection rates. The exploration of IP addresses, ports and taxonomies uncovers valuable insights into attack patterns. Apache Spark evaluation attests to the system's high processing rates. The study emphasizes the scalability, availability and fault tolerance of DDM-ORF, making it suitable for real-world scenarios. Overall, the paper establishes the model's proficiency in handling dynamic, nonstationary data for intrusion detection.

Research limitations/implications – The research acknowledges certain limitations, including the potential challenge of DDM detecting only frequency changes in class labels and not complex concept drifts. The incremental random forest's reliance on memory may pose constraints as the forest size increases, potentially leading to overfitting. Addressing these limitations could involve exploring alternative concept drift detection algorithms and implementing ensemble pruning techniques for memory efficiency. Further research avenues may investigate algorithms balancing accuracy and memory usage, such as compressed random forests, to enhance the model's effectiveness in evolving data environments.

Practical implications – The study's practical implications are noteworthy. The proposed DDM-ORF model, designed for intrusion detection through concept drift detection and online incremental learning, offers a scalable, available and fault-tolerant solution. Leveraging Apache Spark and Microsoft Azure Cloud

Ethical statements: The authors declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere. The authors declare that they have no funding. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The authors have read and agree to the Terms and Conditions of the journal.

Data availability: The data sets generated during and/or analyzed during the current study are available from the authors.

International Journal of Pervasive Computing and Communications
Vol. 21 No. 1, 2025
pp. 81-115
© Emerald Publishing Limited
1742-7371
DOI 10.1108/IJPCC-12-2023-0358

enhances processing capabilities for large data sets in dynamic, nonstationary scenarios. The model's applicability to heterogeneous data sets and its achievement of high-accuracy multi-class classification make it suitable for real-world intrusion detection. Moreover, the auto-scaling features of Microsoft Azure Cloud contribute to adaptability, ensuring efficient resource utilization without downtime. These practical implications underscore the model's relevance and effectiveness in diverse operational contexts.

Social implications – The DDM-ORF model's social implications are significant, contributing to enhanced cybersecurity measures. By providing an effective intrusion detection system, it helps safeguard digital ecosystems, preserving user privacy and securing sensitive information. The model's accuracy in identifying and classifying various intrusion attempts aids in mitigating potential cyber threats, thereby fostering a safer online environment for individuals and organizations. As cybersecurity is paramount in the digital age, the social impact lies in fortifying the resilience of networks, systems and data against malicious activities, ultimately promoting trust and reliability in online interactions.

Originality/value – The DDM-ORF model introduces a novel approach to intrusion detection by combining drift detection and online incremental learning. This originality lies in its utilization of the DDM-ORF algorithm, offering a dynamic and adaptive system for evolving data. The model's contribution extends to its scalability, fault-tolerance and suitability for heterogeneous data sets, addressing challenges in dynamic, nonstationary environments. Its application on a large-scale data set and multi-class classification, along with integration with Apache Spark and Microsoft Azure Cloud, enhances the field's understanding and application of intrusion detection, providing valuable insights for securing digital infrastructures.

Keywords Intrusion detection, Concept drift detection, Online incremental learning

Paper type Research paper

1. Introduction

In today's complex landscape of computer networks, the significance of intrusion detection systems (IDS) cannot be overstated – they play a pivotal role in identifying and mitigating potential threats to network security (Mahdavi *et al.*, 2022). With the escalating frequency and sophistication of cyberattacks, IDS has become indispensable for network administrators striving to uphold the confidentiality, integrity and availability of their systems. Operating on the analysis of network traffic, IDS discerns patterns indicative of potential security breaches. Upon detecting an intrusion, the IDS promptly alerts network administrators, empowering them to take necessary actions to counteract the threat (Folino *et al.*, 2020).

Yet, despite their crucial role, IDS grapple with several challenges due to the continually evolving nature of attacks and the emergence of new attack patterns collectively termed as concept drift. Traditional IDS find it challenging to adapt to these changes, necessitating frequent manual updates that prove to be time-consuming and costly (Dwibedi *et al.*, 2020). For instance, an IDS trained to identify a specific type of attack may lose effectiveness if attackers alter their tactics or use new methods. In such scenarios, the IDS demands updates with new rules or models to stay abreast of the evolving threat landscape (Guarino *et al.*, 2022).

To counter this challenge, a novel approach grounded in incremental learning and concept drift detection has been proposed. Concept drift, a phenomenon where the statistical properties of the analyzed data change over time, poses a challenge to maintaining IDS accuracy. Incremental learning, a machine learning technique, addresses this by dynamically updating the model as new data is introduced. The integration of concept drift detection and incremental learning empowers an IDS to continuously learn from new data, adapting to shifts in network behavior and effectively identifying previously unknown threats (Nugroho *et al.*, 2020). Enter our proposed approach, drift detection method-online random forest (DDM-ORF), designed to markedly enhance IDS accuracy and efficacy – an invaluable tool in the ongoing battle against cyberattacks.

DDM-ORF is crafted to be adaptive, detecting concept drifts and seamlessly updating the model with incoming data to uphold accuracy. This approach boasts numerous advantages over traditional IDS, including heightened accuracy, scalability and the ability to navigate dynamic shifts in data. Leveraging online incremental learning, DDM-ORF updates the model in real-time, proving particularly effective in detecting new and emerging threats. Moreover, with concept drift detection, this approach can discern alterations in data distribution, adjusting the model accordingly. Given the ever-evolving nature of cyber threats, possessing IDS capable of keeping pace with the changing threat landscape is imperative, and DDM-ORF presents a promising solution to this critical issue.

The paper unfolds in six sections: commencing with the introduction, Section 2 delves into a review of related works on IDS. Section 3 elucidates the concept drift detection and online incremental learning techniques underpinning the proposed approach. Following that, Section 4 meticulously details the proposed approach itself. Section 5 outlines the experimental setup used to evaluate the proposed approach. Finally, the last section concludes the paper, emphasizing its contributions and suggesting future research directions.

2. Related work

In Yuan *et al.* (2018), the authors introduce an innovative intrusion detection approach that integrates ensemble incremental learning to grapple with the challenge of concept drift. Acknowledging the need for IDS to adapt to new attack patterns and address concept drift, the authors propose an ensemble incremental learning approach. This involves deploying a set of classifiers that incrementally learn from new data while updating their knowledge of previously learned attack patterns. Central to this approach is a concept drift detection module that vigilantly monitors the incoming data stream for shifts in the underlying concept. Upon detecting a concept drift, the system triggers an incremental learning process to align the ensemble of classifiers with new attack patterns. The authors introduce a weighted voting scheme, dynamically adjusting classifier weights based on historical performance and concept drift significance. Results showcase the superior accuracy and adaptability of the proposed ensemble incremental learning approach over traditional IDS methods. However, the reliance on batch learning limits its efficacy in handling evolving data streams, and the binary classification scope may be constraining in scenarios requiring more nuanced classification.

In Liu *et al.* (2023), the focus shifts to developing an IDS tailored for Internet of Things (IoT) environments. Addressing the challenge of securing interconnected IoT networks, the authors propose an adaptive class incremental learning-based IDS. This system incrementally learns and adapts to new attack classes while maintaining high detection accuracy. Using a feature extraction module to capture relevant characteristics from network traffic data, the IDS integrates a class incremental learning algorithm that combines deep neural networks with an incremental learning approach. This mechanism facilitates learning new attack classes without forgetting previously acquired patterns. An adaptive decision-making component dynamically adjusts detection thresholds based on network conditions, enhancing adaptability and reducing false positives or negatives. While outperforming traditional IDS methods, a potential limitation lies in the initial need for a substantial amount of labeled data and possible performance issues on data sets with significant class imbalance.

In Sun *et al.* (2022), a focus on industrial Internet of Things (IIoT) environments prompts the authors to propose an intrusion detection method rooted in active incremental learning. Aimed at improving accuracy and efficiency in IIoT systems, the method incorporates active learning where human experts interact with the system, providing feedback and labeling samples. An incremental learning algorithm facilitates continuous updates to the system's knowledge, adapting to evolving attack patterns. In addition, a feature selection process enhances efficiency by identifying relevant features. Results exhibit superiority in detection accuracy and efficiency over traditional approaches, yet computational resource demands for the incremental learning algorithm and the focus on a specific type of cyberattack pose challenges.

In Kuppa and Le-Khac (2022), the authors present a robust DDM, “Learn to adapt,” addressing the challenge of detecting and adapting to concept drift in security-related data sets. Utilizing an ensemble of classifiers trained on different data set partitions, the method uses an adaptation algorithm to dynamically adjust weights based on individual classifier performance. Results demonstrate superior detection accuracy and adaptability. However, the approach’s offline nature, requiring historical data for drift detection, may limit real-time applicability.

In Wu *et al.* (2022), a focus on incremental learning for intrusion detection introduces a method based on a dynamic ensemble of relevance vector machines (RVMs). The incremental learning strategy updates knowledge with newly labeled samples, and the dynamic ensemble mechanism adjusts composition based on individual RVM performance. While outperforming traditional approaches, RVMs’ need for sufficient labeled data and their potential black-box nature pose challenges.

In Abdel Wahab (2022), authors present an online deep learning approach for intrusion detection in IoT environments, titled “Intrusion detection in the IoT under data and concept drifts: Online deep learning approach,” published in the IEEE Internet of Things Journal. Their method continuously updates a deep neural network with new data to adapt to evolving attack patterns, achieving high detection accuracy and robustness against concept drift. However, the approach requires substantial computational resources and sophisticated deep learning frameworks, which may not be readily available in all IoT deployment scenarios. In addition, it primarily focuses on binary classification, limiting its scope in identifying and distinguishing between multiple types of attacks.

Numerous approaches have been proposed to address the challenge of concept drift in IDS. For instance, Yuan *et al.* (2018), presents an ensemble incremental learning approach that adapts to new attack patterns. However, this method relies on batch learning, limiting its efficacy in real-time scenarios. Similarly, Liu *et al.* (2023), focus on class incremental learning for IoT environments yet require substantial labeled data for initial training, posing challenges for immediate deployment in diverse settings. In Wu *et al.* (2022), the use of a dynamic ensemble of RVMs for incremental learning also demonstrates improvements but faces challenges due to the need for sufficient labeled data and the potential black-box nature of RVMs. Methods such as Sun *et al.* (2022) and Kuppa and Le-Khac (2022) are limited to binary classification, restricting their applicability in more complex scenarios. Recent works like Abdel Wahab (2022) have explored deep learning approaches for handling concept drift, but these require considerable computational resources and sophisticated frameworks, which may not be readily available in all deployment scenarios.

In contrast, our proposed DDM-ORF method stands out by combining DDM’s real-time drift detection with ORF’s continuous learning capabilities, ensuring that the model adapts instantaneously to new data without the need for batch processing. Furthermore, DDM-ORF’s ability to perform multi-class classification effectively addresses the limitations seen in other methods. Our approach also leverages Apache Spark Structured Streaming, enhancing its scalability and processing speed for handling large-scale, heterogeneous data. These features make our DDM-ORF method not only more effective in diverse and dynamic environments but also more practical for immediate deployment compared to other existing methods.

Table 1 summarizes the key differences between our approach and existing methods, underscoring the unique advantages of DDM-ORF in terms of adaptability, scalability and real-time applicability.

Table 1. Related works

Aspect/feature	Yuan *et al.* (2018) Ensemble incremental learning	Liu *et al.* (2023) Adaptive class incremental learning	Sun *et al.* (2022) Active incremental learning	Kuppa and Le-Khac (2022) Learn to adapt drift detection	Wu *et al.* (2022) Dynamic ensemble of RVMs	Abdel Wahab (2022) Deep learning	DDM-ORF (our approach)
Learning approach	Ensemble incremental learning	Adaptive class incremental learning	Active incremental learning	Ensemble learning for drift detection	Dynamic ensemble of RVMs	Deep learning model (DNN)	Online incremental learning with DDM
Detection accuracy	Superior to traditional IDS methods	Outperforms traditional IDS	Outperforms traditional IDS	Superior detection accuracy	Outperforms traditional IDS	High detection accuracy	Superior accuracy
Data set size sensitivity	Not explicitly mentioned	Requires substantial labeled data	Active learning may require expert interaction	Ensemble trained on partitions of data set	Requires sufficient labeled data	Requires significant amounts of labeled data	Handles massive data
Classification scope	Binary	Binary	Not specified	Binary	Multiclass (dynamic ensemble)	Binary	Multiclass
Real-time applicability	Limited due to batch learning	Real-time applicability not specified	Real-time applicability not specified	Offline detection requires historical data	Real-time applicability not specified	Real-time detection	Real-time detection
Computational resources	Not specified	Potential performance issues with imbalanced data sets	Significant resources for incremental learning	Drift detection requires computational resources	Sufficient labeled data for RVMs	Requires substantial computational resources	Low computational overhead

Source: F. Jemili *et al.*

By integrating DDM with ORF and leveraging Apache Spark Structured Streaming, our DDM-ORF approach addresses critical limitations of existing IDS methods, offering a scalable, real-time solution capable of adapting to the continuously evolving threat landscape. This combination of techniques presents a significant advancement in the field of intrusion detection, providing a robust framework for future research and development in cybersecurity.

3. Concept drift detection and incremental learning

In this section, we describe the concept drift detection and online incremental learning techniques that are used in the literature to make appropriate choices for our contribution.

3.1 Concept drift detection

Concept drift occurs when the target changes over a limited period of time. Consider two target concepts, A and B, and a sequence of samples, I = i_1, i_2, …, i_n. Prior to a certain instance, the target concept remains unchanged in A. However, after the instance, a new concept, *Δ*x, becomes stable and replaces A with B. The speed of this transition can be gradual or abrupt depending on the efficiency of the drift, *Δ*x. There are three different ways to model concept drift: window-related, weight-related and an ensemble of classification models. The former selects samples from a sliding window, while the latter weights samples and removes them based on their weight (Folino *et al.*, 2020).

There are two approaches to handling concept drift: online and batch. The former updates the classifier after each instance, while the latter waits to receive massive instances before starting the learning process.

There are several concept drift detection techniques that are commonly used in machine learning, including Nugroho *et al.* (2020):

- The Kolmogorov–Smirnov (KS) test compares the distribution of the incoming data to a reference distribution and detects significant deviations that may indicate a drift. The KS test can be effective for detecting sudden and significant changes in the data distribution but may be less effective for detecting gradual or subtle changes. The KS statistic measures the maximum distance between the cumulative distribution functions (CDFs) of the two samples:

$$D = \sup_x \, |F_n(x) - F_m(x)| \tag{1}$$

where $F_n(x)$ and $F_m(x)$ are the empirical CDFs of the reference and current samples, respectively.

- The Page-Hinkley Test is a sequential hypothesis testing method that can detect both gradual and sudden changes in the data distribution by monitoring the cumulative sum of deviations from a reference value. It detects shifts in the mean of a data stream:

$$PH_t = \sum_{i=1}^{t} (x_i - \mu - \delta) \tag{2}$$

where x_i is the data point at time i, μ is the mean and δ is the tolerance parameter.

This approach can be effective for detecting both short-term and long-term drift, but may have a higher false positive rate than other methods.

- Exponentially weighted moving average (EWMA): a statistical method that monitors changes in the mean or standard deviation of the incoming data stream and detects significant deviations:

$$S_t = \alpha X_t + (1 - \alpha) S_{t-1} \tag{3}$$

where S_t is the EWMA statistic at time *t*, X_t is the current data point and *α* is the smoothing parameter.

- Adaptive Windowing (ADWIN): a nonparametric method that uses a sliding window to detect changes in the underlying distribution of the data by monitoring the variance of the data within the window. If a significant difference is found, a drift is signaled, and the window is reset;
- Cumulative Sum (CUSUM) is a statistical control chart method that detects changes in the mean of the data by monitoring the cumulative sum of deviations from a reference value:

$$C_t = max\,(0, C_{t-1} + X_t - \kappa) \tag{4}$$

where C_t is the CUSUM statistic at time *t*, X_t is the data point and k is the reference value.

- Hoeffding's Drift Detection Method (HDDM): a hypothesis-testing method that detects changes in the distribution of the data by comparing the mean of two sliding windows of the data. Uses Hoeffding's inequality to determine if two sliding windows of data points have different distributions; and
- DDM can detect drift in real-time with low computational overhead. DDM works by monitoring the error rate of a classification model over time and comparing it to a threshold value. When the error rate exceeds the threshold, it indicates a possible drift, and the model can be updated to adapt to the new data distribution. The error rate ∈ and standard deviation *σ* are tracked, and a drift is signaled if:

$$\in + \sigma > \in_{\min} + 2\sigma_{\min} \tag{5}$$

These techniques are designed to detect different types of concept drift and have varying degrees of sensitivity and computational complexity. It is important to choose the appropriate technique based on the specific application and data characteristics (see Table 2).

DDM is a window-related technique used to detect concept drift in data streams. It works by maintaining a sliding window of the most recent data points and monitoring the changes in the data distribution over time. If the distribution changes significantly within the window, it signals the presence of a concept drift. Here are some reasons why DDM is considered in this contribution (Jemili, 2022):

- *Simplicity and Efficiency*: DDM is a simple and efficient method that can detect drift in real-time with low computational overhead. It works by monitoring the error rate of a classification model over time and comparing it to a threshold value. When the error rate exceeds the threshold, it indicates a possible drift, and the model can be updated to adapt to the new data distribution. This simplicity and efficiency make DDM a practical and scalable approach for real-world applications;

Table 2. Concept drift detection techniques

Technique	Description	Pros	Cons
KS test	Compares incoming data distribution to a reference, effective for sudden changes	– Effective for sudden changes – Provides statistical significance	– May be less effective for gradual changes – Higher false positives for subtle changes
Page-Hinkley test	Sequential hypothesis testing detecting both gradual and sudden changes	– Effective for short-term and long-term drift – Adapts to various drift types	– May have a higher false positive rate
EWMA	Monitors mean or standard deviation changes in incoming data stream	– Detects significant deviations – Provides adaptability to changes	– May not handle complex drift patterns effectively
ADWIN	Nonparametric method using a sliding window to detect changes in data distribution	– Adapts to underlying distribution changes – Real-time drift detection	– Computational resources may be demanding
CUSUM	Statistical control chart method detecting changes in the mean of the data	– Effective for detecting mean changes – Suitable for real-time detection	– May require careful parameter tuning – Sensitive to initial conditions
HDDM	Hypothesis testing method detecting changes in the distribution of the data	– Detects changes effectively – Useful for specific contexts	– Computational overhead may be a concern – Sensitivity to parameter choices
DDM	Window-related technique monitoring the error rate of a classification model over time	– Simplicity and efficiency – Real-time detection capabilities – Robust to noise	– May not perform optimally in all scenarios – Limited by window size

Source: F. Jemili *et al*.

- Real-Time Detection: DDM can detect drift in real-time, which is critical for intrusion detection and other time-sensitive applications. It can quickly identify changes in the data distribution and update the classification model to adapt to the new distribution. This real-time detection capability can improve the accuracy and timeliness of IDS, which can help prevent or mitigate cyberattacks;
- Adaptive Thresholding: DDM uses an adaptive thresholding approach to adjust the detection sensitivity based on the number of observations and the significance level. This approach can reduce the false positive rate and improve the accuracy of drift detection, which is important for reducing the workload of security analysts and avoiding unnecessary alarms; and
- Robustness to Noise: DDM is robust to noise and outliers in the data, which can be common in real-world intrusion detection applications. It can filter out noise and focus on significant changes in the data distribution, which can improve the reliability and effectiveness of the detection system.

DDM is a simple, efficient and effective concept drift detection technique that can provide real-time detection and adaptive thresholding capabilities for intrusion detection and other real-time data analysis applications. Its robustness to noise and scalability make it a practical and reliable approach for detecting concept drift in a variety of settings. In our contribution, we used DDM for concept drift detection.

3.2 Online incremental learning

After drift detection, a suitable drift adaptation algorithm needs to be implemented to handle the detected drifts and maintain high learning performance. Drift adaptation methods can be broadly classified into two main categories: incremental learning methods and ensemble methods (Abid and Jemili, 2020). Incremental learning methods update the model parameters gradually over time to adapt to concept drift, while ensemble methods combine multiple models to improve performance and handle concept drift (Hafsa and Jemili, 2018).

Online incremental learning techniques are used to train models on continuously arriving data, where the model needs to be updated with each new sample. Some of the most popular online incremental learning techniques include D'Angelo *et al.* (2021) (see Table 3):

- *Online Sequential Extreme Learning Machine (OS-ELM)*: OS-ELM is a popular online learning algorithm for training feedforward neural networks. It updates the network parameters using only one sample at a time and has a faster training speed compared to other neural network models. OS-ELM can handle large data sets and noisy data. It involves randomly assigning input weights and biases, then solving the output weights using a least-squares solution:

$$\mathbf{H}\beta = \mathbf{T} \tag{6}$$

where H is the hidden layer output matrix, β is the output weight matrix and T is the target matrix.

- *Incremental and decremental support vector machines (ICU-SVM)*: ICU-SVM is an online learning algorithm for classification and regression tasks. It updates the SVM model parameters using only one sample at a time and can handle nonstationary data. ICU-SVM has high accuracy and is computationally efficient.

Table 3. Online incremental learning techniques

Technique	Description	Pros	Cons
OS-ELM	Efficient online learning algorithm for training feedforward neural networks	– Fast training speed – Suitable for large data sets and noisy data	– Limited interpretability of neural networks – May not perform well with highly imbalanced data sets
ICU-SVM	Online learning algorithm for classification and regression tasks	– High accuracy – Computationally efficient – Handles nonstationary data	– May require tuning of hyperparameters – Limited adaptability to complex data distributions
Hoeffding tree	Online learning algorithm for classification and regression tasks, building decision trees incrementally	– Handles large data sets and noisy data – Fast processing speed	– Limited interpretability of decision trees – May struggle with highly imbalanced data sets
SGD	Popular online learning algorithm used for various machine learning tasks	– Computationally efficient – Suitable for large data sets	– May require careful tuning of hyperparameters – Sensitive to feature scaling – Convergence may be affected by noisy data
Adaptive learning rate methods	Class of online learning algorithms adjusting learning rate based on data characteristics	– Improves convergence rate and accuracy of the model – Handles changing data characteristics effectively	– May require tuning of adaptive learning rate methods – Sensitive to hyperparameter choices – Performance may vary across different data sets
Online Passive-Aggressive algorithm	Online learning algorithm for classification tasks, updating model parameters based on sample margin	– Computationally efficient – Suitable for high-dimensional data	– May require tuning of hyperparameters – Limited interpretability – Sensitivity to feature scaling – May be affected by noisy data
Bayesian methods	Class of online learning algorithms updating model parameters based on Bayesian inference	– Handles uncertainty in data – Improves model generalization	– May require careful tuning of Bayesian methods – Computational overhead may be a concern – Limited interpretability for some Bayesian models
Online random forests	Extension of traditional random Forest for online learning, capable of handling concept drift	– Handles high-dimensional and noisy data – Detects and adapts to concept drift – Low computational cost – Suitable for large-scale data sets	– May require careful tuning of hyperparameters – Limited interpretability of the ensemble – Sensitive to noisy data and outliers – Resource-intensive during training

Source: F. Jemili *et al*.

- *Hoeffding Tree*: Hoeffding Tree is an online learning algorithm for classification and regression tasks. It builds a decision tree incrementally using a statistical test to determine when to split the nodes. Hoeffding Tree can handle large data sets and noisy data and has a fast processing speed. It uses the Hoeffding bound to determine the minimum number of samples required to choose the best splitting attribute with high probability:

$$G(X_i) - G(X_j) > \in \tag{7}$$

where G is the split evaluation measure, and ϵ\epsilonϵ is the Hoeffding bound.

- *Stochastic gradient descent (SGD)*: SGD is a popular online learning algorithm used for a variety of machine learning tasks. It updates the model parameters based on the gradient of the loss function with respect to the parameters. SGD is computationally efficient and can handle large data sets:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t; x_t, y_t) \tag{8}$$

where θ represents model parameters, η is the learning rate and L is the loss function.

- *Adaptive Learning Rate Methods*: Adaptive learning rate methods are a class of online learning algorithms that adjust the learning rate based on the characteristics of the data. These methods include Adagrad, Adadelta, RMSProp and Adam. Adaptive learning rate methods can improve the convergence rate and accuracy of the model.
- *Online Passive-Aggressive (PA) Algorithm*: The PA algorithm is a popular online learning algorithm for classification tasks. It updates the model parameters based on the margin of the sample and the prediction of the current model. The PA algorithm is computationally efficient and can handle high-dimensional data:

$$\theta_{t+1} = \theta_t + \tau y_t x_t \tag{9}$$

where τ is the step size determined by the loss function, and y_t and x_t are the label and feature vector of the current sample.

- *Bayesian Methods*: Bayesian methods are a class of online learning algorithms that update the model parameters based on the Bayesian inference framework. These methods include online Bayesian linear regression, online Bayesian logistic regression and online Bayesian neural networks. Bayesian methods can handle uncertainty in the data and improve the generalization ability of the model.
- *Online Random Forests*: Online random forests are an extension of the traditional random forest algorithm for online learning. They update the model by adding new decision trees to the forest based on the arriving data. Online random forests can handle concept drift and noisy data:

$$\widehat{y} = \frac{1}{n}\sum_{i=1}^{n} T_i(x) \tag{10}$$

where $\widehat{y}$ is the predicted output, n is the number of trees and T_i is the i-th decision tree.

Online random forests (Dhahbi and Jemili, 2021) are a powerful and effective technique for incremental learning. One of the main advantages of online random forests is their ability to handle high-dimensional and noisy data in an online and dynamic environment. They can handle both categorical and continuous data and have the ability to detect and adapt to concept drift, which is a common problem in online learning (Meddeb *et al.*, 2023). In addition, online random forests have a low computational cost and can be trained on large-scale data sets. Overall, online random forests are a flexible and robust technique that can handle a wide range of data types and learning scenarios, making them one of the best choices among the listed techniques (Coccia *et al.*, 2021). In our contribution, we opted for online random forests for online incremental learning.

4. Proposed approach

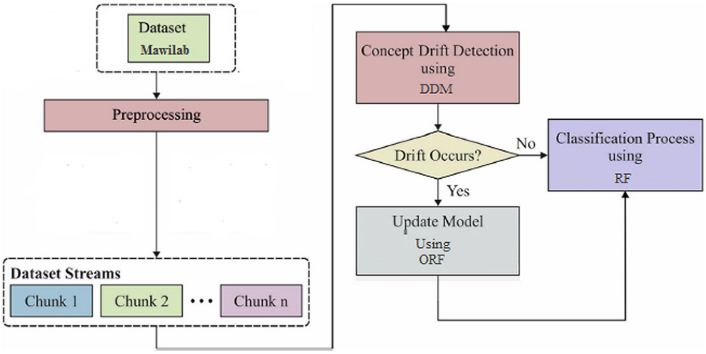
We propose an intrusion detection approach, named DDM-ORF, that combines concept drift detection based on DDM and online incremental learning based on Online Random Forest to improve the accuracy and efficiency of IDS.

Figure 1 illustrates the operational procedure used in the presented approach. The initial step involves data preprocessing, which converts the raw streaming data into a suitable format for subsequent processing. The next stage uses a drift detection technique known as DDM to identify the presence of concept drift. Finally, the ORF model is implemented to determine the class label of the streaming data.

The proposed approach consists of five main components: (1) data collection, (2) data preprocessing, (3) random forest based model training, (4) DDM-based drift detection and (5) online random forest-based incremental learning that updates the model in real-time.

4.1 Data collection

The Mawilab data set [14] is a publicly available data set developed by the Fukuda Laboratory at the University of Tokyo in Japan. The Mawilab data set is a collection of network traffic data that has been collected over several years from various sources, including honeypots, darknets and internet service providers. The data set contains both benign and malicious network traffic



Source: F. Jemili *et al.*

Figure 1. Proposed approach

data, making it a valuable resource for researchers studying network security and cyberattacks. The data set was initiated in 2004, and since then, it has been continuously updated with new network traffic data and improved analysis techniques.

In our contribution, as the Mawilab data set is collected from Web traffic data, we use a Web scrapper (BeautifulSoup) to extract specific data from the HTML files that represent the Web traffic. This data is then processed, transformed and stored in Azure in streaming.

Storing the Mawilab data set on Azure in streaming involves using Azure Stream Analytics, a cloud-based service for real-time data processing. This approach enables the data set to be stored in real-time, as the network traffic data is received from Mawilab.

To store the data set in streaming, we create an Azure Stream Analytics job in the Azure portal. This job acts as a pipeline that receives the network traffic data from Mawilab in real-time and stores it in Azure. The input for the Azure Stream Analytics job is configured as an Azure Event Hub, which is a scalable and real-time data streaming platform. The output is configured to store the data in various Azure storage solutions: Azure Blob Storage or Azure Data Lake Storage. Then, we define the queries for the Azure Stream Analytics job. The queries specify how the data received from Mawilab will be processed and transformed before being stored in Azure.

Once the Azure Stream Analytics job is configured, it begins to receive the network traffic data from Mawilab in real-time and stores it in Azure in streaming. By using this approach, we gain insights from the data in real-time and respond to potential security threats more quickly.

4.2 Data preprocessing

This step is about Mawilab data preprocessing using Spark on Azure HDInsight Spark cluster (Pamarthi and Narmadha, 2022). This step consists in (Abid *et al.*, 2022):

- *Data ingestion*: The Mawilab data set is ingested into the Spark cluster using Azure Blob Storage method. Then, data is converted from CSV into Apache Parquet format;
- *Data cleaning*: The Mawilab data set contains missing and invalid data, which needs to be cleaned before analysis. We use various functions Spark provides for data cleaning: drop() and fillna(). This step also involves eliminating redundancies in the data with dropDuplicates() function; and
- *Data transformation*: Once the data is cleaned, it undergoes several transformation steps to prepare it for machine learning algorithms. This includes feature engineering, feature selection and encoding categorical variables. We leverage Apache Spark's functionalities to efficiently handle large-scale data transformation.

4.2.1 Feature Engineering. Feature engineering involves creating new features from the existing ones to enhance the predictive power of the machine learning model. This step includes several subtasks:

VectorAssembler: Combines multiple columns into a single feature vector. This is useful for algorithms that expect a single vector input. The `VectorAssembler` function in Spark is used as follows:

```
from pyspark.ml.feature import VectorAssembler

 # Define the input columns to be assembled into a feature vectorin-
 put_columns = ['feature1', 'feature2', 'feature3']assembler =
 VectorAssembler(inputCols=input_columns, outputCol='features')
```

```
  # Transform the data set transformed_data = assembler.transform
  (cleaned_data)
```

This transformation combines specified input columns into a single feature vector column named ‘features’.

StringIndexer: Converts categorical variables into numerical indices. This is essential for machine learning algorithms that cannot handle categorical data directly. The `StringIndexer` function in Spark is used as follows:

```
from pyspark.ml.feature import StringIndexer

  # Define the column to be indexedindexer =

  StringIndexer(inputCol='category_column’,
  outputCol='category_index’)

  # Fit the indexer to the data and transform it

  indexed_data = indexer.fit(cleaned_data).transform(cleaned_data)
```

This transformation converts the ‘category_column’ into numerical indices, stored in ‘category_index’.

Polynomial Expansion: Generates polynomial features from the existing features, increasing the feature space to capture nonlinear relationships. In Spark, this can be done using the `PolynomialExpansion` function:

```
from pyspark.ml.feature import PolynomialExpansion

  # Define the polynomial expansion degreepoly_expansion =

  PolynomialExpansion(inputCol='features’, outputCol='poly_features’,
  degree = 2)

  # Transform the data set expanded_data = poly_expansion.transform
  (transformed_data)
```

This transformation creates polynomial features of degree 2 from the input feature vector.

Normalization: Scales the feature vectors to have unit norm. This ensures that all features contribute equally to the distance calculations in algorithms like k-NN or SVM. The `Normalizer` function in Spark is used as follows:

```
from pyspark.ml.feature import Normalizer

  # Define the normalizernormalizer = Normalizer(inputCol='features’,
  outputCol='norm_features’, p = 2.0)

  # Transform the data set normalized_data = normalizer.transform
  (expanded_data)
```

This transformation normalizes the feature vectors to have unit L^2 norm.

4.2.2 Feature Selection. Feature selection is the process of identifying the most relevant features for the machine learning model. This step helps in reducing the dimensionality of the data, improving model performance and avoiding overfitting. Several techniques can be used for feature selection:

Chi-Square Test: Evaluates the independence between each feature and the target variable, selecting features that are statistically significant. In Spark, the `ChiSqSelector` function is used as follows:

```
from pyspark.ml.feature import ChiSqSelector

  # Define the Chi-Square selector selector = ChiSqSelector
  (numTopFeatures = 10, featuresCol=‘features’,
  outputCol=‘selected_features’, labelCol=‘label’)

  # Fit the selector to the data and transform it selected_data = se-
  lector.fit(normalized_data).transform(normalized_data)
```

This selects the top 10 features that have the highest Chi-Square statistic with the target variable.

Correlation Matrix: Computes the correlation between features and the target variable, selecting features with high absolute correlation values. This can be done using Spark’s `Correlation` function:

```
from pyspark.ml.stat import Correlation

  # Compute the correlation matrixcorrelation_matrix = Correlation.
  corr(normalized_data, ‘features’).head()0

  # Select features based on a correlation thresholdselected_fea-
  tures = [i for i in range(len(correlation_matrix)) if abs(correla-
  tion_matrix[i]) > 0.1]
```

This selects features with an absolute correlation value greater than 0.1.

Feature Importance from Tree-Based Models: Uses the feature importance scores from tree-based models like Random Forests to select the most important features. This can be achieved using Spark’s `RandomForestClassifier`:

```
from pyspark.ml.classification import RandomForestClassifier

  # Train a Random Forest modelrf = RandomForestClassifier
  (featuresCol='features’, labelCol='label’)model = rf.fit
  (normalized_data)

  # Extract feature importance scoresfeature_importances = model.
  featureImportances

  # Select features based on importance scores selected_features =
  [i for i, importance in enumerate(feature_importances) if
  importance > 0.01]
```

This selects features with importance scores greater than 0.01.

By incorporating these steps into the data transformation process, we ensure that the data is adequately prepared for machine learning algorithms, enhancing the predictive power and performance of the models.

- Data storage: Once the data is preprocessed and transformed, it is stored in Parquet format in Azure Blob Storage for further analysis.

Mawilab data preprocessing using Spark on Azure HDInsight Spark cluster involves leveraging Spark's data processing and transformation capabilities, along with using Azure Blob Storage service for data storage and ingestion. In addition, using Parquet format for data storage helps improve query performance and reduce storage costs.

4.3 Random Forest based model training

Random Forest (RF) works by building an ensemble of decision trees using a bagging technique, where each tree is trained on a randomly sampled subset of the data and a randomly sampled subset of the features. Given a training set of N data points, the RF algorithm builds T decision trees, each of which predicts the class label of a new data point based on a majority vote of its leaf nodes (Thakkar and Lohiya, 2021):

$$f(x) = argmax_c \sum_{t=1}^{T} I(ht(x) = c) \tag{11}$$

where f(x) is the predicted class label for a new data point x, h_t(x) is the class label predicted by the tth decision tree and\mathbb{I}(\cdot) is the indicator function that returns 1 if its argument is true and 0 otherwise.

The algorithm (van Rijn *et al.*, 2018) works by creating a forest of decision trees, where each tree is built on a random subset of the original data. This process is called bagging or bootstrap aggregating, and it helps to reduce the variance of the model and prevent overfitting.

For classification tasks, the algorithm randomly selects a subset of features at each split point instead of using all the available features, which further helps to reduce the correlation between the trees and improve the accuracy of the model. During the prediction phase, the algorithm takes a new data point and passes it through each decision tree in the forest, and each tree produces a classification or regression output.

The final prediction is then made by aggregating the outputs of all the trees, either by majority voting in the case of classification or by taking the average in the case of regression.

The tree is created using a top-down approach (Zhou *et al.*, 2020) (see Figures 2 and 3), where the algorithm starts with the entire data set and repeatedly splits it into smaller subsets based on the value of a single feature that maximizes the information gain or minimizes the impurity. Information gain measures how much a particular feature contributes to reducing the uncertainty in the classification or regression task, while impurity measures the degree of disorder or randomness in the data.

In our contribution, a trained machine learning pipeline using Random Forest algorithm is loaded from Azure Blob Storage. Live data is classified using the pipeline, and predictions are then saved in Azure Data Lake.

4.4 Drift detection method based drift detection

Let y be the target variable we want to predict, and let f(x; θ) be the predictive model that maps the input variable x to the predicted target value y, where θ is the set of model parameters (Panigrahi and Borah, 2018).

```
Algorithm: Random Forest
#Inputs
T: number of trees
d_max: maximum tree depth
m: number of features
Training dataset: a set of (x, y) pairs
#Initialize an empty forest F with T trees
F ← {}
for i = 1 to T do
t_i ← create_new_tree(d_max, m)
F ← F ∪ t_i
#For each incoming training instance
for each tree t_i in F do
S_i ← sample_feature_subset(m)
n_i ← find_leaf_node(t_i, x)
update_leaf_statistics(n_i, y)
if depth(n_i) < d_max and enough_instances_to_split(n_i) then
(n_i_l, n_i_r) ← split_node(n_i, S_i)
add_child_nodes(n_i, n_i_l, n_i_r)
#To make a prediction
for each tree t_i in F do
n_i ← find_leaf_node(t_i, x_new)
compute_leaf_prediction(n_i)
```

Source: F. Jemili *et al.*

Figure 2. Random Forest algorithm

```
Algorithm: Decision Tree
#Define the tree node
S the set of samples
X the set of features
Y the set of labels
#Calculate the Gini impurity
Gini(S) = 1 - Σ p_i^2
where p_i is the proportion of samples in S with label i
For each feature x in X
#Calculate the Gini impurity of splitting S on x
Let S_l be the set of samples in S where x <= t
where t is a threshold value for feature x
Let S_r be the set of samples in S where x > t
Gini_l = Gini(S_l)
Gini_r = Gini(S_r)
Gini_split = (|S_l|/|S|) * Gini_l + (|S_r|/|S|) * Gini_r
#Calculate the information gain of splitting S on x
Info_gain = Gini(S) - Gini_split
#Keep track of the feature x with the highest gain
If the gain is less than a threshold value
Stop splitting and create a leaf node with label y
Otherwise,
Create an internal node with feature x and threshold value t
```

Source: F. Jemili *et al.*

Figure 3. Decision tree algorithm

At each time step *t*, we observe a new data point (xt, yt) and compute the prediction error εt as:

$$\varepsilon t = yt - f(xt; \theta) \tag{12}$$

We then update the mean error rate m and the standard deviation of the error rate s as follows:

$$m(t) = m(t-1) + (\varepsilon t - m(t-1))/(t+1) \tag{13}$$

$$s(t) = sqrt\left(s^2(t-1) + (\varepsilon t - m(t-1)) * (\varepsilon t - m(t))\right) \tag{14}$$

The drift measure d(t) at time *t* is defined as:

$$d(t) = |\varepsilon t – m(t)|/s(t) \tag{15}$$

If d(t) exceeds a predefined threshold *ω*, we declare a drift at time t.

The threshold *ω* can be computed based on the desired false positive rate (*α*) and false negative rate (*β*) as:

$$\omega = sqrt((1/2) * log(2/\alpha) * (1/\beta)) \tag{16}$$

The idea behind this threshold is to control the risk of false positives and false negatives while detecting changes in the data stream.

The different steps of the DDM-based Drift Detection algorithm are as follows (see Figure 4):

(1) Initialization:
- The algorithm initializes various parameters such as time *t*, evidence x, threshold a, drift rate v, nondecision time T, standard deviation of noise *σ* and window size w.

(2) Processing Loop:
- The loop runs until a decision is made.

(3) Increment Time:
- Time *t* is incremented by a small time step dt;

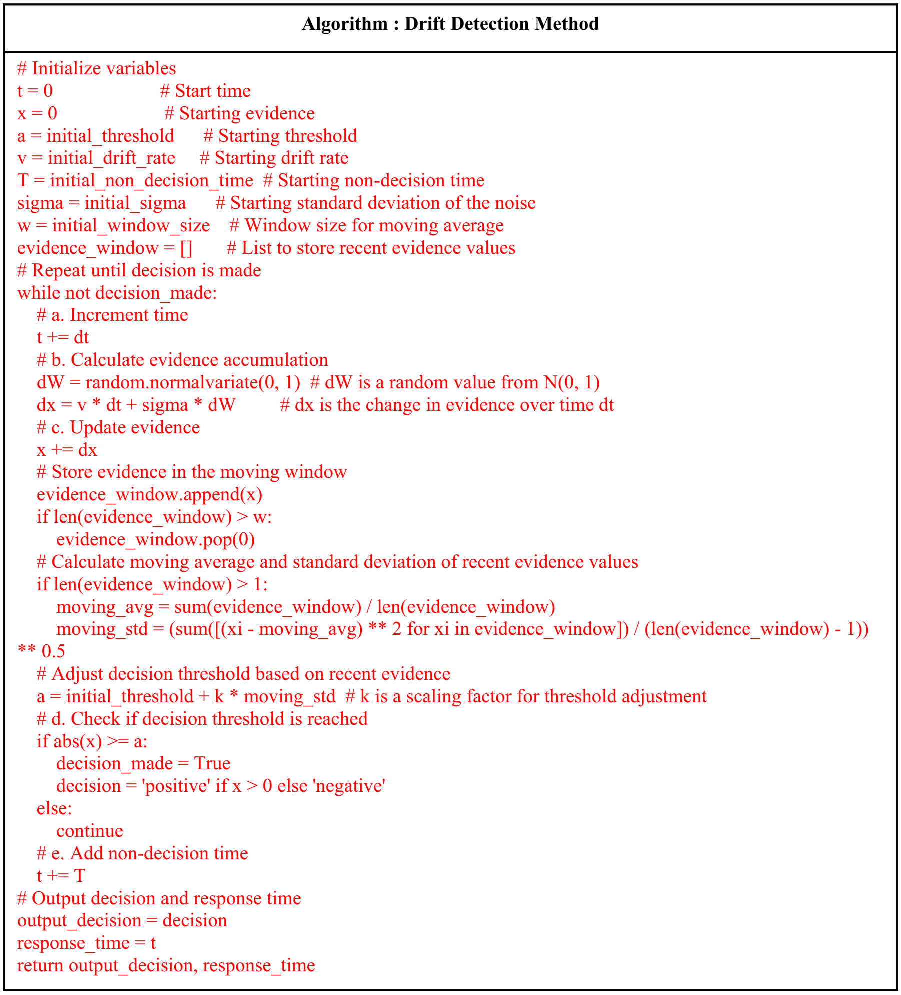
(4) Evidence Accumulation:
- The change in evidence dx is calculated using the drift rate, time step and a random value dW sampled from a normal distribution;

(5) Update Evidence:
- The evidence x is updated by adding dx;

(6) Moving Average and Standard Deviation:
- Evidence values are stored in a moving window. The moving average and standard deviation of recent evidence values are calculated to smooth out noise and maintain stability;

(7) Adjust Threshold:
- The decision threshold a is dynamically adjusted based on the moving standard deviation of recent evidence values;

(8) Threshold Check:



Algorithm : Drift Detection Method

```
# Initialize variables
t = 0                        # Start time
x = 0                        # Starting evidence
a = initial_threshold       # Starting threshold
v = initial_drift_rate      # Starting drift rate
T = initial_non_decision_time  # Starting non-decision time
sigma = initial_sigma       # Starting standard deviation of the noise
w = initial_window_size    # Window size for moving average
evidence_window = []       # List to store recent evidence values
# Repeat until decision is made
while not decision_made:
   # a. Increment time
   t += dt
   # b. Calculate evidence accumulation
   dW = random.normalvariate(0, 1)  # dW is a random value from N(0, 1)
   dx = v * dt + sigma * dW         # dx is the change in evidence over time dt
   # c. Update evidence
   x += dx
   # Store evidence in the moving window
   evidence_window.append(x)
   if len(evidence_window) > w:
      evidence_window.pop(0)
   # Calculate moving average and standard deviation of recent evidence values
   if len(evidence_window) > 1:
      moving_avg = sum(evidence_window) / len(evidence_window)
      moving_std = (sum([(xi - moving_avg) ** 2 for xi in evidence_window]) / (len(evidence_window) - 1))
** 0.5
   # Adjust decision threshold based on recent evidence
   a = initial_threshold + k * moving_std  # k is a scaling factor for threshold adjustment
   # d. Check if decision threshold is reached
   if abs(x) >= a:
      decision_made = True
      decision = 'positive' if x > 0 else 'negative'
   else:
      continue
   # e. Add non-decision time
   t += T
# Output decision and response time
output_decision = decision
response_time = t
return output_decision, response_time
```

Source: F. Jemili *et al.*

Figure 4. DDM Pseudo algorithm

- The algorithm checks if the absolute value of x exceeds the threshold a. If so, a decision is made based on the sign of x;

(9) Nondecision Time:

- Nondecision time T is added to the total time t; and

(10) Output:

- The algorithm outputs the decision and response time.

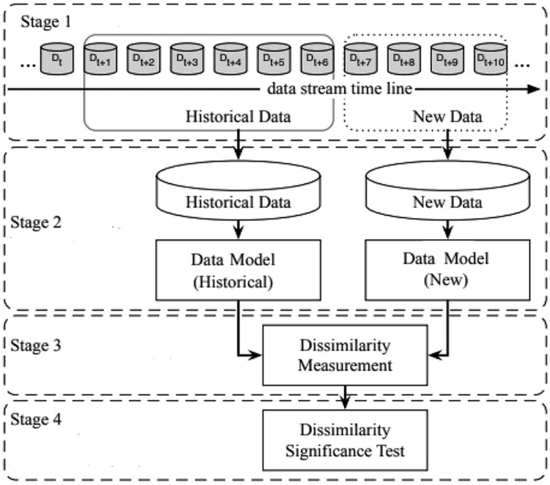
Figure 5 shows the flow chart of our DDM-based Drift Detection contribution. Mawilab data arrives over time in batches. D_t represents de t^{th} batch. Each batch contains a number of instances.

In this algorithm (Ivanov and Taaffe, 2018), the first stage involves a time window that is used to monitor the overall error rate of the system. Whenever a new data instance becomes available, the algorithm checks if there has been a significant increase in the error rate within the time window. If the observed error rate change exceeds a certain confidence level, the algorithm starts building a new learner while still using the old one for predictions. If the change exceeds a certain drift level, the old learner is replaced with the new one for all future prediction tasks. To determine the error rate, the algorithm relies on a classifier to make predictions, which is considered the second stage of the algorithm. The online error rate is then used in the third stage to calculate test statistics. Finally, in the fourth stage, a hypothesis test is conducted by estimating the distribution of the online error rate and calculating warning and drift thresholds.

4.5 Online random Forest incremental learning

Online Random Forest is an extension of Random Forest that allows the model to learn incrementally, i.e. it can update the model parameters with new data points without the need to retrain the entire model from scratch. Online Random Forest (ORF) is a type of incremental learning algorithm that can be used to handle concept drift in streaming data. ORF is based on the popular RF algorithm, but instead of building a static forest on a fixed data set, ORF incrementally updates the forest as new data arrives (Coccia *et al.*, 2021).

The ORF algorithm extends the RF algorithm to handle streaming data by incrementally updating the ensemble of decision trees as new data arrives. Specifically, ORF maintains a sliding window of size W that contains the most recent W data points and updates the ensemble after each new data point x_t arrives. The ORF algorithm consists of the following steps (Wang and Jones, 2017):



Source: F. Jemili *et al.*

Figure 5. DDM Based drift detection

- Add x_t to the sliding window and remove the oldest data point if the window size exceeds W;
- For each decision tree *t* in the ensemble, update the tree using the following procedure:
- Choose a random subset of features for the tree (this is called the "random subspace" method).
- Use the incremental learning algorithm to update the tree with x_t; and
- If the number of trees in the ensemble is less than T, add a new decision tree to the ensemble initialized with x_t.

The prediction of ORF on a new data point x is obtained by taking a majority vote of the predictions of each decision tree in the ensemble:

$$f(x) = argmax_c \sum_{t=1}^{T} I(ht(x) = c) \tag{17}$$

where h_t(x) is the class label predicted by the t^{th} decision tree for the new data point x. ORF uses an incremental learning algorithm to update each decision tree in the ensemble, which allows the algorithm to update the model efficiently as new data arrives. In addition, ORF uses a random subspace method to reduce the correlation between decision trees, which helps to improve the diversity of the ensemble and avoid overfitting.

The algorithm [28] starts by training an initial random forest model on a portion of the available data. As new data becomes available, it feeds it into the model one observation at a time. For each new observation, it determines which leaf node it falls into in each tree of the random forest. The algorithm updates the statistics associated with each leaf node for each tree. These statistics include the count of observations in the node, the sum of the response variable and the sum of the squared response variable.

The algorithm recalculates the prediction for each tree using the updated statistics for each leaf node and combines the predictions from each tree to obtain the final prediction for the new observation. The algorithm periodically reevaluates the performance of the model on a holdout set of data and considers pruning the trees that are not contributing significantly to the model's accuracy.

The algorithm Online Random Forest (Figure 6) starts by initializing a set of decision trees F.

Processing Incoming Data: For each incoming data point x_i, the algorithm processes it through each tree T_j in the forest F.

Finding the Leaf Node: For each tree T_j, the algorithm finds the leaf node l where the data point x_i belongs. This is typically done by traversing the tree from the root to a leaf node based on the features of x_i.

Updating the Leaf Node: If the number of data points in the leaf node l is less than a predefined threshold *n*, the data point x_i is simply added to l.

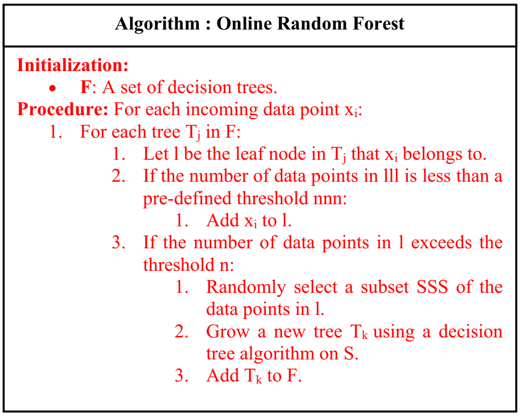
If the number of data points in l exceeds the threshold *n*, the algorithm proceeds to handle the overflow.

Handling Overflow: A subset S of the data points in l is randomly selected. This subset is used to grow a new decision tree, T_k.

The new tree T_k is grown using a standard decision tree algorithm applied to the subset S.

The new tree T_k is then added to the forest F, expanding the ensemble.

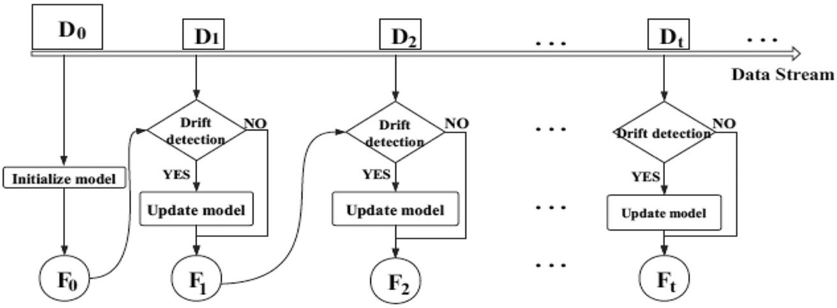
Figure 7 shows the flow chart of our online incremental learning contribution. The process initiates with the continuous monitoring of the data stream for signs of concept drift



```
Algorithm : Online Random Forest
Initialization:
    • F: A set of decision trees.
Procedure: For each incoming data point $x_i$:
    1. For each tree $T_j$ in F:
        1. Let l be the leaf node in $T_j$ that $x_i$ belongs to.
        2. If the number of data points in lll is less than a
           pre-defined threshold nnn:
            1. Add $x_i$ to l.
        3. If the number of data points in l exceeds the
           threshold n:
            1. Randomly select a subset SSS of the
               data points in l.
            2. Grow a new tree $T_k$ using a decision
               tree algorithm on S.
            3. Add $T_k$ to F.
```

Source: F. Jemili *et al.*

Figure 6. Online random Forest pseudo algorithm



Source: F. Jemili *et al.*

Figure 7. Incremental learning flow chart

using the DDM. This method operates by tracking the error rate of the classifier over time and comparing it to predefined threshold values. When the error rate exceeds these thresholds, it indicates a potential drift in the underlying data distribution.

Upon detecting a concept drift, the system triggers the online incremental learning mechanism to adapt to the new data patterns. The classifier update is executed using the Online Random Forest (ORF) algorithm, which is particularly well-suited for real-time data processing due to its ability to incrementally update the model without needing to retrain from scratch.

The ORF algorithm operates by adjusting the ensemble of decision trees in the random forest. When new data arrives, each tree in the forest is updated based on the new instances. Specifically, the algorithm uses techniques such as incremental learning of decision trees, where nodes are split or pruned based on the new data, and the weights of the trees are adjusted to reflect the recent changes in the data distribution.

Furthermore, the ORF algorithm integrates mechanisms to handle various types of concept drift, including abrupt, gradual and recurrent drifts. It does this by maintaining a sliding window of recent instances and periodically evaluating the performance of each tree within this window. Trees that consistently perform poorly are replaced or updated more aggressively, ensuring that the forest remains robust and adaptive to the evolving data landscape.

This approach ensures that the classifier remains accurate and efficient in the face of continuous data stream changes, providing timely and reliable intrusion detection. The integration with Apache Spark Structured Streaming further enhances the system's capability to process large-scale data in a distributed manner, making it suitable for real-world applications with high throughput and low latency requirements.

5. Experimentation and discussion

To evaluate the detection capabilities of the proposed DDM-ORF model, a series of experiments were conducted using the Mawilab data set and a variety of evaluation metrics: precision, recall, accuracy and F-measure.

The MAWILab data set is comprised of labeled traffic flows indicating whether they are anomalous or not. This data set uses four different anomaly detection methods (Hough transform, Gamma distribution, Kullback–Leibler divergence and Principal Component Analysis "PCA") (Salah *et al.*, 2022). The traffic flows in the data set are classified into four main categories as follows:

(1) *Anomalous*: Traffic flows that are deemed abnormal and are detected by the used anomaly detectors;

(2) *Suspicious*: Traffic flows that are likely to be anomalous but are not clearly detected by the anomaly detectors;

(3) *Notice*: Traffic flows that are normal but have been reported by one or more of the anomaly detectors; and

(4) *Benign*: Normal traffic flows that have not been reported or detected by any of the anomaly detectors.

In their work (Singh and Ranga, 2021), introduced a taxonomy that outlines the characteristics of backbone traffic anomalies. The MAWILab data set leverages this taxonomy by including a dedicated field to provide further insights into the nature of anomalies. The Table 4 presented below illustrates the various taxonomies used in the data set.

We conduct data preprocessing using a Microsoft HDInsight cluster running Apache Spark Structured Streaming. Structured Streaming is built on top of Spark SQL engine, which gives us exactly once delivery and provides end-to-end reliability.

When deploying an HDInsight cluster, Azure uses the Hortonworks Data Platform (HDP), which is powered by Apache Hadoop. HDP is a massively scalable and open source; it is used for storing, processing and analyzing big data. It is designed to handle multiple data sources and formats with a user-friendly dashboard. HDP consists of a set of Hadoop projects, including Storm, Spark and Ambari. Once the cluster is deployed, HDInsight provides multiple options for the user to choose from. In addition, Azure provides multiple visualization tabs to track and monitor the cluster performance for processing, storage and bandwidth.

When we transform CSV files into Parquet format, we observe benefits in both cost and performance. By using Parquet, we not only reduce the time spent waiting for data to be scanned and processed, but also lower storage expenses (van Rijn *et al.*, 2018). The MAWILab data set is updated frequently, with new files being added to their website on a regular basis. Our

Table 4. Description of taxonomies

Taxonomy	Description
“Unk”, “empty”	Unknown labels
“ttl_error”,”hostout”,”netout”, “icmp_error”	Other labels
“alphflHTTP”,”ptmpHTTP”,”mptpHTTP”,”ptmplaHTTP”,”mptplaHTTP”	HTTP
“ptmp”,”mptp”, “mptmp”	Multi points
“alphfl”,”malphfl”,”salphfl”,”point_to_point”, “heavy_hitter”	Alpha flow
“ipv4gretun”, “ipv46tun”	IPv6 tunneling
“posca”, “ptpposca”	Port scan
“ntscIC”, “dntscIC”	Network scan ICMP
“ntscUDP”, “ptpposcaUDP”	Network scan UDP
“ntscACK”,”ntscSYN”,”sntscSYN”,”ntscTCP”,”ntscnull”,”ntscXmas”,”nts cFIN”, “dntscSYN”	Network scan TCP
“DoS”,”distributed_dos”,”ptpDoS”,”sptpDoS”,”DDoS”, “rflat”	DoS

Source: F. Jemili *et al.*

Web scraper promptly ingests any new files that become available. To estimate the storage gain for each new file, we use an average file size since the file sizes can vary. Table 5 displays the size differences between the old and new formats after conversion to Apache Parquet.

Structured Streaming is a stream processing model introduced in Apache Spark version 2.0. It is scalable and fault-tolerant, and it uses Data Frame API to simplify the development of real-time Big Data applications. The key concept of Structured Streaming is to treat an incoming stream of live data as a table that is being appended by new rows. For Structured Streaming, the idea is to append data to an unbounded input table. Users can specify how often these tables are appended by using triggers. If a trigger is set to one second, then Structured Streaming collects streaming data for one second and then appends all of it to the input table. And this process reruns depending on a trigger interval set by the user. Afterwards, a query is run on the data, and the result of that query is saved to the result table or output table. The result table is written every time to an output sink that is specified by the user and can be a database, storage space or another streaming job.

With Structured Streaming, it is possible to select relevant columns from a data set and ignore the rest of unselected columns. Aggregation and filter operations can be applied to data too. This includes filtering data sets based on one or many columns; also, aggregation on data can be applied to extract only relevant fields. Spark’s Structured Streaming API offers a solution to eliminate duplicate rows from a continuous stream of data by utilizing a unique identifier column. By keeping and storing data from previous records, Structured Streaming is able to filter out duplicate records.

We begin by reading Parquet files as a stream, and then selecting specific features that aid in the detection process. To reduce computational burden, unnecessary features such as

Table 5. Format conversion

Average size (CSV)	Average size (parquet)	Speedup
9.5 KB	6.5 KB	X1.46

Source: F. Jemili *et al.*

anomaly_id and label (empty columns) are eliminated. In addition, instead of deleting records that contain missing fields, we fill them with default values to create a complete data set, and we eliminate any duplicate records.

After data preparation, we proceed to load the RF classifier, the DDM detector and the ORF classifier and transform incoming data to obtain predictions for each record. A pipeline is a sequence of stages where each stage is either a transformer or an estimator, and these stages are executed in a specific order. In our work, we used two transformers: String indexer and vector assembler. The string indexer encodes a string column of labels into a column of label indices, which are ordered by label frequencies, with the most frequent label receiving index 0. The vector assembler is a transformer that combines a specified list of columns into a single vector column. This transformer is helpful in consolidating raw features and features produced by different feature transformers into a single feature vector. We then use a portion of the historic data obtained from the Mawilab data set to train the pipeline model through the aforementioned stages.

The pipeline model is saved in Azure Blob Storage. The process of creating a pipeline model is done offline. Once a concept drift is detected, the RF model is automatically updated using ORF incremental learning to train on new data, and the updated model is exported to replace the old model. Predictions are then written to a specified output location or sink, with the format of the output data (such as Parquet and JSON) being specified, along with a checkpoint location to ensure fault-tolerance.

To conduct our experiments, we use Microsoft Azure HDInsight, which runs on Linux virtual machines and Spark version 2.0 on top of YARN, using Jupyter Notebook and the Python API (Pyspark). During the streaming job, Spark ingests data, classifies it, detects concept drift, updates the model and writes each file to the designated sink. The sink may be a file directory, a database or even another Spark job.

5.1 Experiment 1

In this experiment, we evaluate the proposed DDM-ORF approach in terms of classification performance. Apache Spark offers a collection of metrics that can be used to assess the performance of machine learning models. In our work, we used the following metrics to evaluate the performance of our pipeline:

- Accuracy: Measures precision across all labels:

$$AC = \frac{TP + TN}{TP + FN + TN + FP} \tag{18}$$

- Precision: Measures the proportion of correct classified labels over all labels:

$$P = \frac{TP}{TP + FP} \tag{19}$$

- Recall: Measures the proportion of correct labels correctly classified over all positive labels:

$$R = \frac{TP}{TP + FN} \tag{20}$$

- F-Measure: Measures the average of precision and recall:

$$FM = 2 * \frac{P * R}{P + R} \tag{21}$$

where TP = true positives, TN = true negatives, FP = false positives and FN = false negatives.

The results generated by the pipeline can be seen in Table 6 below:

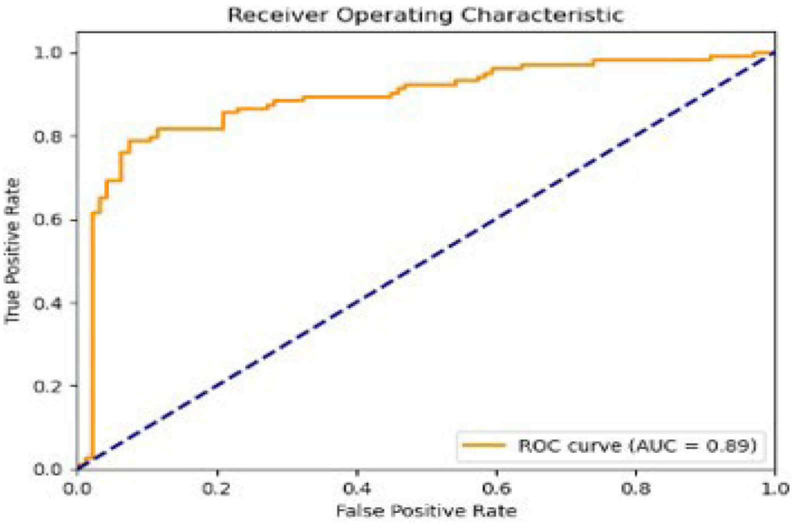
The Receiver Operating Characteristic curve (ROC curve) (see Figure 8) is a technique that can be used to assess an individual's capabilities in distinguishing between groups. Derived from the ROC curve, the numerical measure used to estimate the curve is the Area Under the ROC Curve (AUC). This measure can be interpreted as the probability that the model makes correct predictions. The AUC serves as an effective means of summarizing the overall diagnostic accuracy of a test. Typically:

- If AUC = 0.5, the diagnostic result is normal;
- If 0.5 < AUC < 0.7, the result is not very significant; and
- If 0.7 < AUC < 0.8, the result is considered acceptable.
- Results are deemed excellent if 0.8 < AUC < 0.9.

Table 6. DDM-ORF results

Accuracy	Precision	Recall	F-measure
99.96%	99.93%	99.95%	99.94%

Source: F. Jemili *et al*.



Source: F. Jemili *et al.*

Figure 8. DDM-ORF ROC Curve

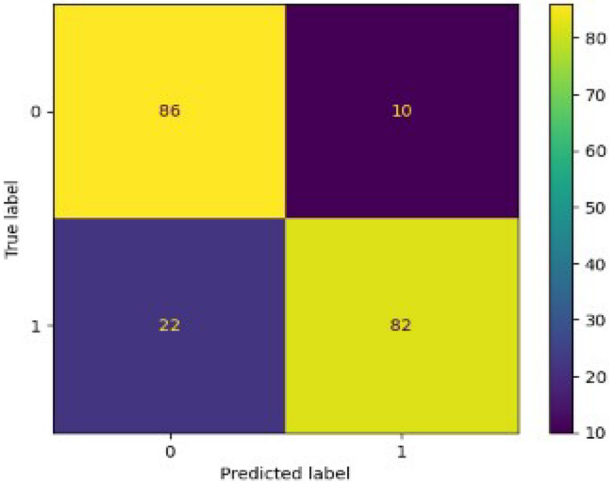
The confusion matrix (Salah *et al.*, 2022) (see Figure 9), can be regarded as a tool with the capability to analyze whether a classifier can successfully recognize tuples from different classes. True positive and true negative values provide insights into the true nature of the classifier when classifying data. On the other hand, false positive and false negative values provide information when the classifier is incorrect in classifying data.

To evaluate the effectiveness of the proposed DDM-ORF approach, we compare it against several state-of-the-art methods: ensemble incremental learning (Yuan *et al.*, 2018), adaptive class incremental learning (Liu *et al.*, 2023) and active incremental learning (Sun *et al.*, 2022). These methods were chosen based on their relevance and reported performance in handling concept drift and online incremental learning. The section below also includes a comparison between our DDM-ORF proposed approach and the approach proposed in Dhahbi and Jemili (2021). Authors in Dhahbi and Jemili (2021) proposed an intrusion detection system that uses deep learning techniques to analyze network traffic and detect anomalous and suspicious traffic. The authors used a deep neural network architecture called a convolutional neural network (CNN) to analyze features and classify network traffic. The CNN consisted of several layers, including convolutional layers, pooling layers and fully connected layers. The authors also used dropout regularization to prevent overfitting. They trained the CNN on the training set and tuned the hyperparameters using the validation set. They then evaluated the performance of the system on the testing set.

Table 7 and Figure 10 present a comprehensive comparison of the proposed DDM-ORF method against the state-of-the-art methods, including the CNN-based model proposed in (Dhahbi and Jemili, 2021). Metrics such as accuracy, precision, recall, F1-score and processing time were used to evaluate the performance.

The proposed DDM-ORF method outperformed the other methods in terms of accuracy, precision, recall and F1-score. In addition, the processing time of DDM-ORF was significantly lower, highlighting its efficiency in real-time applications.

- Proposed DDM-ORF



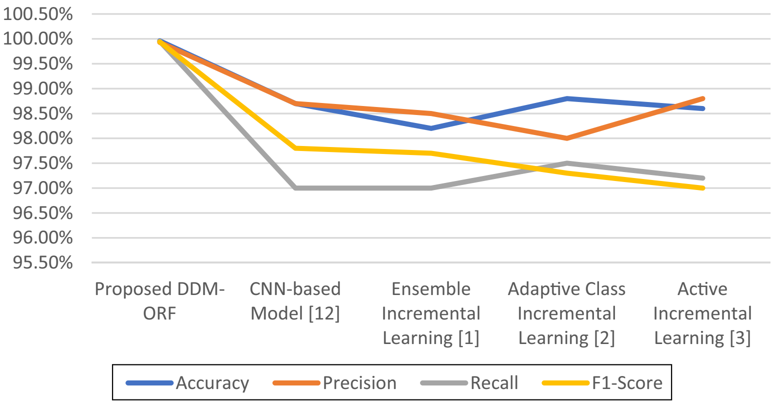
Source: F. Jemili *et al.*

Figure 9. Confusion matrix

Table 7. Comparison results

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Processing time (s)
Proposed DDM-ORF	99.96	99.93	99.95	99.94	0.02
CNN-based model (Dhahbi and Jemili, 2021)	98.7	98.7	97.0	97.8	0.06
Ensemble incremental learning (Yuan *et al.*, 2018)	98.2	98.5	97.0	97.7	0.03
Adaptive class incremental Learning (Liu *et al.*, 2023)	98.8	98.0	97.5	97.3	0.04
Active incremental Learning (Sun *et al.*, 2022)	98.6	98.8	97.2	97.0	0.05

Source: F. Jemili *et al*.



Source: F. Jemili *et al.*

Figure 10. Comparison results

High accuracy, precision, recall and F1-score: The DDM-ORF's high performance metrics suggest that it is highly effective at detecting and adapting to concept drifts in real-time. This can be attributed to the dynamic updating of the random forest classifier and the efficient handling of new data instances, which allows the model to quickly adapt to changes in the data distribution.

Low processing time: The efficient incremental update mechanism of the ORF algorithm, combined with the quick detection of drifts using DDM, contributes to the low processing time. This efficiency makes it suitable for real-time applications where quick response is critical.

- CNN-based Model (Dhahbi and Jemili, 2021)

High accuracy and precision but lower recall and F1-score: While the CNN-based model performs well in terms of accuracy and precision, its recall and F1-score are slightly lower, indicating that it may not be as effective in identifying all instances of anomalies or suspicious traffic. This could be due to the static nature of the CNN architecture, which might not adapt as quickly to changes in data patterns as the DDM-ORF.

Higher processing time: The deeper architecture of the CNN, with multiple convolutional, pooling and fully connected layers, along with dropout regularization, increases the computational complexity and processing time compared to the DDM-ORF.

- Ensemble Incremental Learning (Yuan *et al.*, 2018)

Good performance but slightly lower than DDM-ORF: The ensemble approach generally offers robustness by combining multiple models, but it might not adapt as swiftly to concept drifts as the DDM-ORF, resulting in slightly lower performance metrics.

Moderate processing time: The need to update multiple classifiers in the ensemble leads to a moderate processing time, higher than DDM-ORF but lower than the CNN-based model.

- Adaptive Class Incremental Learning (Liu *et al.*, 2023)

High accuracy with some trade-offs: This method shows good accuracy and recall but slightly lower precision and F1-score, indicating some issues with false positives. Its design for IoT environments might contribute to its solid performance, though it may struggle with imbalanced data sets or require substantial labeled data.

Moderate processing time: The approach involves incremental updates that, while efficient, do not match the low processing time of DDM-ORF due to the additional steps required for handling class increments and updating the neural network structure.

- Active Incremental Learning (Sun *et al.*, 2022):

Good but not top performance: While maintaining good accuracy and precision, the reliance on active learning and human expertise might limit its scalability and adaptability, resulting in slightly lower metrics compared to the DDM-ORF.

Higher processing time: The interactive nature of active learning, which involves periodic human intervention, contributes to a higher processing time, making it less suitable for real-time applications compared to the fully automated DDM-ORF.

These results demonstrate the effectiveness of DDM-ORF in maintaining high accuracy and efficiency while handling concept drift and online incremental learning, making it a robust choice for real-time IDS.

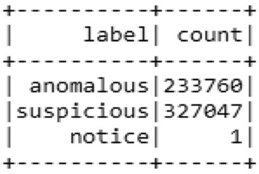
5.2 Experiment 2

In this evaluation, we analyze the DDM-ORF model predictions to determine and find some insights. Our objective is to identify the top IP addresses responsible for launching attacks and the top IP addresses that were targeted by attacks. In addition, we aim to determine the ports used for launching attacks and the most frequently attacked ports.

As indicated in Table 4, the MAWILab data set features a “taxonomy” field that offers a comprehensive classification of each recorded event. This taxonomy can be segregated into two primary categories, anomalous and suspicious.

Our evaluation data set comprises more than 1,873,545 events. Following the execution of our streaming job and the deduplication operation via Spark Structured Streaming to eliminate duplicate records, we were left with a total of 560,808 events. The subsequent Figure 11 illustrates how these events are distributed.

According to our analysis, anomalous events account for more than 41.68% of the evaluated data set, while suspicious events account for 58.31%. The top 10 taxonomies found in the data set, as shown in Table 8, account for 92.15% of the total taxonomies found, with a total of 516821. Other types of attacks, such as DoS and DDoS, account for only 1.74% of the anomalous traffic. Table 9 displays the top 5 taxonomies found in both anomalous and suspicious traffic. The top 5 anomalous taxonomies, which make up 81.97%



```
+----------+------+
|     label| count|
+----------+------+
| anomalous|233760|
|suspicious|327047|
|    notice|     1|
+----------+------+
```

Source: F. Jemili *et al.*

Figure 11. Labels distribution

Table 8. Top 10 taxonomies

Taxonomy	Count	Taxonomy	Count
Multi. points	157,018	ntscUDPUDPrp (network scan UDP)	24,812
HTTP	151,866	ptmpla (HTTP)	22,935
Alpha flow	52,314	mptpla (HTTP)	21,655
ntscSYNt (network scan TCP)	31,996	network scan TCP	19,337
Network scan UDP	28,981	ntscSYNt139445 (Network scan TCP)	5,907

Source: F. Jemili *et al*.

Table 9. Top five anomalous and suspicious taxonomies

Taxonomy (anomalous)	Count	Taxonomy (suspicious)	Count
Multi. points	88,429	HTTP	84,803
Http	67,063	Multi. points	68,589
ntscSYNt	14,773	Alpha flow	39,887
Alpha flow	12,427	Network scan UDP	21,205
Network scan TCP	8,508	ntscUDPUDPrp (network scan UDP)	19,442

Source: F. Jemili *et al*.

of all anomalous events, and the top 5 suspicious taxonomies, with a total of 233926 events, represent 71.52% of all suspicious events.

Table 10 shows the most targeted destination ports and the most used source ports for both anomalous and suspicious events. We notice that in the analyzed data set, the most frequently used ports for both anomalous and suspicious traffic were unknown ports, with ports 80 (HTTP), 53 (DNS) and 443 (HTTPS) following closely behind. Even though IP addresses are not always a true source of traffic since they can be masked or changed, they are still used to identify sources of attacks or at least the country of the original attack.

We list in Table 11 the top IP addresses that were the source of anomalous and suspicious activities as well as the top IP addresses that were the destination of such activities. Most anomalous/suspicious events were originated from unknown sources followed by 0.0.0.0 address, which are also from unknown source.

Our DDM-ORF proposed approach provides multi-class classification (suspicious and anomalous). This enables the detection system to not only detect whether an intrusion

Table 10. Anomalous and suspicious most targeted and used ports

	Destination ports	Count	Source ports	Count
Anomalous	0 (unknown)	123,093	0 (unknown)	100,656
	80	33,109	80	65,368
	443	11,468	443	18,569
	53	11,393	53	9,531
	22	3,809	6,000	5,906
Suspicious	0 (unknown)	138,370	0 (Unknown)	176,086
	80	48,909	80	55,186
	53	25,811	53	16,228
	443	18,217	443	15,271
	23	12,537	22	2,080

Source: F. Jemili *et al*.

Table 11. Anomalous and suspicious most emitting and receiving IP addresses

	Destination IP	Count	Source IP	Count
Anomalous	Unknown	87,198	Unknown	83,334
	0.0.0.0	232	0.0.0.0	409
	193.42.178.137	22	172.20.32.73	42
	193.42.178.130	21	172.20.32.48	40
	10.5.115.115	20	94.164.147.39	15
Suspicious	Unknown	112,614	Unknown	91,559
	0.0.0.0	1,766	0.0.0.0	1,278
	10.64.17.12	88	29.15.106.164	55
	10.5.115.115	61	172.20.32.48	54
	10.64.169.9	37	138.131.183.14	50

Source: F. Jemili *et al*.

attempt is occurring but also to provide insights into the type of attack being attempted. By categorizing intrusion attempts, the system can provide insights into the frequency and type of attacks being attempted, allowing security professionals to better understand and respond to threats.

5.3 Experiment 3

This experiment concerns Apache Spark evaluation. Authors in Ivanov and Taaffe (2018) presented significant metrics for assessing the performance of Apache Spark streaming. As our work involves the ongoing collection of data from the Fukuda Lab, we rely on two metrics to measure performance: processed records per second and input rows per second. These metrics are described in the following Table 12 along with the corresponding results.

Our system collected more than 560,808 rows of data, with each file containing an average of 163 rows. The system was able to process these records at a rate of 55,175 records per second and collected all files in a single batch. This high processing rate was made possible by setting the maxFilesPerTrigger parameter to its maximum value, which allowed

Table 12. Apache spark evaluation

Metric	Description	Results
Input rate	Describes how many rows were loaded per second	560,808 Event/second
Processing rate	Describes how many rows were processed per second	55,175 Event/second

Source: F. Jemili *et al*.

for the collection of all files at once rather than limiting the number of files collected at a time.

The performance of our Spark Cluster is influenced by two key factors. The first factor is the size of the cluster, which can affect the system's ability to process data depending on the workload. Allocating more processing power to the cluster generally results in the system being able to process more data. The second factor is the rate at which data is incoming. If the rate exceeds the system's processing capacity, it can cause a bottleneck and require intervention to restrict the input rate.

In general, our proposed DDM-ORF system achieves excellent classification results and processing speed, which are suitable for real-world scenarios and can be further improved by adding more machines to the cluster. In addition, Apache Spark Structured Streaming provides crucial features for handling streaming data, such as dynamic modification of data structure, filling missing fields and removing duplicate records. Moreover, Apache Spark is fault-tolerant, distributed and capable of reading and writing data from various sources in various formats.

In comparison with similar contributions described in the related work section, our DDM-ORF proposed contribution is tested on a large-scale data set suitable for heterogenous data, based on online learning approach, and provides multi-class classification with very good accuracy.

6. Conclusion

This paper proposes a novel DDM-ORF model to detect intrusions based on concept drift detection and online incremental learning. The model uses Drif Detection Method for drift detection and the Online Random Forest algorithm for the incremental learning. Incremental learning algorithms are designed to continuously update the model parameters as new data becomes available and can adapt to changes in the data distribution over time. This makes them well-suited for applications where the data is dynamic and nonstationary. However, it is worth noting that incremental learning is more computationally complex compared to traditional machine learning, and requires additional resources for training and monitoring the model. In addition, incremental learning algorithms require more data to achieve better accuracy levels than traditional machine learning approaches, as they need to continuously update the model parameters to account for changes in the data distribution.

In our contribution, we prioritize the key features of scalability, availability and fault-tolerance by choosing Apache Spark as our distributed computing framework. This enables us to process large volumes of data simultaneously across multiple machines. To further strengthen our system, we take advantage of Microsoft Azure Cloud's auto-scaling capabilities. This enables us to dynamically adjust the size of our cluster to accommodate changes in workload without any downtime or interruption of service. In addition, Microsoft Azure Cloud ensures high availability of our cluster by providing redundancy and failover mechanisms that prevent data loss and maintain system accessibility even in the event of hardware failures or network outages.

Our proposed contribution, DDM-ORF, uses an online learning approach and has been tested on a large-scale data set. It is suitable for heterogenous data set and provides multi-class classification with high accuracy.

From another side, DDM only considers changes in the frequency of class labels and may not be effective in detecting more complex types of concept drifts. One perspective to address these limitations is to explore other concept drift detection algorithms, such as those based on clustering, density estimation or change-point detection. Besides, one limitation of online incremental random forest is that it requires more memory to maintain a large number of trees as new data arrives. It can suffer from overfitting if the size of the forest is not carefully controlled. One perspective to overcome this limitation is to use ensemble pruning techniques that can dynamically remove unnecessary trees while retaining the accuracy of the model. Another perspective is to explore new algorithms that can effectively balance the trade-off between accuracy and memory usage, such as compressed or compact random forests.

References

Abdel Wahab, O. (2022), "Intrusion detection in the IoT under data and concept drifts: online deep learning approach", *IEEE Internet of Things Journal*, Vol. 9 No. 20, pp. 19706-19716, doi: 10.1109/JIOT.2022.3167005.

Abid, F., Jemili and Korbaa, O. (2022), "Distributed architecture of an intrusion detection system in industrial control systems", *ICCCI 2022 14th International Conference on Computational Collective Intelligence*.

Abid and Jemili, F. (2020), "Intrusion detection based on graph oriented big data analytics", *KES-2020 24th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 448-457, doi: 10.1016/j.procs.2020.08.059.

Coccia, M., Roshani, S. and Mosleh, M. (2021), "Scientific developments and new technological trajectories in sensor research", *Sensors*, Vol. 21 No. 23, p. 7803, doi: 10.3390/s21237803.

D'Angelo, G., Palmieri, F. and Robustelli, A. (2021),"Effectiveness of video-classification in android malware detection through API-Streams and CNN-LSTM autoencoders", *5th International Symposium on Mobile Internet Security (MobiSec)*, pp. 171-194.

Dhahbi, R. and Jemili, F. (2021), "A deep learning approach for intrusion detection", *2021 IEEE 23rd International Conference on High Performance Computing and Communications (HPCC)*, pp. 1-8, doi: 10.1109/HPCC-SmartCity-DSS51687.2021.00033.

Dwibedi, S., Pujari, M. and Sun, W. (2020), "A comparative study on contemporary intrusion detection datasets for machine learning research", *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*

Folino, G., Pisani, F.S. and Pontieri, L. (2020), "A GP-based ensemble classification framework for time-changing streams of intrusion detection data", *Soft Computing*, Vol. 24 No. 23.

Guarino, G., Bovenzi, D., Di Monda, G., Aceto, D., Ciuonzo and Pescapé, A. (2022), "On the use of machine learning approaches for the early classification in network intrusion detection", *2022 IEEE International Symposium on Measurements and Networking (M&N)*.

Hafsa, M. and Jemili, F. (2018), "Comparative study between big data analysis techniques in intrusion detection", *Big Data and Cognitive Computing*, Vol. 3 No. 1, pp. 1-12, doi: 10.3390/bdcc3010001.

Ivanov, T. and Taaffe, J. (2018), "Exploratory analysis of spark structured streaming", *International Conference on Performance Engineering, Berlin*.

Jemili, F. (2022), “Intelligent intrusion detection based on fuzzy big data classification”, *Cluster Computing*, Vol. 26 No. 6, doi: 10.1007/s10586-022-03769-y.

Kuppa, N.-ALe-Khac. (2022), “Learn to adapt: robust drift detection in security domain”, *Computers and Electrical Engineering*, Vol. 102, p. 108239, doi: 10.1016/j.compeleceng.2022.108239.

Liu, Q., Zhang, Y., Zhou, W., Jiang, X., Zhou, W. and Zhou, M. (2023), “Adaptive class incremental learning-based IoT intrusion detection system”, *Computer Engineering*, Vol. 49 No. 2, pp. 169-174.

Mahdavi, E., Fanian, A., Mirzaei, A. and Taghiyarrenani, Z. (2022), “ITL-IDS: incremental transfer learning for intrusion detection systems”, *Knowledge-Based Systems*, Vol. 253, p. 109542, doi: 10.1016/j.knosys.2022.109542.

Meddeb, R., Jemili, F., Triki, B. and Korbaa, O. (2023), “A deep learning based intrusion detection approach for mobile ad-hoc network”, *Soft Computing*, Vol. 27 No. 14, doi: 10.1007/s00500-023-08324-4.

Nugroho, E., Djatna, T., Sitanggang, I.S., Buono, A. and Hermadi, I. (2020), “A review of intrusion detection system in IoT with machine learning approach: current and future research”, *6th International Conference on Science in Information Technology (ICSITech)*.

Pamarthi, S. and Narmadha, R. (2022), “Literature review on network security in wireless mobile Ad-hoc network for IoT applications: network attacks and detection mechanisms”, *International Journal of Intelligent Unmanned Systems*, Vol. 10 No. 4, pp. 482-506.

Panigrahi, R. and Borah, S. (2018), “A detailed analysis of CICIDS2017 dataset for designing intrusion detection systems”, *International Journal of Engineering and Technology*, Vol. 7 No. 3.24, pp. 479-482.

Salah, I., Jouini, K. and Korbaa, O. (2022), “Augmentation-based ensemble learning for stance and fake news detection”, *in Advances in Computational Collective Intelligence – 14th International Conference, ICCCI 2022, Hammamet, Tunisia, September 28-30, 2022, Proceedings, vol. 1653 of Communications in Computer and Information Science, Springer*, pp. 29-41.

Singh, P. and Ranga, V. (2021), “Attack and intrusion detection in cloud computing using an ensemble learning approach”, *International Journal of Information Technology*, Vol. 13 No. 2, pp. 565-571.

Sun, Z., Ran, G. and Jin, Z. (2022), “Intrusion detection method based on active incremental learning in industrial internet of things environment”, *Journal on Internet of Things*, Vol. 4 No. 2, pp. 99-111.

Thakkar, RLohiya. (2021), “A review on machine learning and deep learning perspectives of IDS for IoT: recent updates, security issues, and challenges”, *Archives of Computational Methods in Engineering*, Vol. 28 No. 4, pp. 3211-3243.

van Rijn, J.N., Holmes, G., Pfahringer, B. and Vanschoren, J. (2018), “The online performance estimation framework: heterogeneous ensemble learning for data streams”, *Machine Learning*, Vol. 107 No. 1, pp. 149-176.

Wang, RJones. (2017), “Big data analytics for network intrusion detection: a survey”, *International Journal of Networks and Communications*, Vol. 7 No. 1, pp. 24-31.

Wu, Z., Gao, P., Cui, L. and Chen, J. (2022), “An incremental learning method based on dynamic ensemble RVM for intrusion detection”, *IEEE Transactions on Network and Service Management*, Vol. 19 No. 1, pp. 671-685.

Yuan, X., Wang, R., Zhuang, Y., Zhu, K. and Hao, J. (2018), “A concept drift based ensemble incremental learning approach for intrusion detection”, *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, pp. 350-357, doi: 10.1109/Cybermatics_2018.2018.00087.

Zhou, Y., Cheng, G., Jiang, S. and Dai, M. (2020), “Building an efficient intrusion detection system based on feature selection and ensemble classifier”, *Computer Networks*, Vol. 174, p. 107247.

Further reading

Coccia, M., Roshani, S. and Mosleh, M. (2022), "Evolution of sensor research for clarifying the dynamics and properties of future directions", *Sensors*, Vol. 22 No. 23, p. 9419, doi: 10.3390/s22239419.

Jemili, F. (2023), "Towards data fusion-based big data analytics for intrusion detection", *Journal of Information and Telecommunication*, doi: 10.1080/24751839.2023.2214976.

Kamel, Y., Jemili, F. and Meddeb, R. (2022),"Ensemble learning based big data classification for intrusion detection", *22nd International Conference on Intelligent Systems Design and Applications*, Springer, pp. 1-8.

Karthika, S., Loganathan and M., Vanathi. (2024), "A hybrid machine learning based feature selection technique for attack detection in NIDS".

Salih, A. and Abdulazeez, A.M. (2021), "Evaluation of classification algorithms for intrusion detection system: a review", *Journal of Soft Computing and Data Mining*, Vol. 2 No. 1, pp. 31-40.

Shaukat, S., Luo, V., Varadharajan, I.A., Hameed, S., Chen, D., Liu and J., Li. (2020), "Performance comparison and current challenges of using machine learning techniques in cybersecurity", *Energies*, Vol. 13 No. 10, p. 2509.

Sultan, Z. and İskefiyeli, M. (2020), "Anomaly-based intrusion detection from network flow features using variational autoencoder", *IEEE Access*, Vol. 8, pp. 108346-108358.

Tama, A., Comuzzi, M. and Rhee, K.-H. (2019), "Tse-ids: a two-stage classifier ensemble for intelligent anomaly-based intrusion detection system", *IEEE Access*, Vol. 7, pp. 94497-94507.

About the authors

Farah Jemili holds an Engineer degree (2002), an MSc degree (2004) and a PhD degree (2010) in computer science. She is currently an Assistant Professor at Higher Institute of Computer Science and Telecom of Hammam Sousse (ISITCom-University of Sousse) and a Senior Researcher at MARS Research Laboratory (ISITCom-University of Sousse). She has extensive experience as a researcher in artificial intelligence, big data analysis and distributed systems, with special focus on intrusion detection systems. She has around 40 published research papers and served as reviewer for many international conferences and journals. Farah Jemili is the corresponding author and can be contacted at: jmili_farah@yahoo.fr

Khaled Jouini received the PhD degree in Computer Science from Paris-Dauphine University (France) in 2008. He was a research staff member at Telecom ParisTech (France). Since 2011, he has been with Sousse University (Tunisia), where he is currently an Assistant Professor. His research interests include Data engineering, machine learning, NLP and large-scale data management and mining.

Ouajdi Korbaa is a full-time professor at the University of Sousse (Tunisia). He received his Engineering Diploma from the Ecole Centrale de Lille (France) in 1995 and his Master's degree in Production Engineering and Computer Science from the University of Lille (France) in the same year. He obtained his PhD in Production Management, Automatic Control and Computer Science from the University of Science and Technologies of Lille (France) in 1998 and his "Habilitation to Supervise Researches" degree in Computer Science from the same university in 2003. Pr. Korbaa has published around 150 research papers on optimistation, simulation and modeling, applied and computational mathematics, manufacturing engineering and computer engineering.