Unsupervised anomaly detection : methods and applications

Andrian Putina

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAT012

Thèse de doctorat

Unsupervised Anomaly Detection Methods and Applications

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Paris, le 18-02-2022, par

ANDRIAN PUTINA

Composition du Jury :

Fabrice Rossi Professor, Université Paris-Dauphine	Président
Leman Akoglu Professor, Carnegie Mellon University	Rapporteur
Raja Chiky Research Director, Institut supérieur d'électronique de Paris	Rapporteur
João Gama Professor, University of Porto	Examinateur
Rita P. Ribeiro Professor, University of Porto	Examinateur
Mauro Sozio Professor, Télécom Paris	Directeur de thèse
Dario Rossi Chief Expert, Huawei	Co-directeur de thèse

a Flavia, a mamma, a Enrico, grazie ai quali sono la persona che sono

Acknowledgements

First of all I would like to show my gratitude to my advisors, Mauro Sozio and Dario Rossi. Their knowledge and expertise have allowed me to develop not only this thesis and my technical and research skills but also to become a better human being. Their support has pushed me to do my best, to always raise the bar a bit higher. I still remember with great pleasure, during the development of my master degree, Dario telling me *"Che faccio lascio?"* when I was proposing him my first research solution. I still remember Mauro asking me to dig deeper and understand better methods and results while assuring me of a better future post covid. I still remember the many team buildings that helped me meet new friends and colleagues, generating great union and collaboration in the team.

I would like to thank the members of my dissertation committee for their time, detailed review and suggestions. Their contribution helped further improving this manuscript.

I wish to thank my colleagues and friends from the office and the people for the long journey we have shared together during all these years. *23 Avenue d'Italie, Paris, France* will remain forever in my memories, Lincs offices made me understand what collaboration, support and second family means. I love you all Lincs people. You kept me sane over the years and offered endless support and encouragement. I loved having breakfast and lunch with you. Thank you Leonardo, Andrea and Jon for sharing with me great experiences, travels around the globe and your previous experiences. Thank you Camila, Stefan, Stefano and Anna for making the last few months extremely fun. Our debates, over a cup of coffee, are the best. A big thank you also goes to the my friends in Turin, they are always there to make me laugh.

I want to thank Télécom Paris, Cisco and Huawei for giving me access to their respective networks, resources and grants. I met great people in all these places.

I want to express my most sincere gratitude to my family, for being always there, for having supported my studies and for their constant presence and thoughtfulness.

Finally, I am deeply thankful to Flavia, the one and only lighthouse in the dark during these years. All this would not have been possible without her help, patience and support. Without her, I would not be in this stage now. I am really not able to put in words how grateful I am. Thanks.

Contents

List of Figures

List of Tables

Introduction en français

Motivation et applications

L'objectif principal de cette thèse est d'extraire des modèles de données et des anomalies dans un large éventail d'applications du monde réel en utilisant des algorithmes et des méthodes évolutifs. Les anomalies peuvent s'avérer vitales dans de nombreux scénarios tels que les réseaux informatiques, les fraudes et les soins de santé. Par exemple, la détection d'anomalies dans les réseaux informatiques et les transactions par carte de crédit est essentielle pour des raisons de sécurité, tandis qu'elle pourrait s'avérer indispensable pour détecter les défauts des moteurs ou les masses cancéreuses dans les images. Par conséquent, ces problèmes du monde réel ont suscité un nombre croissant de questions de recherche.

Une anomalie (également connue sous le nom de *outlier*) est une instance qui s'écarte considérablement du reste des données [66] et étant défini par Hawkins [72] comme "*une observation qui s'écarte tellement d'autres observations que l'on peut soupçonner qu'elle a été générée par un mécanisme différent.*.'

La détection d'anomalies (également connue sous le nom de détection de valeurs aberrantes ou de nouveautés) est donc le domaine de l'apprentissage automatique et de l'exploration de données dont l'objectif est d'identifier les instances dont les caractéristiques semblent être incompatibles avec le reste de l'ensemble de données. Dans de nombreuses applications, il est très important de distinguer correctement l'ensemble des points de données anormaux (*outliers*) de l'ensemble des points de données normaux (*inliers*). Une première application est le nettoyage des données, c'est-à-dire l'identification des mesures bruyantes et fallacieuses dans un ensemble de données avant l'application ultérieure d'algorithmes d'apprentissage. [56, 151].

Cependant, avec la croissance explosive du volume de données pouvant être collectées à partir de diverses sources, par exemple les transactions par carte, les connexions Internet, les mesures de température, etc., l'utilisation de la détection d'anomalies devient une tâche autonome cruciale pour la surveillance continue des systèmes. Dans ce contexte, la détection d'anomalies peut être utilisée pour détecter les attaques d'intrusion en cours [160], réseau de capteurs défectueux [122] ou des masses cancéreuses [120].

Cette quantité croissante d'applications et de modèles de données génère une demande pour des solutions de plus en plus sophistiquées capables d'être performantes dans de nombreux scénarios et applications. Par conséquent, cette thèse se concentre sur plusieurs défis qui émergent dans la tâche de détection d'anomalies, en proposant des algorithmes innovants capables d'être performants dans différentes exigences et applications.

Objectifs

Cette thèse vise à proposer des méthodes pratiques pour la détection d'anomalies dans trois domaines différents. Plus précisément, nous proposons :

- une nouvelle batch méthode de détection d'anomalies non supervisée qui améliore les performances des algorithmes non supervisés existants. L'algorithme s'attaque à trois défis principaux dans la détection d'anomalies : le problème de la dimensionnalité dans les grands ensembles de données qui a un impact important sur la précision de la plupart des algorithmes, le problème de la sélection des hyperparamètres dans différents ensembles de données et enfin les problèmes d'évolutivité dans les grands ensembles de données.
- un nouveau moteur de détection d'anomalies de séries temporelles basé sur le regroupement incrémentiel qui s'attaque aux problèmes d'évolutivité dans les environnements de surveillance dans lesquels de nouvelles instances sont disponibles en mode continu.
- une méthode de détection automatisée des anomalies qui s'attaque au problème de la sélection des hyperparamètres des algorithmes sur différents ensembles de données. Elle

utilise des algorithmes existants de détection d'anomalies comme blocs de construction et leur attribue des poids proportionnels à la preuve qu'ils sont des modèles performants dans la tâche de détection.

Plan de la thèse

Suivant les objectifs de la thèse, nous organisons la thèse en quatre parties principales. A partir du chapitre 2 nous discuterons des travaux connexes et de l'état de l'art des algorithmes de détection des anomalies en tenant compte des modèles *batch* et de *time-series*. Dans la dernière partie du chapitre, nous fournissons une explication détaillée de tous les algorithmes utilisés dans les chapitres suivants. Les contributions aux modèles batch sont présentées au chapitre 3 tandis que ceux liés aux modèles de time-series sont présents dans le chapitre 4. La contribution à la détection automatisée des anomalies est décrite au chapitre 5. Enfin le chapitre 6 conclut cette thèse en résumant les contributions et en présentant quelques idées pour les travaux futurs.

- Chapter 2 présente une revue des stratégies et techniques de détection des anomalies. Nous considérons et expliquons en détail, pour les familles *batch* et *time-series*, les algorithmes bien connus et ceux qui se sont avérés être les plus performants dans les études et comparaisons précédentes. Cette revue permet de construire une base solide pour les algorithmes utilisés dans les chapitres suivants par rapport aux méthodes proposées dans cette thèse.

- Chapter 3 propose une approche basée sur les arbres pour la détection non supervisée des anomalies, appelée *Random Histogram Forest (RHF)*. L'algorithme résout le problème de la dimensionnalité en utilisant le quatrième moment central (alias *kurtosis*) dans la construction du modèle, tout en présentant un temps d'exécution linéaire. Les résultats démontrent que la méthode améliore la précision globale de la détection des anomalies tout en étant robuste à la sélection des hyperparamètres.

- Chapter 4 présente un moteur de détection d'anomalies de time-series, appelé *ODS*, qui s'appuie sur DenStream, une technique de regroupement non supervisé, et l'applique aux mesures collectées à partir d'équipements de réseau réels étudiant ainsi une application *monitoring*. Notre campagne expérimentale compare plusieurs algorithmes du point de vue de la précision et de l'évolutivité : les résultats montrent que ODS (i) obtient des résultats

de détection comparables aux méthodes de pointe et (ii) est nettement plus rapide que les autres approches, notamment plus de deux ordres de grandeur plus rapide que les modèles d'ensemble.

- Chapter 5 présente un moteur de détection automatique d'anomalies qui allège l'effort humain nécessaire pour traiter plusieurs algorithmes et hyperparamètres. En exploitant un pool de différents algorithmes de détection d'anomalies, notre méthode pondère automatiquement chacun d'entre eux avant d'agréger les résultats. L'évaluation expérimentale montre que notre méthode surpasse les méthodes automatisées non supervisées existantes basées sur des mesures de corrélation.
- Chapter 6 conclut cette thèse en résumant les contributions et présente quelques idées pour les travaux futurs.

Publications

Le contenu de cette thèse a été partiellement publié dans des conférences et revues internationales. Dans ce qui suit, nous présentons la liste des articles publiés ou en cours de révision

- Andrian Putina, Mauro Sozio, Dario Rossi, José M Navarro, "Random Histogram Forest for Unsupervised Anomaly Detection". 2020 IEEE International Conference on Data Mining (ICDM).
- Andrian Putina, Dario Rossi, "Online Anomaly Detection Leveraging Stream-Based Clustering and Real-Time Telemetry". IEEE Transactions on Network and Service Management.
- Andrian Putina, Dario Rossi, Albert Bifet, Steven Barth, Drew Pletcher, Cristina Precup, Patrice Nivaggioli, "Telemetry-based stream-learning of BGP anomalies". Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks.
- Andrian Putina, Steven Barth, Albert Bifet, Drew Pletcher, Cristina Precup, Patrice Nivaggioli, Dario Rossi, "Unsupervised real-time detection of BGP anomalies leveraging high-rate and fine-grained telemetry data". IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops.

- Maroua Bahri, Flavia Salutari, Andrian Putina, Mauro Sozio, ”AutoML: state of the art with a focus on anomaly detection, challenges, and research directions”. International Journal of Data Science and Analytics
- (Under Review) Andrian Putina, Mauro Sozio, Dario Rossi, José M Navarro, ”Random Histogram Forest for Unsupervised Anomaly Detection: An Extensive Experimental Evaluation”.
- (Under Review) Putina,A.,Salutari,F.,Bahri,M.Sozio,M.,“AutoAD: an Automated Framework for Unsupervised Anomaly Detection”

Chapter 1

Introduction

1.1 Motivation and Applications

The main goal of this thesis is mining data patterns and anomalies in a wide range of real-world applications using scalable algorithms and methods. Anomalies could prove to be vital in many scenarios as computer networks, frauds and healthcare. For example, anomaly detection in computer networks and credit card transactions is critical for security reasons while it could prove to be indispensable in detecting faults in engines or cancerous masses in images. As a result, these real-world problems have posed an increasing amount of research questions.

An anomaly (also known as *outlier*) is an instance that significantly deviates from the rest of the input data [66] and being defined by Hawkins [72] as "*an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism*."

Anomaly detection (also known as outlier or novelty detection) is thus the machine learning and data mining field with the purpose of identifying those instances whose features appear to be inconsistent with the remainder of the dataset. In many applications, correctly distinguishing the set of anomalous data points (*outliers*) from the set of normal ones (*inliers*) proves to be very important. A first application is data cleaning, i.e., identifying noisy and fallacious measurement in a dataset before further applying learning algorithms [56, 151].

However, with the explosive growth of data volume collectable from various sources, e.g., card transactions, internet connections, temperature measurements, etc. the use of anomaly detection becomes a crucial stand-alone task for continuous monitoring of the systems. In this context, anomaly detection can be used to detect ongoing intrusion attacks [160], faulty sensor networks [122] or cancerous masses [120].

This increasing amount of applications and data patterns generates a demand for more and more sophisticated solutions capable to perform well in many scenarios and applications. Therefore, this thesis focuses on several challenges that emerge in the anomaly detection task offering innovative algorithms able to perform well under different requirements and applications.

1.2 Objectives

This thesis aims at proposing practical methods for anomaly detection in three different areas. More specifically we propose:

- a novel batch unsupervised anomaly detection method which enhances the performance of existing unsupervised algorithms. The algorithm tackles three main challenges in anomaly detection: the curse of dimensionality problem in large datasets which heavily impacts the accuracy of most of the algorithm, the hyperparameter selection problem in different datasets and finally the scalability issues on large datasets.
- a novel time-series anomaly detection engine based on incremental clustering which tackles the scalability issues in monitoring environments in which new instances are available in a stream mode.
- an Automated Anomaly Detection method tackling the problem of hyperparameter selection of algorithms on different datasets. It uses existing batch anomaly detection algorithms as building blocks and assigns them weights proportional to the evidence of being properly performing models in the detection task.

1.3 Thesis Outline

Following the thesis objectives, we organize the thesis into four main parts. Starting from Chapter 2 we will discuss related work and state-of-art anomaly detection algorithms considering both *batch* and *time-series* models. In the last part of the chapter we provide a detailed explanation of all the algorithms used in the following chapters. The contributions to the batch models follow in Chapter 3 while those related to the time-series models are present in Chapter 4. The Automated Anomaly Detection contribution is described in Chapter 5. Finally Chapter 6 concludes this dissertation by summarizing the contributions and presenting some ideas for future work.

- Chapter 2 presents a review of anomaly detection strategies and techniques. We consider and explain in detail, for both *batch* and *time-series* families, the well known algorithms and those which proved to be the best performing ones in previous studies and comparisons. This review builds a solid background of the algorithms used in the following chapters when compared to the methods proposed in this thesis.

- Chapter 3 proposes a batch tree-based approach for unsupervised anomaly detection, called *Random Histogram Forest (RHF)*. The algorithm solves the curse of dimensionality problem using the fourth central moment (aka *kurtosis*) in the model construction while boasting linear running time. The experiment results demonstrate that the method enhances the overall accuracy of batch anomaly detection while being robust to hyperparameter selection.

- Chapter 4 introduces a time-series anomaly detection engine, called *ODS*, that leverages DenStream, an unsupervised clustering technique, and apply it to measurements collected from real network equipment studying so a *monitoring* application. Our experimental campaign compares several algorithms under both accuracy and scalability viewpoints: results testify that ODS (i) achieves detection results on par with state-of-art methods and (ii) is significantly faster than other approaches, notably over two orders of magnitude faster than ensemble models.

- Chapter 5 presents an Automated Anomaly Detection engine which alleviates the human effort required when dealing with several algorithms and hyperparameters. By leveraging a pool of different anomaly detection algorithms, our method automatically weights each of them before aggregating the results. The experimental evaluation shows that our

method outperforms existing unsupervised automated methods based on correlation measurements.

- Chapter 6 concludes this dissertation by summarizing the contributions and presents some ideas for the future work.

1.4 Publications

The content of this dissertation has been partially published in international conferences and journals. In the following we report the list of papers published or under review:

- Andrian Putina, Mauro Sozio, Dario Rossi, José M Navarro, ”Random Histogram Forest for Unsupervised Anomaly Detection”. 2020 IEEE International Conference on Data Mining (ICDM).
- Andrian Putina, Dario Rossi, ”Online Anomaly Detection Leveraging Stream-Based Clustering and Real-Time Telemetry”. IEEE Transactions on Network and Service Management.
- Andrian Putina, Dario Rossi, Albert Bifet, Steven Barth, Drew Pletcher, Cristina Precup, Patrice Nivaggioli, ”Telemetry-based stream-learning of BGP anomalies”. Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks.
- Andrian Putina, Steven Barth, Albert Bifet, Drew Pletcher, Cristina Precup, Patrice Nivaggioli, Dario Rossi, ”Unsupervised real-time detection of BGP anomalies leveraging high-rate and fine-grained telemetry data”. IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops.
- Maroua Bahri, Flavia Salutari, Andrian Putina, Mauro Sozio, ”AutoML: state of the art with a focus on anomaly detection, challenges, and research directions”. International Journal of Data Science and Analytics
- (Under Review) Andrian Putina, Mauro Sozio, Dario Rossi, José M Navarro, ”Random Histogram Forest for Unsupervised Anomaly Detection: An Extensive Experimental Evaluation”.
- (Under Review) Putina,A.,Salutari,F.,Bahri,M.Sozio,M.,“AutoAD: an Automated Framework for Unsupervised Anomaly Detection”

Chapter 2

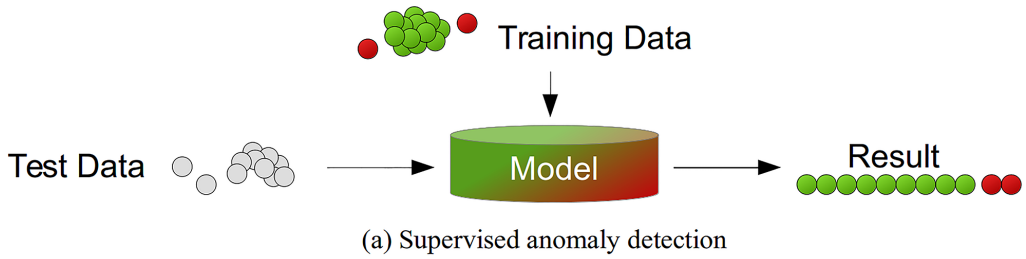
Background

In this chapter, a comprehensive and detailed review of existing machine learning anomaly detection strategies and methods is provided. We first formally describe the concept of anomaly detection and novelty detection. We further discuss a general taxonomy for the task of anomaly detection introducing the three types of anomalies, the three big families of algorithms, the evaluation metrics, research problems and challenges. We then describe different anomaly detection strategies (Section 2.4) and discuss the main difference between *batch* and *stream* algorithms afterwards (Section 2.5). Well known batch algorithms and those which proved to be the best performing ones in previous studies are described in Section 2.6 while stream algorithms are illustrated in Section 2.7. All such methods are used in the evaluation phase of the following chapters against our proposed methods.

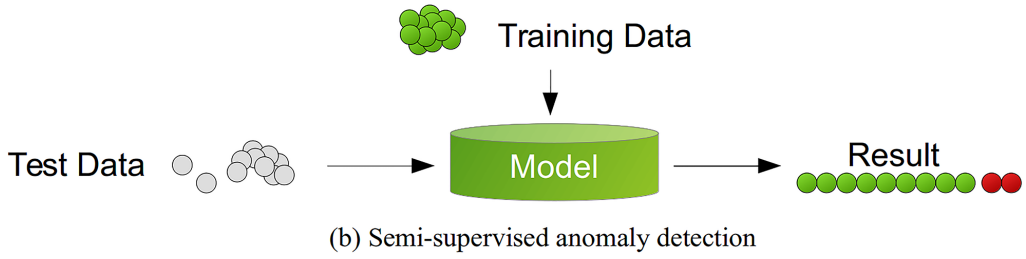
2.1 Machine Learning and Anomaly Detection

Anomaly Detection consist of a wide range of techniques and methods concerning the identification of abnormality in the data. It can be performed relying on expert knowledge in which specific application experts recognize anomalous patterns or through the usage of Statistics and Machine Learning.

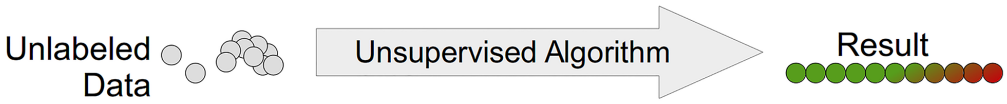
Arthur Samuel first popularized the concept of Machine Learning [129] defining it as: *the field of Artificial Intelligence and Computer Science that gives the computer the ability to learn without being explicitly programmed*. In this scenario, given a general dataset $X = \{x_1, x_2, ..., x_n\}$ with



(a) Supervised anomaly detection



(b) Semi-supervised anomaly detection



(c) Unsupervised anomaly detection

Figure 2.1: Differences between *Supervised*, *Semi-Supervised* and *Unsupervised* Anomaly Detection techniques. Image Source: [62]

$x_i \in \mathbb{R}^d, i \in \{1, 2, ..., n\}$ where *n* denotes the number of instances, *d* the number of dimensions in the dataset and a label space $Y = \{y_1, y_2, ..., y_n\}$ representing the output variable, a *model* optimizes a function *h* such that $h(x_i) \rightarrow y_i$. Such models fall under the category of *supervised* models. Based on the absence, full or partial, of the target variable Y models can be further split into *unsupervised* and *semi-supervised* ones (illustrated in Fig. 2.1). The former are not provided with labels so algorithms have to devise and discover data patterns on their own. The latter are provided with only a small fraction of labeled data, often belonging to only one class, forcing the model to build a decision boundary around the provided class.

Anomaly Detection, known also as *outlier, novelty* and *noise* detection, is thus the task of pinpointing, through the usage of models, instances whose characteristics significantly deviate from the knowledge learned. However, while *anomaly detection* is usually interchanged with *outlier* and *novelty detection* these two can be better defined as:

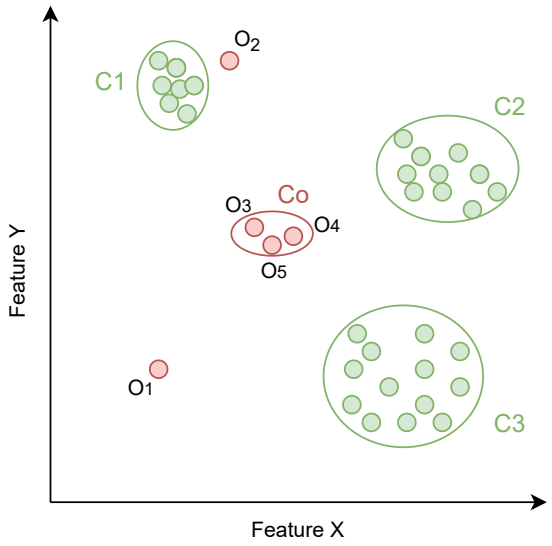


Figure 2.2: Two-dimension toy Dataset representing a *Global Point Anomaly* (e.g. O_1), a *Local Point Anomaly* (e.g. O_2) and a *Group Anomaly* (e.g. cluster Co composed by O_3, O_4 and O_5).

Outlier Detection is the task of identification of an instance $x \in X$ which significantly deviates from the remainder of the input instances X. In such a scenario the model has access to the full dataset X which is populated by both *normal* and *anomalous* instances. The models can thus fit and estimate regions and boundaries between the two classes.

Novelty Detection is the task of identification of a novel instance $x \in X'$ according to the observations in X where X' is the set of new observations. In such a scenario the model has access only to the initial set of instances X, not polluted by outliers, and has to detect new patterns in X'. Such models can be further trained or updated when new instances are available and fall under the category of **Incremental** or **Stream** models.

While *outlier detection* highlights anomalous instances in a dataset X, *novelty detection* first fits a dataset with no outliers and successively examines new instances X' for anomalies. Therefore the two share the common objective and principles of abnormal data identification but, based on requirements such as memory usage and data availability, they are used in different applications and scenarios.

2.2 Types of anomalies

It is possible to group anomalies in: *Point Anomalies*, *Group Anomalies* and *Contextual Anomalies*. Based on their characteristics, anomalies are harder or easier to detect. Consider Fig. 2.2 depicting a hypothetical two-dimension dataset populated by 5 anomalies. O_1 is the easiest instance to detect as it is "far" from any clusters in the data. O_1 is thus a *point anomaly* but so it is O_2. The difference between the two is that while O_1 is globally anomalous with respect to all the other instances in the dataset, O_2 is harder to detect as it is anomalous only with respect to instances in the cluster C_1. O_1 is a *global point anomaly* while O_2 is a *local point anomaly*. An even more difficult task for anomaly detection algorithms is the detection of the anomalous cluster Co composed by O_3, O_4 and O_5. The anomalous cluster indeed contains only a small fraction of the overall data. It is thus very challenging to define an universal measurement to detect *global, local* and *group anomalies*.

Consider now Fig. 2.3. t_2 is a typical *contextual anomaly* which are commonly found in time-series. The plot shows the monthly temperature over a few years timespan. When the contextual information is used (e.g. seasonality) as data source, a low temperature might be normal during the winter (at time t1), but the same value during summer (at time t2) would be an anomaly. Formally:

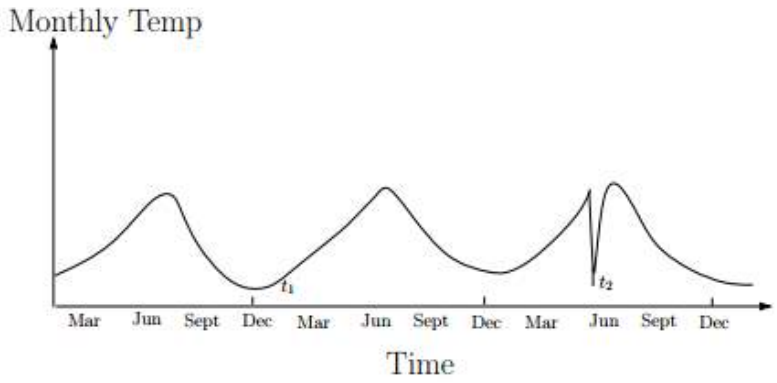


Figure 2.3: Example of a *Contextual Anomaly*. The plot shows the monthly temperature over a few years timespan. When the contextual information is used (e.g. seasonality) as data source, a low temperature might be normal during the winter (at time t1), but the same value during summer (at time t2) would be an anomaly. Image source: [34]

- **Global Point Anomalies**: Is an instance x that deviates significantly from all the remaining instances X. For example, O_1 in Fig. 2.2 is far from all the remaining instances.
- **Local Point Anomalies**: Is an instance x that deviates significantly only with respect to a subset of instances $C_x \subset X$. For example, O_2 in Fig. 2.2 is far only from C_1.
- **Group Anomalies**: Is a set of instances $X_o \subset X$ that do not follow regular patterns observed by other set of instances in $X_n \subset X$. For example, C_4 in Fig. 2.2 is composed by only 3 instances while other clusters contain at least 8 instances.
- **Contextual Anomalies**: Is an instance x that deviates significantly from all the remaining instances X according to a certain context. For example, t_2 in Fig. 2.3 is similar to t_1 from an absolute value point of view but it is anomalous when the context (seasonality/month) is considered.

2.3 Performance Evaluation

The evaluation of anomaly detection algorithms is performed taking into account their outputs and comparing them against known *ground truth*, known also as *labels*. For each instance $x \in X$, the models output an anomaly score which can be ranked from the most anomalous to the most normal one. By setting an outlierness threshold each instance is finally classified as *normal* or *anomalous*. Typical metrics used in machine learning are *True Positive Rate (TPR)*, *False Positive Rate (FPR)*, *Precision*, *Recall* and F_1 defined as:

$$TPR = Recall = \frac{tp}{tp + fn} \tag{2.1}$$

$$FPR = \frac{fp}{fp + tn} \tag{2.2}$$

$$Precision = \frac{tp}{tp + fp} \tag{2.3}$$

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall} \tag{2.4}$$

where tp, tn, fp and fn represent respectively *true positive*, *true negative*, *false positive* and *false negative*.

By considering all possible thresholds (from smallest to largest anomaly score) it is possible to compute, for each of them, a tuple FPR and TPR. Such tuples generate the *Receiver Operating Characteristic (ROC)* curve: an overall measure which trades-off FPR (x axis) vs TPR (y axis). The perfect classifier reaches an optimal ROC score of 1.

Similarly, the Precision-Recall curve, known also as Average Precision (AP), is generated using the same principles of the ROC one with the difference that it trades-off Recall (x axis) vs Precision (y axis). An example of both ROC and AP curves are present in Fig. 2.4.

We observe that the Receiver Operating Characteristic (ROC) is often employed to evaluate anomaly detection methods [62]- [92]. However, it has been shown in [39] that when the classes are not balanced (e.g. anomaly detection task) the *AP* curves better reflect the efficacy of an

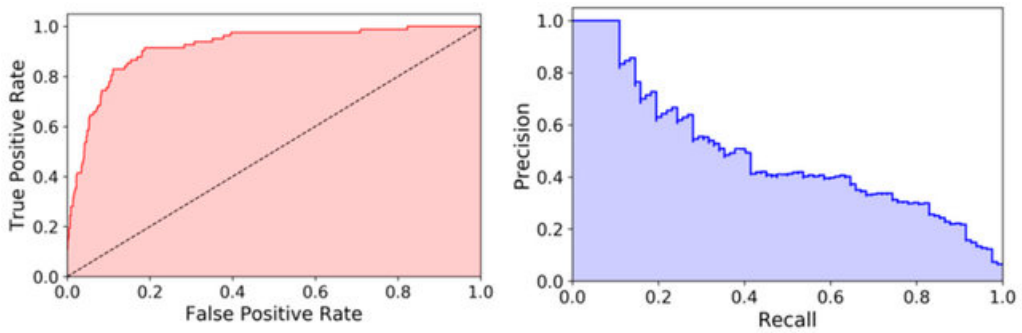


Figure 2.4: Example of a *Receiver Operating Characteristic (ROC)* curve (a) and *Average Precision (AP)* curve (b). Image Source: [21]

algorithm. [39] shows moreover that a curve dominates in ROC if and only if it dominates in AP space. We evaluate thus the performance of the algorithms by measuring the *Average Precision (AP)* of the *Precision-Recall Area Under the Curve* [159] (without interpolation):

$$AP := \sum_{n} (R_n - R_{n-1})\, P_n \tag{2.5}$$

where $P_n = \frac{tp}{tp+fp}$ and $R_n = \frac{tp}{tp+fn}$ are the Precision and Recall at the n^{th} threshold.

2.4 Machine Learning Anomaly Detection Strategies

Machine Learning Anomaly Detection techniques can be categorized [34] into three big families according to the input type: *supervised* anomaly detection, *unsupervised* anomaly detection and *semi-supervised* anomaly detection (see. Fig.2.1).

In the case of *supervised* models, the task of anomaly detection is reduced to a binary classification in which *normal* and *anomalous* instances are considered instances of two classes $\{0,1\}$. When treating imbalanced datasets, especially true in the case of anomaly detection, it is usually recommended to use re-sampling methods (down/up-sampling) as supervised methods under perform when a class is under-sampled or not well represented [14]. Such models are trained on a training dataset and tested on unseen data. Usually supervised methods provide the best

detection rates as they have access to the labels in the training phase. However, they are rarely usable in many different applications due to the lack of labels. Moreover, labeling is very expensive as it requires a huge amount of human effort to annotate instances.

Among the supervised approaches we mention Random Forests [91], One Class Support Vector Machines [125] and, with recent advent of Neural Networks [40], Autoencoders [158] (based on reconstruction) and Generative Adversarial Networks [63]. More details about Neural Networks used for Anomaly Detection can be found in [111, 37] while more supervised techniques can be found in the following related surveys [79, 34, 16, 17, 14].

Although supervised methods provide the best detection rates, two main challenges arise when using such models in anomaly detection. The first one is linked to the rare nature of anomalies and consequently to the class imbalance problem. Such problem is addressed in many machine learning studies [80, 82, 35]. Secondly, obtaining accurate and a representative set of labels is not only challenging but would require also a large human effort to generate and maintain them. This problem is addressed using re-sampling methods [14] or by injecting artificial anomalies into normal datasets [142, 137]. Supervised anomaly detection is thus similar to building normal predictive/classification models hence we will not address this category anymore.

Semi-Supervised models are fit only on one class and are known also as *one-class classification* (OCC) models. Such algorithms build decision boundaries only around the provided class (usually normal data) and classify instances as either belonging or not belonging inside the decision boundary. They are used when anomalies are very difficult to capture. For example in the space craft [57], a single anomalous instance would require an accident. However, such algorithms poorly perform [141] for many reasons. Not only some of the features might be irrelevant but it is difficult to understand how tight the boundary should be set especially when the training set is contaminated by anomalous instances. Moreover inappropriate training set not covering the full boundaries of normal behavior originates poorly performing models generating lots of false positives.

Unsupervised models do not use *a priori* knowledge of data. In other words, no label is provided to the algorithm. They are based on the assumption that normal instances are more

frequent than anomalies. Typically they group objects together (e.g. clustering) based on similarity functions such as distance-based measurements between instances in the dataset. However performances are often linked to the choice of the input hyperparameters which cannot be known in advance. Detection is achieved identifying first shared patterns among data and pinpointing instances which deviate from them. As a result, unsupervised anomaly detection has received increasing attention in recent years (e.g. *Isolation Forest* [92], *PIDForest* [64], *XSTREAM* [97], *OCSVM* [131], *LOF* [27]).

Due to the limitations of supervised and semi-supervised methods described in the previous paragraphs we will focus in the following chapters mainly on unsupervised methods. More details about the strategies not considered in this thesis (e.g. supervised, semi-supervised, etc) can be found in the following books [15, 100] while general anomaly detection challenges and future directions are present in [17, 79, 34, 16, 14].

2.5 Batch vs Stream Mode

Based on the data type, memory usage and requirements, algorithms can be further categorized into *batch* and *stream* mode models.

Batch algorithms, usually, process a set of instances X in batches and simultaneously. They require to load all the data in memory and through the usage of predefined metrics pinpoint *anomalous* instances. Batch mode is thus used on a fixed dataset whose data size is known, finite and all the instances fit in memory. Several batch algorithms require to process data in multiple passes and update so inner model variables as distances, clusters centroids, radius etc. By definition, data collection introduces latencies between the arrival time and the processing time.

Stream algorithms, instead, are required to process a continuous stream of data immediately as it is produced. They usually analyze newly available instances as soon as they are generated, before discarding them. If such data processing is fast enough (e.g. sub-seconds) it is perceived as real-time processing. Stream algorithms thus process instances in only one or a few passes. Such methods are thus used on possibly infinite continuous streams whose size and distribution are unknown. The main differences between batch and stream algorithms are summarized in

Batch Processing	Stream Processing
Instances collected over a period of time	Instances streamed continuously
Instances processed all at once	Instances processed piece-by-piece
Data size is known and finite	Data size is unknown and possibly infinite
Multiple passes on the data	Single pass on the data
No real-time analytics results required	Real-time analytics results required

Table 2.1: Main difference between batch vs stream algorithms

Table 2.1.

Due to the nature of the data (known and finite size), batch algorithms are mostly used in the *outlier detection* task. In such scenario indeed, data is collected over a period of time and it is not required to generate real-time results. Several algorithms (e.g. DBScan described in Section 2.6) perform multiple passes on the data updating its model inner variables such as number of clusters, centers, radiuses etc.

Stream algorithms, instead, are required to process a continuous stream of instances immediately as it is produced and no *a priori* assumption can be done on the data whose size could be infinite. In such a scenario instances cannot be stored and have to be discarded after the processing phase forcing the algorithms to perform a single pass on all the instances. Such algorithms are suited for *novelty detection*.

Data streams, as previously defined, do not assume any correlation between subsequent instances. In such scenario all instances are independent and do not depend on previous instances. An example of such streams is the fraud detection in which transactions of different clients are independent. When in the stream the order and the arrival time are relevant, i.e. new instances depend on previous instances, we deal with timeseries. In such scenario the time is an important feature of the data and several algorithms extract data patterns from the past to predict future events. Typical applications of time series analysis are forecasting applications in demand of electricity, price stocks etc. To summarize, a time series is a sequence of instances composed by successive measurements made over a time interval which present time correlation while a data stream is more generally a sequence of instances in which there is not data ordering or time-dependency between streaming instances.

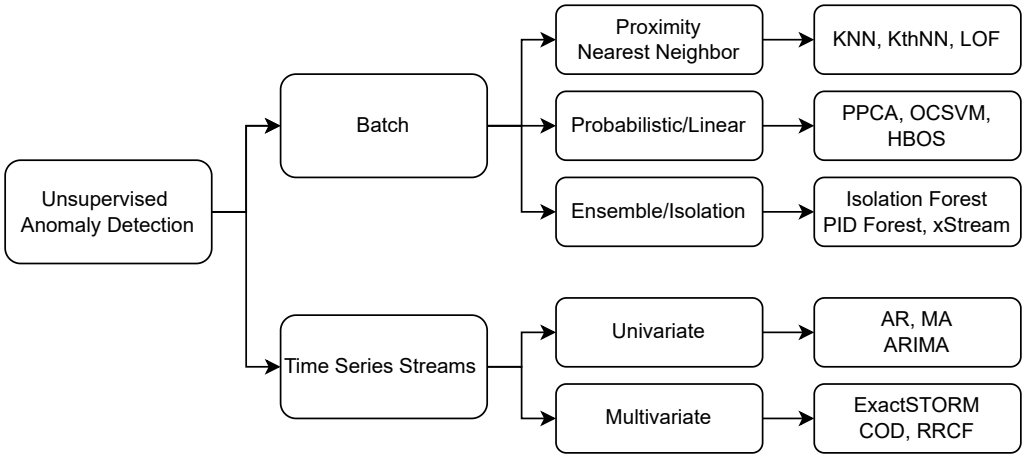


Figure 2.5: Summary of Anomaly Detection Algorithms. Related work is grouped in two branches depending on the underlying algorithmic family: batch vs stream. Batch algorithms are further classified in Proximity vs Probabilistic vs Ensemble methods. Stream methods are differentiated into univariate vs multivariate methods.

In the literature, the term *streaming* is often used to address time series and authors do not always agree on the definitions. For example, authors in [97] define a stream as a sequence of instances in which there is not data ordering or time-dependency between streaming instances while authors in [67] use the same terminology (i.e. streams) referring to time series. While many streaming algorithms (e.g. LODA [114], xStream [97], Rs-Hash [130]) can be applied also on time series sequences, we will discuss and analyse in the following chapters algorithms designed to detect anomalies in time series or which were used in the literature to perform such task. In Chapter 4 we will compare such algorithms in a monitoring application in which data is produced in form of time series.

Fig. 2.5 illustrates a summary of both batch and time series methods and further classifies them in subcategories. More details about such models are provided in Section 2.6 and Section 2.7

2.6 Batch Methods

Several batch unsupervised anomaly detection algorithms have been developed in recent years. We can classify the related work as follows:

- (i) *Proximity/Nearest-Neighbor* based methods (e.g. *K-NN*, K^{th}*-NN*, *Local Outlier Factor*);
- (ii) *Probabilistic/Linear* based methods (e.g. *PPCA*, *HBOS*, *OCSVM*, etc.);
- (iii) *Ensemble/Isolation* based methods (e.g. *Isolation Forest*, *PIDForest*, *XSTREAM*).

Proximity/Nearest-Neighbor based methods compute the neighborhood of all the instances $x \in X$ and uses the distances of each instance x to describe abnormality. Such methods assume that *normal instances are close to remaining instances while anomalies are far away from the neighbors.* All the methods use a *distance* measurement to detect close and far instances. Several distance measurements have been proposed during the years [43] such as the *Mahalanobis* distance [96] and the well known and most used *Euclidean* distance:

$$D_{Euclidean} = \sqrt{\sum_{i}^{d} ||u_i - v_i||^2} \tag{2.6}$$

where u and v are two d-dimensional instances in X.

Anomalies can be detected using distance measurements by either i) employing the distance itself as *anomaly scores* or ii) using distances to group close instances into *clusters* and point out afterwards instances not belonging to none of the clusters. A detailed survey comprising distance based method is present in [34].

K-NN [20]: computes the distances for each instance $x \in X$ to the k-nearest-neighbors and assign them the mean distance to its k nearest neighbors where k is an integer number. The running complexity of this method is $\mathcal{O}\left(n^2\right)$ while the number of input hyperparameters is one: k. Setting the input hyperparameter k is not always easy as it depends on the application and domain knowledge is required; this is especially true when duplicates are present in the dataset.

Kth-NN [124]: computes the distances for each instance $x \in X$ to the k-nearest-neighbors and assign the distance to its kth nearest neighbors where k is an integer number. The running

complexity of this method is $\mathcal{O}\left(n^2\right)$ while the number of input hyperparameters is one: k. As in the previous case, the input hyperparameter depends most of the times on the application and domain knowledge is required.

DBScan [52]: is a data clustering algorithm proposed by Ester et al. [52]. It is a density-based clustering algorithm: it computes the distances between the samples and clusters altogether the points which are neighbors (i.e. whose distance is less than ϵ). By computing the ϵ neighborhood of each point, it is able to discover clusters of arbitrary shape. The points falling in low-density regions (whose nearest neighbors are far) are labeled as *noise*. In particular, the algorithm defines the local point density p by two hyperparameters: ϵ and $MinPts$. The first one is the radius and defines the neighborhood of a point p. The neighborhood is thus the group of all the points within the radius ϵ from the point p:

$$N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\} \tag{2.7}$$

$MinPts$ instead defines the minimum number of samples in a given neighborhood. Combining ϵ and $MinPts$, the density of an object p is $high$ when in the $N_\epsilon(p)$ there are $MinPts$ or more and low otherwise. Points with $high$ density are called $core$ points while low density points in the neighborhood of a core point are called $border$ points. Finally low density points are outliers. The clusters are obtained grouping together density-reachable points where a point *p* is directly density-reachable from a point q if $p \in N_\epsilon(q)$ and $|N_\epsilon(q)| > MinPts$. The outliers are the points not belonging to any cluster C_i.

The running complexity of the algorithm is $\mathcal{O}\left(n^2\right)$ [68] reducible, only when $n >> d$ and $d < 3$ using efficient indexing data structures (i.e. R^*-tree), to $\mathcal{O}\left(n \log\left(n\right)\right)$ [52] on average. For $d > 3$ [58] shows it is possible to improve the expected running time to $\mathcal{O}\left(n^{2-\frac{2}{\lfloor d/2 \rfloor + 1} + \delta}\right)$ where $\delta > 0$ can be an arbitrarily small constant. The input hyperparameters of the algorithms are ϵ and $MinPts$. Both the hyperparameters are strongly dependent on the application and input dataset as shown also in our experiments in Chapter 4.

Local Outlier Factor (LOF) [28]: is a data clustering algorithm proposed by Breunig et al. [28] to detect *anomalous* data point by measuring local densities. The algorithm computes the local density of a point as the distance of the k-th nearest neighbor. It is possible in this way to identify regions of similar densities and the points with low density are considered outliers.

In particular LOF computes, for each sample, the euclidean k-distance d_k to its k-th nearest neighbor. Distances are then used to compute the reachability distance of two points p and q as the maximum of their real distance and the k-th distance of the second point.

$$reachDist_k(p,q) = max(k - dist(q), dist(p,q)) \tag{2.8}$$

and the local reachability distance of an point p as the average reachability distance of p from its neighbors:

$$lrd_k(q) = {}^1/\left[\frac{\sum_{q \in N_K(p)} reachDist_k(p,q)}{|N_k(p)|}\right] \tag{2.9}$$

In the end, the local outlier factor of an object is the average local reachability distance of the neighbors divided by the object's own local reachability density, i.e.:

$$LOF_k(p) = \frac{\sum_{q \in N_k(p)} \frac{lrd(q)}{lrd(p)}}{|N_k(p)|} \tag{2.10}$$

Objects with similar density will have a LOF value close to 1, higher density objects (*normal*) will have LOF lower than 1 while a LOF value higher than 1 will mean an object which has a lower density than the neighbors and thus an *outlier*. The decision function can be properly tuned by observing the scores of the top-most abnormal samples (*contamination*) in the dataset. LOF hyperparameters are $n\_neighbors$ and *contamination* and its running complexity is $\mathcal{O}(n \log(n))$ [28]. As for the all distance based methods the input hyperparameters are again dependent on the application and dataset as shown also in our experiments in Chapter 4

Probabilistic based methods estimate the parameters θ of the dataset X and assigns to each instance $x \in X$ an anomaly score equal to the likelihood $P(X \mid \theta)$. Such models assume that *instances are generated by the same stochastic source*. After the fitting phase, anomalies are assigned low probability of being generated while high probabilities of being generated are assigned to normal instances.

Histogram-based Outlier Score (HBOS) [61]: generates a histogram for each feature assuming they are independent. Similar to the Naive Bayes approach in which all the independent feature probabilities are multiplied, HBOS outputs an anomaly score given by the multiplication of the inverse height of the bins of all the features. The only input hyperparameters is the number

of bins k which might be difficult to select depending on the data distribution. For such reason, authors propose two variants of the algorithm: the first one uses a fixed bin width whose running complexity is $\mathcal{O}(n)$ and a second version in which dynamic bin width is used whose running complexity is $\mathcal{O}(n \log(n))$

Probabilistic Principal Component Analysis (PPCA) [144]: estimates the principal components of the data and projects the d-dimensional dataset to a q-dimensional one estimating the latent variables by iteratively maximizing the likelihood function of the transformation

$$t = Wx + \mu + \epsilon \tag{2.11}$$

where t is the d-dimensional dataset, x is the q-dimensional transformation, W the $d \times q$ matrix which relates the two sets of variables, μ permits the model to have non-zero mean while ϵ is the error. The input hyperparameter of the algorithm is the number of components $n\_components$, estimated using Minka's MLE previously described while the running complexity is $\mathcal{O}(nd^2)$

One-Class SVM [131]: determines a separating hyperplane in a higher dimensional space by maximizing the distance from the hyperplane to the origin. The model is fit with training instances which contain only normal instances and not contaminated by outliers. Moreover, OCSVM supports also the ν which was introduced to control the effect of outliers in the fitting phase. The hyperparameter indeed acts as a trade-off between maximizing the distance of the hyper-plane from the origin and the number of data points that are allowed to cross the hyper-plane. Such method can be used also for the *novelty detection* task as, once trained, can easily estimate anomaly scores for new instances. The running complexity of the algorithm is $\mathcal{O}(n^3)$ while its input hyperparameter ν depends on the anomalies fraction in the dataset.

Ensemble/Isolation based methods isolate anomalies instead of profiling normal instances by recursively splitting the data through a random tree and generating so isolation forests. Unlike proximity-based methods, such methods does not rely on distance but they fragment the data space to identify instances laying far from other data points.

Isolation Forest [92]: builds a forest of randomly generated trees on a sub-sample of instances. To be more specific, it randomly selects a feature of the sub-sample and randomly selects a value for the feature to form a node of a binary tree called iTree determining a left child and right child. The splitting is repeated iteratively until it is not possible to split the data anymore (e.g. only one instances in the leaf). For each instance in the dataset $x \in X$ the termination node is computed and assigned the anomaly score equal to average path from the root nodes to the termination nodes in all the trees. The authors show empirically that shorter path lengths are representative of anomalies as they are more easily to isolate with respect to normal data. The input hyperparameters of the algorithm are the number of instances ψ and the ensemble size t. While it is common to set $t = 100$ in ensemble models, we show in Chapter 3 that ψ can strongly depend on the number of anomalous instances in the dataset. The running complexity of the algorithm is in the worst case $\mathcal{O}\left(t\psi^2\right)$ for the building phase and $\mathcal{O}\left(nt\psi\right)$ for the evaluation phase.

PID Forest [64]: builds a collection of decision trees that partition space into subcubes. Instead of using binary trees, PIDForest allows a finer partition splitting the data in *k* different regions at each level of the tree with the goal of having a large variance in the sparsity of the subcubes. The sparsity of a subcube C with respect to a dataset T is $p(C,T) = \frac{vol(C)}{C \cap T}$. Ultimately, the leaves with large *p* values will point to regions with anomalies. Similarly to *Isolation Forest*, each tree is built using a random sample $P \subseteq T$ of *m* points. Each node *v* in the tree corresponds to a subcube $C(v)$ and a set of points $P(v) = C(v) \cap P$. For the root, $C(v) = [0,1]d$ and $P(v) = P$. In each internal node a coordinate $j \in [d]$ partitions the data I_j into k intervals. The number of partitions k is a hyperparameter of the algorithm. The partitions stop when the tree reaches the maximum depth or when $P(v) \leq 1$. The key aspect of the algorithm is the choice of j which partitions the data in some sparse and dense regions. The problem of finding the partition along a coordinate j that maximizes the variance in the sparsity is reduced to the problem of finding a k-histogram for a discrete function $f : [n] \Rightarrow R$, which minimizes the squared $l2$ error. Finally, each tree maps each instance into a leaf node v and assigns a PIDScore(v). The 75% percentile score is the output score. The input hyperparameters of the algorithm are the number of regions k, the ensemble size t, max height h and the number of samples used ψ. While authors show performances to be steady for $k > 3$, and h the algorithm uses sub-sampling techniques (ψ) which results to depend on the number of anomalous instances in the dataset similarly to Isolation Forest.

XSTREAM [97]: xStream is an ensemble of Half-Space Chains that approximates density efficiently. Each chain approximates the density of a point by counting its nearby neighbors at multiple scales. The algorithm first applies a subspace-selection and dimensionality reduction via sparse random projections and subsequently builds and ensemble of Half-Space Chains to estimate density at multiple scale. In the random projection phase the algorithm selects a set of Gaussian random vectors $\{r_1, ..., r_k\}$ and projects each instance $x \in R^d$ to a low-dimensional embedding $y \in R^K$ as $y = (x^T r_1, ..., x^T r_k)$, where K is the number of random projections. The low-dimensional embedding preserves the pairwise distance between instances. Instead of a full set of Gaussian random vectors the authors propose to use sparse vectors where only 1/3 of the vector components are non-zero. Under this scheme, each projection ignores a 2/3 fraction of the feature space, essentially selecting a subspace, while maintaining the pairwise distances. The algorithm then estimates the density in the projected K-dimensional space denoted as $P = \{1, ..., K\}$. Each chain, of depth D randomly selects a split dimension $p \in P$ at each level $l = 1, ..., D$ and recursively splits the space along the dimension into discrete bins. If a feature is selected again at a subsequent level, it is discretized with a halved bin width, enabling density approximation at multiple scales. Finally, the multi-scale outlier score of an instance y from a chain is the minimum extrapolated bin-count across all levels, corresponding to the lowest density this instance has among all the considered granularities. The overall anomaly score is the average across all chains. The input hyperparameters of the algorithm are projection size k, the depth of the chains d and the number of chains t generating so the following complexity $\mathcal{O}(ntkd)$. While it is common to set $t = 100$ in ensemble models, authors show performances to be steady for a wide range of hyperparameters k and d. We confirm such results in our comparison in Chapter 3.

2.7 Time series streams

For what concerns anomaly detection in time series, it is possible to classify the algorithms in two big families:

- *Univariate* methods: approaches that analyse univariate time series (e.g. AR, MA, ARIMA, etc.)
- *Multivariate* methods: approaches that model multivariate time series (e.g. ExactSTORM, COD, RRCF, etc.)

Univariate algorithms are predictive models which extract patterns in a given time series with the final goal of predicting subsequent values. Several methods have been proposed ranging from traditional methods as Auto-regressive models (AR) [29], Moving Average model (MA) [51], ARIMA [25], Seasonal ARIMA [78] and Exponential Smoothing (ES) algorithm [60] to more recent strategies as LSTM Neural Network [134]. Such models are often simple to fit and use in real time applications. However, they are designed for univariate time series so they are not applicable in multivariate time series predictions, which limits their applications. This is especially true in monitoring applications where the number of sensors and measurements is very large.

Multivariate algorithms, instead of analyzing each time series at a time, extract patterns from the multivariate dataset in a manner similar to the one observed in batch algorithms. Such models define a decision function and assign an anomaly score to each processed instance. A first family of algorithms are state transition models (e.g. Markov Models [154] or Hidden Markov Model [90]) in which timeseries are assumed to maintain a steady state pattern which can be modeled as a state. However, the utilization of such methods can be cumbersome because states are often unknown in advance and usually the systems may dynamically change over time.

A second class of multivariate algorithms are evolving prediction models which can be used to update the models as new data arrives in order to better capture the trends in the data. For example, [152] presents a predictive model which incrementally learns the probabilistic mixture model of the data, using a decay factor to account for drifts. The anomaly score of each instance is the probabilistic fit value to the learned model. More predictive techniques can be found in the following related surveys [69, 86].

The bulk of the multivariate unsupervised anomaly detection algorithms for data streams is composed by the category of the distance based methods applied on sliding windows. Such methods assume that only recent data is relevant for the detection, as a result when the sliding window moves, old objects expire and new objects are added to the window. The algorithms usually define outliers all the instances in the window having less that *k* neighbors within a distance *R*. The most popular algorithms are:

ExactSTORM is an online anomaly detection method proposed by Angiulli et al. [19]. The method uses a sliding window model and stores the data instances in nodes of a suitable data structure called Indexed Stream Buffer (ISB). Each node contains the data instance *p*, its arrival time *p.t*, the number of succeeding neighbors *p.count_after* and the list, of size at most *k*, of the preceding neighbors *p.nn_before*. The ISB data structure supports the *range query search*, that given an object *p* and a real number $R > 0$, returns the objects in the ISB whose distance to *p* is not greater than R.

For each incoming instance *p*, a range query is performed in ISB which returns the list of its preceding neighbors *Q*. For each $q \in Q$, the number of succeeding neighbors *q.count_after* is increased by 1. Finally, *p* is inserted into the ISB while the expiring instance *o* is removed from the data structure.

All the operations previously described are performed by the so called *Stream Manager* which updates the ISB for each incoming instance. It is subsequently up to the *Query Manager* to scan the ISB searching for outliers.

For each instance *p* in the ISB, the *Query Manager* discriminates between *inliers* and *outliers* by considering the sum of the succeeding neighbors *p.count_after* with the size of the preceding neighbors list *p.nn_before*. If the sum is lower than *k*, then *p* is an outlier and inlier otherwise.

Continuous Outlier Detection (COD) is an online anomaly detection method proposed by Kontaki et al. [84]. Similar to [19], the method uses a sliding window model and stores the instances in a data structure that supports range queries efficiently (e.g. M-Tree). The authors observe that (i) a departing instance can transform inliers into outliers, (ii) an incoming instance can transform outliers into inliers and (iii) not all the instances are affected by the expiring ones. Since only the neighbors of the expiring instances have to be updated, COD uses a priority queue (Fibonacci queue) to schedule processing of affected instances.

The method stores for each instance of the stream *p*, its arrival time *p.time*, its expiration time *p.exp*, the list of its preceding neighbors *p.P* and the number of the succeeding neighbors $p.n^+$.

When a new instance *p* is available, the algorithm sets the expiration time to w samples from the current time. It subsequently performs a query which returns the list of objects (*p.P*) lying at distance at most *R* from *p*. Comparing the total number of neighbors against the threshold *k*, the algorithm discriminates inliers from outliers. If *p* is an outlier it is added to the outlier list; it is added to the inlier list and to the priority queue otherwise. The key in the priority queue is set according to the minimum expiration time of all its preceding neighbors *p.P*. For each instance $q \in p.P$, the number of succeeding neighbors $q.n^+$ is incremented by 1. When the total number of neighbors exceeds *k*, *q* is promoted to the inliers list and added to the priority queue. The key in the priority queue follows the minimum neighbors expiration time previously described.

When an instance *o* expires, it is removed from the range query data structure and from the preceding neighbors lists. To do so, the priority queue is polled until all the elements set to be checked in the current time are extracted. Each extracted element *q* is either added to the outlier list if its total number of neighbors falls below *k*, or its key is updated and q is reinserted into the queue otherwise.

Finally, ensemble methods constitute the last group of multivariate algorithms for anomaly detection in data streams. The algorithms (RRCF [67], LODA [114], xStream [97]) typically use sliding window models and are composed by a set of weak learners. Each learner partitions, updates, and maintains the data using random trees data structures. On one hand ensemble algorithms are the best performing algorithms in terms of accuracy. On the other hand, they are

slower than remaining algorithms as they have to update and maintain *t* different learners.

Robust Random Cut Forest (RRCF) is an streaming anomaly detection method proposed by Guha et al. [67]. It is an ensemble algorithm composed by *t* different models which maintains *w* instances (tree size) in binary trees. Each instance $p \in w$ is isolated in a leaf of the tree while internal nodes act as splitting nodes. Each internal node, in addition to splitting criterion (attribute and value) maintains also the dimensions bounding box (support) of all the instances in the sub-tree.

Given a set of instances *S* and a tree *T(S)*, when a new instance *p* is available, the algorithm tries to insert it from the root. It first sums together the supports of all the dimensions contained in the bounding box and extracts a random number $r \in [0, \sum_i (x_i^h - x_i^l)]$ which determines the attribute and the splitting value. If the split separates the instance *p* from the remaining tree, the algorithm generates a new splitting node with a branch containing *p* and another one containing the previous tree *T(S)*. If the split does not isolate *p*, then *p* follows the path of the existing tree and the procedure is repeated on each sub-tree until the instance is isolated. All the bounding boxes on the path of *p* are updated.

When an instance *o* departs, the algorithm removes its parent and replaces it with the sibling. All the bounding boxes starting from *o* upwards are updated. Notice that all the operations previously described (insertion, deletion, bounding box updating process, etc.) are repeated for each of the *t* trees in the ensemble.

The insertion and removal of each instance in the tree leads to a modification of the tree structures. The variation in tree complexity is used to determine the anomaly score. Given a set of points Z, a point p and a tree T, the depth of p is $f(p, Z, T)$. Assigning to each left branch of the tree the bit 0 and the right branches the bit 1, the model complexity $|M(T)| = \sum_{p \in Z} f(p, Z, T)$ is the number of bits required to write down the description of all points p in the tree. The bit-displacement of an instance *x* is:

$$Disp(x, Z) = \sum_{T, p \in Z - x} Pr[T] \left(f(p, Z, T) - f(p, Z - x, T') \right) \tag{2.12}$$

the increase in the model complexity of all other points, i.e., for a set Z, to capture the externality

introduced by x, where $T' = T(Z - x)$. To avoid outlier masking (phenomenon in which dense outliers mask each other) instead of removing just x, they propose to remove a set C with $x \in C$ and obtain so the *Collusive Displacement* $CoDisp(x, Z, S)$

$$\mathop{\mathbb{E}}_{S \subseteq Z, T} \left[\max_{x \in C \subseteq S} \frac{1}{|C|} \sum_{p \in S - C} (f(p, S, T) - f(p, S - C, T'')) \right] \tag{2.13}$$

Anomalies correspond to large $CoDisp$ values: similarly to LOF, we properly tune the decision function by observing the scores of the top-most abnormal samples (contamination) in the dataset.

2.8 Conclusion

In this chapter, we presented a detailed taxonomy and an overview of the techniques used in unsupervised anomaly detection. We distinguish between *batch* and *time series stream* algorithms. The former process the data, usually collected over a period of time, at once and output the anomaly scores at the end. The latter have to process each instance as soon as streamed and output its anomaly score shortly after. From the next chapter we present i) a novel *batch* algorithm, ii) a *time series stream* clustering engine based on DenStream and finally iii) an automated algorithm for anomaly detection using *batch* algorithms.

Chapter 3

Random Histogram Forest for Unsupervised Outlier Detection

In this chapter we describe the proposed *batch* unsupervised algorithm called *Random Histogram Forest (RHF)*. We first introduce the main ideas in Section 3.1 while analyze the most recent comparative studies of unsupervised algorithms in Section 3.2. We then describe in details the algorithms in Section 3.3 and perform several experimental evaluations in Section 3.4. Finally, a summary of our finding and remarks are given in Section 3.5.

3.1 Introduction

We present *Random Histogram Forest* (RHF) [118] an effective tree-based approach for unsupervised anomaly detection. In our approach, for every node in the tree, the splitting feature is chosen with a probability proportional to its fourth central moment (aka *kurtosis*). This allows to focus on the most informative features, while being resilient to the presence of "noisy" features. RHF, then, computes a score for every point, measuring its likelihood of being an anomaly, with larger anomaly scores being assigned to points lying in less populated leaves. More precisely, such a score is defined as the *Shannon's information content* [132, 104] of the leaf containing the corresponding point. Moreover, RHF leverages the predictive power of multiple independent trees (aka *a forest*) for identifying anomalies even more effectively. Some of these ideas have been successfully employed in the most successful algorithms for unsupervised anomaly detection (e.g. Random Forest [91], Isolation Forest [92], XSTREAM [97], Rs-Hash [130]), as well as

in a wide range of machine learning tasks.

Previous studies [44, 50] show that one of the most effective algorithms for unsupervised anomaly detection is Isolation Forest (*iForest*), as partially confirmed also by our experimental evaluation. However, our experiments show *XSTREAM* to be a more robust choice when applied to unseen data. In our work, we present *Random Histogram Forest* (RHF) [118] an effective approach for unsupervised anomaly detection. Similarly to *iForest*, our approach is probabilistic while relying on an "ensemble" of "weak" building blocks (trees) for effectively identifying anomalies. This has been proved to be effective for a wide range of tasks (e.g. Random Forest [91], Isolation Forest [92], XSTREAM [97]).

Our approach builds a random forest based on all input instances, whereas *iForest* builds a random forest based solely on some random samples of the data. The latter strategy has the drawback that some anomalies might be neglected entirely in the construction of the forest (particularly noticeable in large datasets containing only a few anomalies), thereby impairing the capability of the algorithm of finding those anomalies. Nevertheless, our algorithm boasts linear running time in the size of the input. Another key idea of our algorithm is to employ the fourth central moment (aka *kurtosis*), so as to guide the search for anomalous instances in the dimensions that are most likely to contain them. Notice that authors in *iForest*, also suggest to use *kurtosis* as an attribute selector in case of high dimensional data. Authors suggest to rank each attribute and select them according to their rankings. However, such selection phase implies adding to the algorithm another hyperparameter (number of top attributes to select) which might be difficult to set. Moreover, selecting only attributes with hight kurtosis scores might neglect anomalies separable only in low kurtosis features. We propose an attribute selector without adding hyperparameters. Furthermore, our approach enables the selection of low kurtosis features at higher depths of the trees. Finally, our algorithm computes a score for every instance, measuring the likelihood of such an instance of being an anomaly. Such a score is defined as the *Shannon's information content* [132, 104] of the leaf containing the corresponding instance.

We conduct an extensive experimental evaluation on 38 publicly available datasets including all benchmarks for anomaly detection and 64 private datasets from Huawei [106], as well as the most successful algorithms for unsupervised anomaly detection, to the best of our knowledge. We evaluate all the approaches in terms of average precision (AP), that is, the area under the precision-recall curve. We observe that as suggested in [39], ROC might not reflect the real performances of the algorithm, in that, anomalies typically represent a small fraction of the input

data.

Our experimental evaluation shows that RHF outperforms all other approaches in terms of AP. Moreover, it shows that our algorithm delivers consistently good results over a wide range of hyper-parameter values, which allows for an effective hyper-parameter tuning. Another appealing feature of our algorithm is that it boasts linear running time, which makes it an effective tool for processing large amounts of data. Finally, our experimental evaluation shows that both *RHF* and *XSTREAM* are more effective and robust to noise than *iForest*.

The rest of the chapter is organized as follows. We start by providing an overview of the state-of-the-art anomaly detection algorithms in Section 3.2, while we introduce Random Histogram Forest in Section 3.3. We then describe our experimental evaluation and results in Section 3.4. Finally, we discuss and summarize our main findings in section 3.5.

3.2 Related Work

Among the most recent comparative studies of unsupervised techniques, [62] compares most of the existing proximity-based methods on 10 different datasets and conclude that it is of great importance the initial assumption whether the anomalies in the datasets are global or local: they recommend to use a *global* anomaly detection methods if there is no further knowledge about the nature of anomalies in the dataset to be analyzed. [44] compares 14 methods belonging to all the groups previously described on 15 different datasets (12 publicly available and 3 private ones). However, their study assesses if the models are able to generalize to future points, so they perform a Monte Carlo cross validation of 5 iterations, using 80% of the data for the training phase and 20% for the prediction which indicates a semi-supervised setup to our understanding. While [62] study does not include the latest methods presented (e.g. Isolation Forest, thought to be the state-of-art), [44] describes the models generalization capacity using labels that most of the time are not available. We compare the methods which have proven to be the best in previous studies [44, 62] using default or reasonable hyperparameters and use labels only to assess their performance in a completely *unsupervised* environment.

Based on the most recent comparison studies [62, 44, 50], the algorithms previously described have proven to be among the best in regards *anomaly detection*. In general both [44] and [50] suggest *iForest* to be, on average, the best one closely followed [44] by *PPCA* and *OCSVM*.

3.3 Random Histogram Forest (RHF)

RHF is a tree ensemble model apt at randomly partitioning the input data, so as to highlight anomalies. We illustrate our main intuition in Figure 3.1, where red points represent anomalies and the remaining points are depicted in green. In such a figure, we randomly partition the input points by means of several lines drawn uniformly at random. The same process is iterated three times. Observe that input points that end up in a relatively large group are less likely to be anomalies. By iterating such a process multiple times, we can obtain for each point a score measuring how likely such a point is an anomaly.

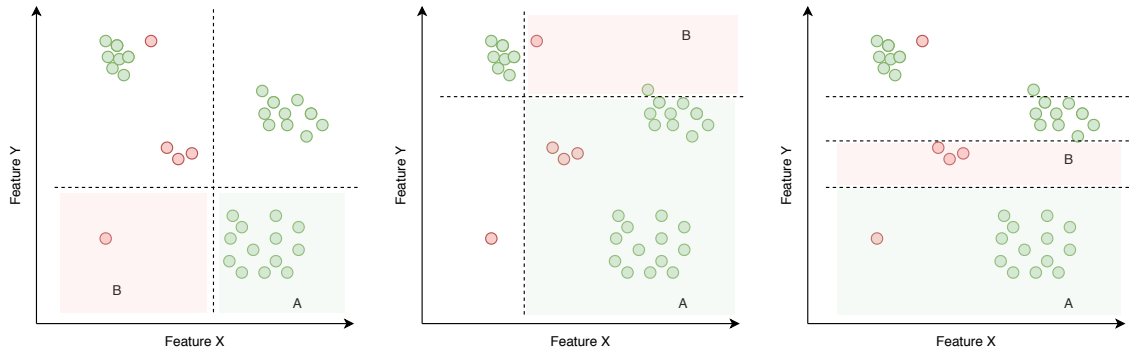


Figure 3.1: Example of 3 different Random Splits in 4 bins η. One can observe that some areas (e.g. *A*) have noticeable higher mass than others (e.g. *B*)

In real-world applications, it is often the case that a relatively large dimensions are non-informative when looking for anomalies. Splitting the dataset along non-informative or "noisy" dimensions might result in poor results and it should be avoided. To this end, we select the splitting feature according to their kurtosis value, which given a random variable $\mathcal{X}$ is defined as follows:

$$K[\mathcal{X}] = E\left[\left(\frac{\mathcal{X}-\mu}{\sigma}\right)^4\right] = \frac{E\left[(\mathcal{X}-\mu)^4\right]}{E\left[(\mathcal{X}-\mu)^2\right]^2} = \frac{\mu_4}{\sigma^4}, \tag{3.1}$$

where μ_4 is the fourth central moment and σ the standard deviation. The kurtosis of input data $\mathcal{X}$ denotes the standardized data raised to the fourth power, while it can be seen as a measure of the *tailedness* of $\mathcal{X}$. As a result, points within the region of the peak have negligible contribution to the kurtosis score, while points outside the region of the peak (e.g. *outliers*) contribute the most. In [103], Moors defined it as a measure of dispersion, while he concluded that high values of K are due to either i) occasional values far from the mean in a distribution whose probability mass is concentrated around the mean or ii) probability mass concentrated in the tails of the distribution.

The kurtosis score measures the heaviness of the tails and it is therefore an indicator of the presence of *outliers* in the tail. Consider, for example, Fig. 3.2 representing 4 features extracted from datasets *Annthyroid* (top) and *Mulcross* (bottom) depicting both *normal* and *anomalous* probability density functions. One can observe that features with heavier tails and consequently higher kurtosis score (e.g. X_1-top and X_2/X_3-bottom) are more likely to contain anomalous points than the remaining ones (e.g. X_0/X_4-top and X_0/X_1-bottom).

Roughly speaking, when selecting the splitting feature, our algorithm will compute the kurtosis score κ for each feature and then sample one feature with probability proportional to $\log(\kappa + 1)$. The logarithm allows us to focus on the order of magnitude of the kurtosis score, while preventing our approach from being sensitive to small values on such score. The additional term of 1 in the logarithm ensures that a feature is selected with zero probability, when the corresponding kurtosis is zero (i.e. the feature is constant). In what follows, we provide a more formal description of our algorithm.

Preliminaries. We are given input data $X \in \mathbb{R}^{n \times d}$, where n, d denote the number of points and *features* (aka *attributes*), respectively. We denote with $X_{i\cdot}$ the ith row of X, corresponding to an input point, while we denote with $X_{\cdot j}$ the j column of X, corresponding to feature f_j. We use x_i as a shorthand for $X_{i\cdot}$. Given $S \subseteq [n]$, we denote with $X_{S\cdot}$ the submatrix of X containing for every $i \in S$ the ith row of X, while we denote with $X_{S,j}$ the jth column of $X_{S\cdot}$. RHF builds a forest containing several (connected) binary trees, i.e. each node in a tree has either two children or no children (in the latter case it is called a *leaf*). Each leaf v in a tree is associated with a set $S_v \subseteq [n]$, specifying the set of points associated to v. Every other node v in a tree, is associated with a tuple $(S_v, f_v, \lambda_v, l_v, r_v)$, where $S_v \subseteq [n]$ is the set of points associated to v, f_v is the splitting feature for v, λ_v is its splitting value, while l_v, r_v denote its left and right child nodes, respectively. For every node v in a tree we have the following invariants: 1) every point x_i, $i \in S_v$, satisfies either the constraint $X_{ij} < \lambda_v$ (if v is a left child) or $X_{ij} \geq \lambda_v$ (if v is a right child), where $f_v = f_j$; 2) $S_v = S_{l_v} \cup S_{r_v}$; 3) $S_{l_v} \cap S_{r_v} = \emptyset$. For every tree T and every point i, RHF computes a score w_i^T, while the overall score w_i for point i is obtained by computing the average score over all trees. Observe, that the number of leaves in a tree is at most 2^h, where h is its height.

Tree construction. RHF builds each tree in the following way. It first creates a root node v, which is set to be the current node with $S_v := [n]$. Then, it selects one feature f_j with probability

$$\frac{\log\left(K(X_{S_v,j})+1\right)}{\sum_{j=1}^{d}\log\left(K(X_{S_v,j})\right)+1)}. \tag{3.2}$$

After that, it selects a splitting value λ, which is taken uniformly at random from the interval $(\min(X_{S_v,j}), \max(X_{S_v,j}))$. Then, two child nodes of the current node are created: one left node associated with all points in S_r satisfying $f_j < \lambda$ and another node with the remaining points. This is executed recursively on the left and right node until one of the following conditions is met:

i the maximum height h is attained;

ii the kurtosis of every feature in the current set of points is zero.

When the construction of a tree is built, every node with no children is labeled as a leaf. The pseudocode of the algorithm to construct a tree is provided in Algorithm 1, which is initially called with $RHF(X, [n], r, 1, h)$ with r being the root node.

Anomaly scores. Given a tree T, we compute the anomaly score for every point x_i as follows. Let ℓ be a leaf in T. For every leaf ℓ, for every point $i \in S_\ell$ define:

$$w_i^T := \log\left(\frac{1}{|S_\ell|}\right)$$

to be the anomaly score of point x_i w.r.t. tree T. w_i^T can be seen as the *Information Content* (also called Shannon's information [133]) of x_i in T. The information content measures the level of "surprise" of an event, with rare events being more surprising than relatively common events. As a result, the smaller the cardinality of S_ℓ the more likely x_i is considered to be an outlier. We adopt the convention that $-log(0) = +\infty$. The overall anomaly score w_i of x_i is obtained by computing the average anomaly scores over all trees in RHF i.e. $w_i = \frac{1}{t}\sum_T w_i^T$.

Implementation Details. In order to sample a feature according to Equation 3.2 we proceed as follows. We first compute the sum of all kurtosis scores:

$$\kappa = \sum_{j=1}^{d}\log\left(K(X_{S_v,j})+1\right)$$

we then sample uniformly at random a value r from $[0, \kappa]$. Finally, we compute:

$$j = \arg\min \left(j | \sum_{l=1}^{j} \log \left(K(X_{S_v,j}) + 1 \right) > r \right).$$

Scalability Analysis. Overall the running complexity of our approach is $\mathcal{O}(ntdh)$, where h denotes the maximum height (input hyper-parameter), t denotes the number of trees (input hyper-parameter), n the number of points while *d* denotes the number of dimensions. The space complexity of the algorithm is instead $\mathcal{O}(nd)$ which is linear in the input data size. Notice that there is not the need to keep in memory all the trees. It is sufficient to split all the data according to a tree, collect the scores from the leafs, and discard it before generating another tree.

Algorithm 1: RHF(X, S, v, ℓ, h)

Input: dataset *X*, $S \subseteq [n]$, set of nodes V, node height ℓ, max height *h*
Output: a binary tree T with root v

$S_v := S$
if $\ell \geq h$ or $K(X_{S,j}) == 0 \; \forall j \in [1, d]$ **then**
  label v as a leaf with $S_v := S$
else
  sample one feature f_j with probability $\frac{\log(K(X_{S,j})+1)}{\sum_{j=1}^{d} \log(K(X_{S,j}))+1)}$
  select a split value λ uniformly at random from $(\min(X_{S,j}), \max(X_{S,j}))$
  $f_v := f_j$, $\lambda_v := \lambda$
  create new nodes l, r
  $l_v := l$, $r_v := r$
  $S_l := \{i \in S : X_{ij} < \lambda\}$
  $S_r := S \setminus S_l$
  RHF($X, S_l, l, \ell + 1, h$)
  RHF($X, S_r, r, \ell + 1, h$)
end if

3.4 Experimental Evaluation

3.4.1 Settings.

Libraries: Our experimental evaluation is conducted on a Linux Fedora 31 server equipped with Intel(R) Xeon(R) CPU E5-2665 @ 2.40GHz - 32 CPUs and 48 GB RAM. Our code is written in *Python 3/Cython* [145, 23] while it uses $NumPy == 1.17.4$ *[109]* and $Pandas == 0.25.3$ *[99]* for data preprocessing. The implementations of the algorithms described in Section 3.2 belong to either $PyOD == 0.7.9$ [156] (HBOS, PPCA, K-NN, OCSVM) or $Scikit-learn == 0.23.1$ [113]

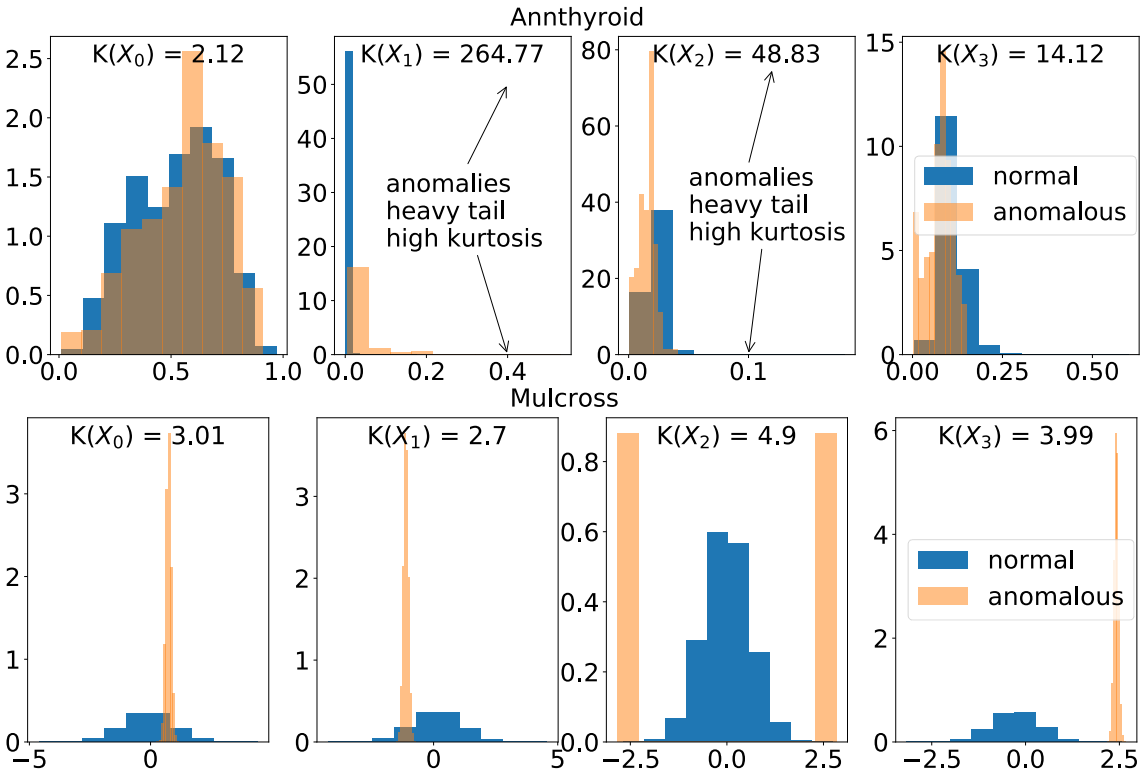


Figure 3.2: Probability Density Function of 4 features depicting both *normal* and *anomalous* class extracted from datasets *Annthyroid* and *Mulcross* respectively. It is easily observable that features with heavier tails (depicted by arrows) and consequently higher *kurtosis* score (e.g. X_1-top and X_2/X_3-bottom) are more likely to contain separable *anomalous* points than remaining ones (e.g. X_0/X_4-top and X_0/X_1-bottom in which *anomalies* are clearly not separable).

(iForest and LOF) packages. We use moreover the python code released by the authors for *PID* [12] and the c++ one for *XSTREAM* [13]. Our *RHF*'s *Python 3* (Cython) implementation is available at [116].

HyperParameters: For each approach considered in our experimental evaluation, we set its hyperparameters while following the directions of the corresponding authors. In particular, we run *HBOS* selecting the input hyper-parameter *number of bins* using the rule of thumb $K = \sqrt{n}$ as suggested by the authors. Similarly, we set $n\_components = mle$ and $svd\_solver = full$ which finds the best number of PPCA's components. We run the proximity based methods *K-NN* and *LOF* using K=50 neighbors. As already successfully done in [44], we use *OCSVM*'s default hyper-parameters *kernel=rbf*, *degree=3* with regularization hyper-parameter $\nu = 0.5$. All the ensemble methods (ISO, PID, XSTREAM, RHF) use an equal number of models $t = 100$. Moreover we use the recommended sample size $\psi = 256$ and $hlim = log(\psi)$ for *ISO*, the default sample size $\psi = 100$, max degree $k = 3$ and max depth $h = 10$ for *PID* and the recommended number of projections dimensions $k = 100$ and depth $l = 15$ for *XSTREAM*. Finally *RHF* uses max height *h = 5* corresponding to at most 32 *leafs/bins*.

Metrics: We evaluate the performance of the algorithms by their *Average Precision (AP)*, that is, the area under the curve of the precision-recall curve [159] (without interpolation):

$$AP := \sum_n (R_n - R_{n-1})\, P_n$$

where $P_n = \frac{tp}{tp+fp}$ and $R_n = \frac{tp}{tp+fn}$ are the Precision and Recall at the n^{th} threshold, respectively. We observe that the Receiver Operating Characteristic (ROC) is often employed to evaluate anomaly detection methods [62] -[92]. However, it has been shown in [39] that when the classes are not balanced (which is typical of an anomaly detection task) the *AP* curves better reflect the efficacy of an algorithm. [39] shows moreover that a curve dominates in ROC if and only if it dominates in AP space.

Datasets: We put a major effort in providing an extensive experimental evaluation. In particular, we include all datasets that have been considered in the literature for anomaly detection, to the best of our knowledge. This is crucial to ensure a fair comparison, in that, the overall results might change dramatically depending on the selection of the datasets, as pointed out in [50]. We

use 38 publicly available benchmark datasets ranging from 240 to 623091 points and from 3 to 274 dimensions. Each of them is available either at the UCI [47] or at the ODDS [126] repositories. We furthermore consider the recently released *WikiQOE* [128] dataset, which consists of a Wikipedia large measurements campaign of WebQOE metrics.

The KDD'99 Cup dataset is one of the most widely used benchmark for anomaly detection. The dataset contains information about network connections as exchanged bytes ("source bytes", "destination bytes", etc.) and service type ("http", "smtp", "ftp", etc.). It consists of 4,898,431 points and 41 attributes. Similarly to the filtering technique used by [153] [62] we extract 5 subsets according to the values of the *service* attribute (*http*, *smtp*, *ftp*, *finger* and *other*). Out of the 41 available attributes, we select, as already done in [153] only 3 of them namely "duration", "source bytes", and "destination bytes" as they are thought to be the most relevant ones [153]. We obtain in this way the datasets we call *kdd_http* (623091 points), *kdd_smtp* (95554 points), *kdd_ftp* (5214 points), *kdd_finger* (1033 points) and *kdd_other* (12844 points). While [62] filtered the dataset according to the *service* attribute only, [153] filters them also by the positive *logged_in* attribute as they are successful attacks. We also consider this additional filter by further reducing the *kdd_http* dataset into the *http_logged* (567498 points) one by excluding the negative values of *logged_in* attribute.

In order to determine to which extent the presence of duplicates might affect the overall results, we consider also a smaller version in which duplicates have been filtered out: we will refer to them as *kdd_http_distinct*, *kdd_smtp_distinct* and *kdd_ftp_distinct*. We include in our comparison also the full version *kdd_http29* and *kdd_smtp29* in which all the 29 continuous attributes are used. The same 29 features are used also by [62] in which the authors use only the relevant anomalies (by limiting the number of duplicated ones) and present a new dataset (composed by 620098 points 0.17% *anomalous* points). We will refer to it as *kdd99G* by author's name.

Additionally, we consider a set of 64 private datasets (collected and stored at Huawei) presented in this comparison [106]. They are collected over a lifespan of 125 days from real routers and labeled by network experts. Each dataset is collected locally from real routers of a different Internet Service Provider so datasets are completely independent. Overall datasets are composed by 211 up to 11700 points out of which 4.4% are anomalous and from 6 up to 1650 dimensions.

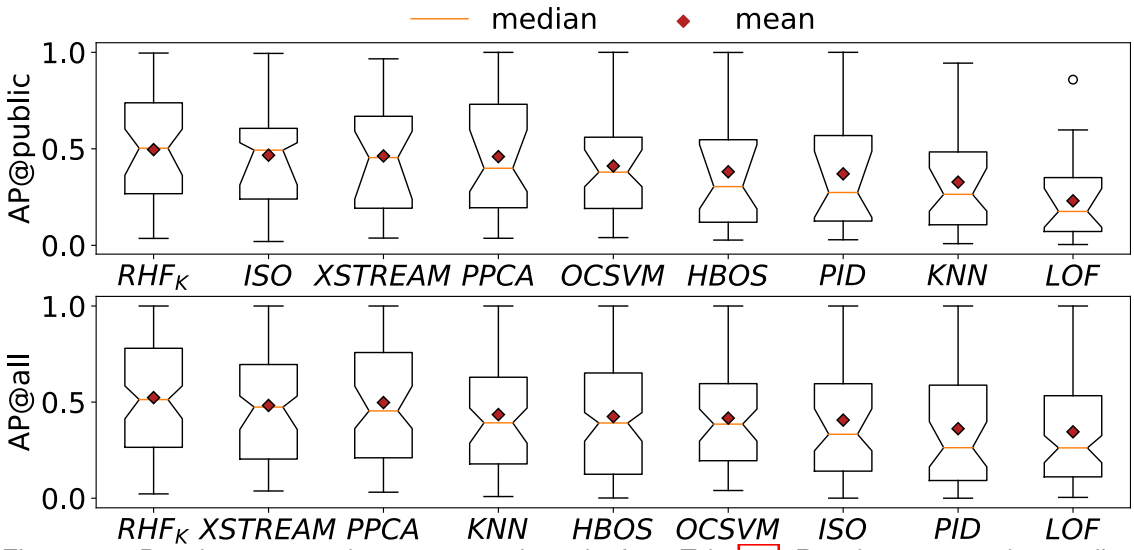


Figure 3.3: Boxplot representing aggregated results from Tab. 5.3. Results are sorted according to the median value in decreasing order complemented with a bootstrapped 0.95 confidence interval. In both *public* and *private* datasets RHF_K achieves the highest *q25, mean, median* and *q75* values.

3.4.2 Comparison

We evaluate all the algorithms discussed in Section 3.2, which have proven to be most effective according to previous experimental evaluations [62], [44], [50]. We also consider two variants of *RHF*: one variant where *Kurtosis Split* is used (RHF_K), and one where random splits are used (RHF_R).

Table 5.3 reports a full comparison of all approaches and datasets considered in our paper. The last four rows in the table are the mean and median computed on public vs all (public and private) datasets complemented with 0.95 boostrapped conficende intervals. Text in bold represents the best method for each dataset. The best method is awarded using the two sample Kolmogorov-Smirnov test ($\alpha = 0.05$) [136] under the null hypothesis that the two distributions are identical while the Welch's two-tailed t-test [149] is used to test if they have the same mean. Two or more algorithms are declared the best performing ones, when the two tests cannot reject the null hypothesis.

Our results on the public datasets (AP - best on #number of datasets) confirm the results provided in [44]. In particular, *ISO* (0.462 ± 0.097 - #7) and *PPCA* (0.459 ± 0.010 - #5) are indeed effective algorithms for *anomaly detection*. To the latter we add also *XSTREAM* (0.462 ± 0.091 *#6*) and both RHF_K (0.496 ± 0.097 - #6) and RHF_R (0.479 ± 0.096) which outperform the other

approaches.

Figure 3.3 shows by means of boxplots a summary of the overall results on all the datasets in both *public* and *private* scenarios. The boxplots are complemented with a 0.95 median confidence level estimated through bootstrapping. The methods are sorted according to the median value in decreasing order. Our method RHF_K presents in both the scenarios the best *q25, mean, median* and *q75* values proving to be consistently better than remaining methods.

The largest discrepancies between the aforementioned algorithms have to be found in some specific datasets. This is the case of the *kdd_distinct* dataset composed by 220027 points and only 75 anomalies in which *ISO* produces the baseline result (0.02) while *PPCA* (0.637), *XSTREAM* (0.669) and RHF_K (0.757) produce a much higher AP. Unsatisfactory results on the same dataset are produced also by *PID* which builds the trees, as *ISO*, on sampled data. We observe that algorithms which build their model on a relatively small sample of the dataset might deliver poor results, in that, they might miss important information required to retrieve the anomalies. Similarly, in *http29* the *ISO* approach is again the worst performing one (0.532) with respect to *PPCA* (0.758), *XSTREAM* (0.933) and RHF_K (0.709), which are methods that use all points in the dataset to generate their models.

Other discrepancies have to be found in datasets such as *musk, thyroid* or *kdd_other* in which *XSTREAM* produces slightly worse results (0.651, 0.192 and 0.099) with respect to the other methods (e.g. RHF_K produces respectively 0.996, 0.516 and 0.543). Similarly, *PPCA*'s AP on the *smtp29* (0.774 vs RHF's 0.95), *penglobal* (0.301 vs RHF's 0.561) and *thyroid* (0.362 vs 0.516) show this method to be a poor choice on some datasets.

Considering instead all the available datasets results change significantly. While RHF_K (0.513 ± 0.080 - #26) remains the best method followed by *XSTREAM* (0.462 ± 0.080 - #17) and *PPCA* (0.454 ± 0.080 - #16), ISO (0.333 ± 0.100 - #16) appears to be less reliable on such datasets. Further studies (see. Section 3.4.3-3.4.4) show that *ISO*'s performance deteriorates when the datasets contain irrelevant features. Moreover it is unclear how to properly select its hyper-parameter ψ as small perturbations of the dataset are sufficient to change significantly the output.

As also stated in [50], we observe that many approaches achieve similar performances in most of the datasets, with the most prominent differences being found in a few of them. Overall, RHF_K, *ISO* and *XSTREAM* are the algorithms that better perform in our benchmark datasets. A better overall picture of the three algorithms can be derived from our further studies in which the methods undergo robustness to noisy dimensions and hyperparameters tests.

	n	d	anomalous - duplicates	LOF	KNN	PID	HBOS	OCSVM	PPCA	XSTREAM	ISO	RHF_K	RHF_R
musk	3062	166	3.2% - 0.0%	0.164±0.0	0.592±0.0	**0.999±0.001**	0.999±0.0	**1.0±0.0**	1.0±0.0	0.382±0.036	0.994±0.003	0.996±0.003	0.988±0.006
smtp29	96554	29	1.2% - 2.1%	0.014±0.0	0.106±0.0	0.899±0.041	0.985±0.0	0.399±0.0	0.774±0.0	0.917±0.008	**0.988±0.001**	0.95±0.008	0.968±0.003
shuttle	49097	9	7.2% - 0.0%	0.122±0.0	0.191±0.0	0.737±0.021	0.948±0.0	0.907±0.0	0.915±0.0	0.888±0.008	**0.974±0.003**	0.932±0.003	0.948±0.002
breastcancer	683	9	35.0% - 1.3%	0.316±0.0	0.944±0.0	0.931±0.003	0.916±0.0	0.918±0.0	0.958±0.0	0.941±0.002	**0.967±0.002**	0.965±0.002	0.978±0.002
http_logged	567498	3	0.4% - 96.7%	0.037±0.0	0.009±0.0	0.454±0.08	0.318±0.0	0.492±0.0	0.769±0.0	**0.965±0.001**	0.924±0.028	**0.965±0.002**	0.976±0.002
satimages	5803	36	1.2% - 2.8%	0.021±0.0	0.623±0.0	0.75±0.054	0.755±0.0	**0.965±0.0**	0.872±0.0	0.9±0.01	0.923±0.004	0.927±0.004	0.914±0.007
ionosphere	351	33	35.9% - 0.8%	0.859±0.0	**0.879±0.0**	0.741±0.007	0.279±0.0	0.839±0.0	0.747±0.0	0.801±0.003	0.809±0.002	0.811±0.003	0.793±0.004
magicgamma	19020	10	35.2% - 1.7%	0.565±0.0	**0.729±0.0**	0.645±0.004	0.631±0.0	0.626±0.0	0.586±0.0	0.65±0.004	0.641±0.004	0.625±0.006	0.622±0.004
satellite	5100	36	1.5% - 0.0%	0.359±0.0	0.629±0.0	**0.63±0.011**	0.542±0.0	0.622±0.0	0.583±0.0	0.623±0.003	**0.64±0.006**	**0.647±0.007**	0.637±0.007
penglobal	809	16	11.1% - 0.0%	0.598±0.0	**0.873±0.0**	0.418±0.016	0.237±0.0	0.569±0.0	0.301±0.0	0.621±0.01	0.61±0.011	0.561±0.013	0.562±0.013
wbc	378	30	5.6% - 4.8%	0.591±0.0	0.557±0.0	0.661±0.014	**0.69±0.0**	0.529±0.0	0.556±0.0	0.516±0.007	0.596±0.009	0.555±0.012	0.599±0.012
cardio	1831	21	9.6% - 0.6%	0.193±0.0	0.392±0.0	0.444±0.018	0.416±0.0	0.533±0.0	**0.612±0.0**	0.531±0.008	0.563±0.013	0.581±0.013	0.567±0.014
http29	623091	29	0.6% - 31.3%	0.013±0.0	0.108±0.0	0.572±0.029	0.522±0.0	0.499±0.0	0.758±0.0	**0.933±0.006**	0.532±0.01	0.709±0.047	0.514±0.016
kdd99	620098	29	0.2% - 1.3%	0.004±0.0	0.196±0.0	0.526±0.018	0.529±0.0	0.325±0.0	0.683±0.0	**0.886±0.005**	0.531±0.003	0.771±0.028	0.583±0.019
mulcross	262144	4	10.0% - 0.0%	0.17±0.0	0.052±0.0	0.14±0.004	0.064±0.0	0.643±0.0	**0.979±0.0**	0.548±0.012	0.529±0.026	0.748±0.014	0.734±0.019
thyroid	3772	6	2.5% - 0.0%	0.186±0.0	0.379±0.0	0.281±0.019	**0.776±0.0**	0.318±0.0	0.362±0.0	0.192±0.008	0.521±0.025	0.516±0.015	0.357±0.019
kdd_other	12844	3	3.7% - 0.0%	0.069±0.0	0.089±0.0	0.029±0.001	0.09±0.0	0.092±0.0	0.052±0.0	0.099±0.003	0.509±0.008	**0.543±0.026**	0.453±0.021
pima	768	8	34.9% - 0.0%	0.451±0.0	0.523±0.0	0.483±0.003	**0.531±0.0**	0.464±0.0	0.454±0.0	0.492±0.003	0.506±0.004	0.49±0.004	0.496±0.004
kdd_http	623091	3	0.6% - 98.1%	0.026±0.0	0.01±0.0	0.223±0.05	0.227±0.0	0.369±0.0	0.55±0.0	**0.549±0.004**	0.504±0.025	**0.548±0.004**	0.56±0.003
spambase	4601	57	39.4% - 7.4%	0.323±0.0	0.415±0.0	0.488±0.009	**0.549±0.0**	0.399±0.0	0.404±0.0	0.445±0.007	0.482±0.009	0.413±0.009	0.417±0.012
arrhytmia	452	274	14.6% - 0.0%	0.368±0.0	0.392±0.0	**0.461±0.009**	0.438±0.0	0.391±0.0	0.395±0.0	0.435±0.004	**0.467±0.008**	0.429±0.011	0.454±0.009
abalone	1920	7	1.5% - 0.0%	0.416±0.0	0.289±0.0	0.13±0.01	0.215±0.0	0.308±0.0	0.298±0.0	0.2±0.008	**0.463±0.029**	0.369±0.014	0.362±0.016
kdd_ftp	5214	3	26.7% - 79.8%	0.295±0.0	0.259±0.0	0.439±0.036	0.392±0.0	0.596±0.0	**0.846±0.0**	0.429±0.018	0.423±0.007	0.452±0.015	0.403±0.002
kdd_ftp_distinct	2876	3	9.8% - 0.0%	0.136±0.0	0.27±0.0	0.251±0.012	0.306±0.0	0.19±0.0	0.259±0.0	0.318±0.01	0.389±0.006	**0.408±0.013**	0.354±0.008
annthyroid	7200	6	7.4% - 0.0%	0.185±0.0	0.226±0.0	0.271±0.008	**0.558±0.0**	0.186±0.0	0.19±0.0	0.15±0.004	0.313±0.009	0.313±0.012	0.223±0.008
mnist	7603	100	9.2% - 0.0%	0.326±0.0	**0.409±0.0**	0.135±0.014	0.079±0.0	0.386±0.0	0.383±0.0	0.339±0.009	0.27±0.012	0.346±0.017	0.303±0.018
kdd_smtp	96554	3	1.2% - 97.9%	0.028±0.0	0.015±0.0	0.107±0.008	0.287±0.0	0.323±0.0	**0.353±0.0**	0.273±0.068	0.247±0.005	0.275±0.007	0.288±0.005
kdd_finger	1033	3	2.4% - 0.0%	**0.558±0.0**	0.345±0.0	0.091±0.006	0.303±0.0	0.299±0.0	0.191±0.0	0.35±0.016	0.238±0.009	0.265±0.008	0.367±0.011
yeast	1191	8	4.6% - 0.0%	**0.263±0.0**	0.249±0.0	0.125±0.008	0.125±0.0	0.202±0.0	0.241±0.0	0.216±0.005	0.217±0.006	0.232±0.008	0.226±0.008
mammography	11183	6	2.3% - 2.3%	0.13±0.0	0.176±0.0	0.248±0.023	0.086±0.0	0.183±0.0	0.205±0.0	**0.231±0.005**	0.215±0.013	0.158±0.007	0.164±0.009
wikiqoe	55932	17	8.0% - 0.0%	0.093±0.0	0.134±0.0	0.163±0.001	0.117±0.0	0.122±0.0	0.12±0.0	0.085±0.003	**0.168±0.0**	0.119±0.004	0.124±0.005
vowels	1456	12	3.4% - 8.0%	0.403±0.0	**0.493±0.0**	0.141±0.012	0.118±0.0	0.195±0.0	0.068±0.0	0.143±0.006	0.141±0.014	0.12±0.012	0.136±0.014
vertebral	240	6	12.5% - 0.0%	0.09±0.0	0.089±0.0	0.084±0.0	0.084±0.0	0.103±0.0	**0.104±0.0**	0.095±0.001	0.095±0.001	0.096±0.001	0.093±0.002
cover	286048	10	1.0% - 0.0%	0.017±0.0	0.043±0.0	0.047±0.006	0.027±0.0	**0.096±0.0**	0.078±0.0	0.062±0.006	0.052±0.007	0.08±0.015	0.083±0.009
kdd_smtp_distinct	71257	3	0.0% - 0.0%	**0.181±0.0**	0.178±0.0	0.03±0.005	0.162±0.0	0.058±0.0	0.085±0.0	0.087±0.009	0.049±0.003	0.072±0.001	0.072±0.001
wine	4898	11	0.5% - 0.0%	0.083±0.0	0.081±0.0	0.043±0.009	0.033±0.0	0.065±0.0	0.062±0.0	**0.186±0.004**	0.045±0.004	0.066±0.002	0.079±0.005
aloi	50000	27	3.0% - 0.0%	**0.079±0.0**	0.051±0.0	0.032±0.0	0.027±0.0	0.04±0.0	0.037±0.0	0.055±0.001	0.033±0.0	0.036±0.0	0.04±0.001
kdd_http_distinct	222027	3	0.0% - 0.0%	0.033±0.0	0.456±0.0	0.096±0.051	0.125±0.0	0.373±0.0	0.637±0.0	0.669±0.003	0.02±0.002	**0.757±0.011**	0.793±0.007
mean@public	//	//	//	0.231	0.328	0.367	0.381	0.411	0.46	0.453	0.463	0.496	0.48
	//	//	//	(0.168;0.3)	(0.249;0.411)	(0.278;0.458)	(0.288;0.477)	(0.328;0.497)	(0.367;0.556)	(0.364;0.548)	(0.371;0.557)	(0.403;0.591)	(0.386;0.574)
median@public	//	//	//	0.176	0.264	0.276	0.304	0.379	0.399	0.432	0.493	0.503	0.454
	//	//	//	(0.093;0.306)	(0.178;0.401)	(0.141;0.472)	(0.189;0.522)	(0.303;0.496)	(0.279;0.598)	(0.273;0.548)	(0.291;0.532)	(0.358;0.598)	(0.356;0.575)
mean@all	//	//	//	0.346	0.435	0.360	0.425	0.417	0.498	0.479	0.405	**0.524**	0.495
	//	//	//	(0.289;0.404)	(0.377;0.495)	(0.299;0.425)	(0.361;0.490)	(0.368;0.468)	(0.438;0.559)	(0.422;0.537)	(0.343;0.468)	**(0.464;0.585)**	(0.434;0.558)
median@all	//	//	//	0.262	0.392	0.260	0.391	0.386	0.454	0.474	0.333	**0.513**	0.456
	//	//	//	(0.185;0.326)	(0.286;0.469)	(0.162;0.398)	(0.296;0.446)	(0.297;0.464)	(0.362;0.583)	(0.357;0.532)	(0.246;0.463)	**(0.413;0.585)**	(0.354;0.556)

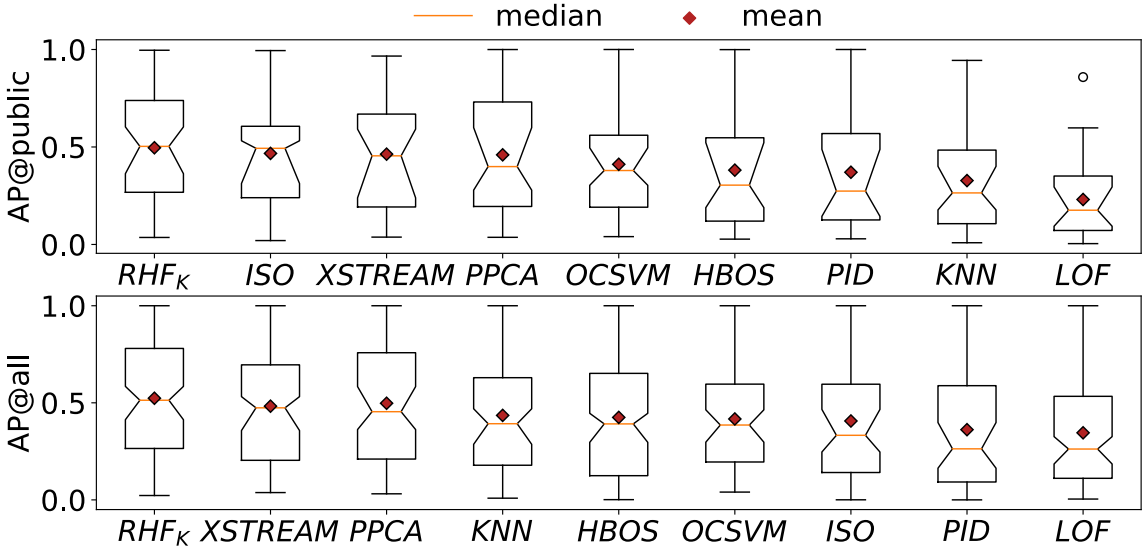


Table 3.1: AP scores of all approaches on all our datasets. The results are sorted in decreasing order of *ISO* scores. In the case of probabilistic approaches, each value is an average over 30 runs which is complemented with a 0.95 confidence interval. The best results for each dataset are represented in **bold** while the best between *ISO* and *RHF* is represented in **gray bold**.

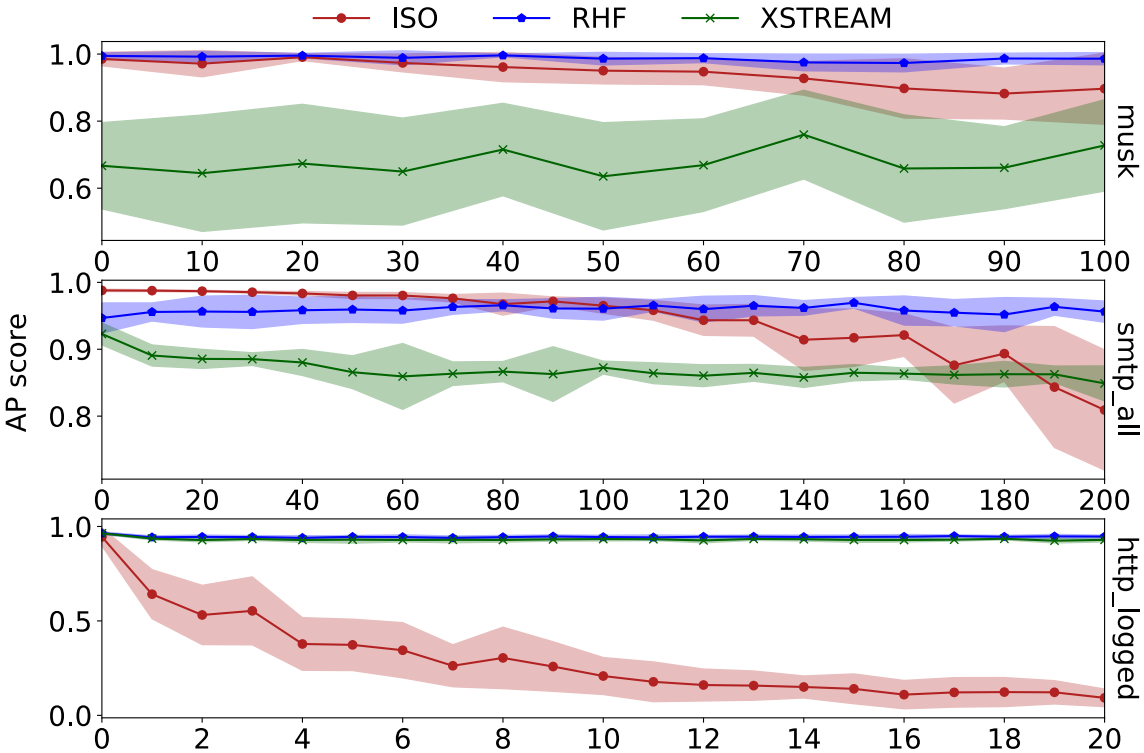


Figure 3.4: Robustness to noisy dimensions: an increasing number of gaussian dimensions are added to three datasets *musk* (top), *smtp_all* (middle) and http_logged (bottom). Steady results are produced by *RHF* and *XSTREAM* while ISO's perfomance is higly impacted

3.4.3 Robustness to noise

We investigate how the presence of noise in the datasets affect the effectiveness of the different algorithms. Our experimental evaluation shows large performance discrepancies between RHF_K, *XSTREAM* and *ISO* in most of our private high dimensional datasets. In particular, while RHF_K and *XSTREAM* are resilient to the presence of noisy features, a prevalence of those features might significantly compromise the results of *ISO*. To illustrate this, we artificially introduce noisy features, following a Gaussian distribution, to three datasets: (*http_logged*, *musk* and *smtp29*). Such features are clearly non-informative when looking for anomalies in the datasets, while ideally they should not disturb too much an anomaly detection algorithm.

Fig. 3.4 shows the results for the three algorithms on *musk* (top), *smtp_all* (middle) and http_logged (bottom) datasets. The plot shows that the presence of noisy features has a negative impact on the *ISO* performance, with a larger number of those features significantly affecting its effectiveness. In particular, for the *http_logged* dataset, *ISO's* AP score decreases from AP=0.98 to AP=0.60 after adding one single noisy feature.

On the other hand, *RHF* and *XSTREAM* deliver consistently relative good results. This is

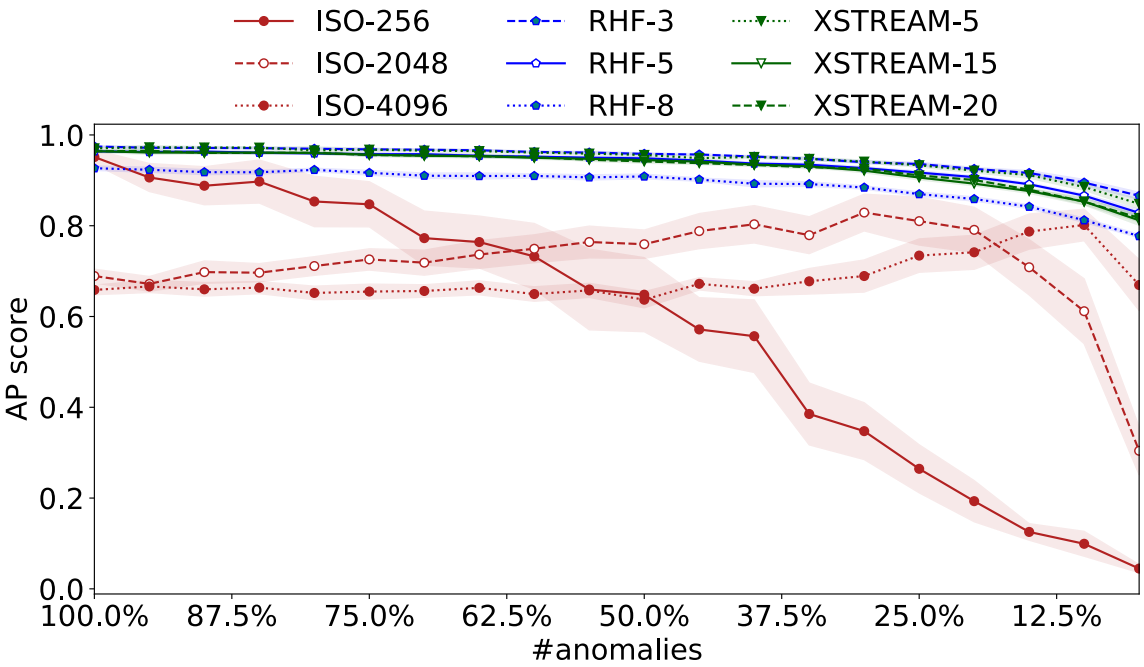


Figure 3.5: Average Precision for different number of anomalies in dataset *http_logged*. As *http_logged* contains 97% duplicated *anomalous* points we gradually remove them. The figure contrasts different Isolation Forest's sampling size $\psi \in [128, 4096]$ against *RHF*'s max height $h \in [4, 5]$. The plot illustrates that *iForest*'s performance is sensible to the hyper-parameter ψ when varying the number of *anomalous* points in the dataset.

perhaps not surprising, given that *RHF* chooses features with a probability proportional to their Kurtosis scores, as opposed to *ISO* where a uniform probability distribution guides its random choices. *XSTREAM*, which is based on random projections, appears to be resilient to noisy features, as well.

3.4.4 Robustness to the choice of hyperparameters

We further investigate the impact of the input hyperparameters on the overall results. We proceed in two main directions: i) we decrease artificially the number of anomalies in the input data; ii) we test several input hyperparameters for the three algorithms: RHF_K's $h \in [3, 5, 7]$, ISO's $\psi \in [256, 2048, 4096]$ and $XSTREAM$'s $l \in [5, 15, 20]$.

Figure 3.5 shows the AP of the methods on the *http_logged* dataset, as a function of the number of anomalies in the input dataset. Each value is the average over 30 runs being complemented with its corresponding confidence interval. The number of normal points is kept constant (we use all the 565287 points present in the original dataset) while the number of anomalies is reduced from 2211 (100% - left side of the plot representing the original dataset) to 100 (0.05%).

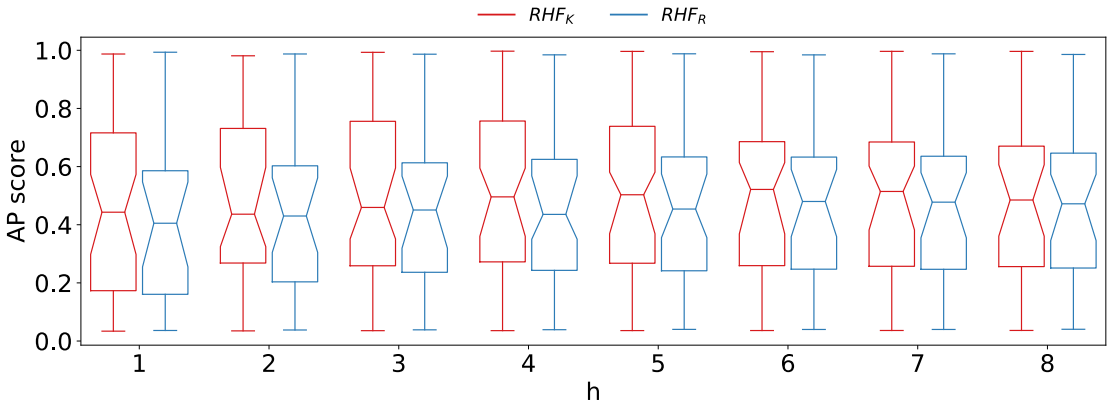


Figure 3.6: HyperParameters tuning: Average Precision score over all the datasets for increasing maximum tree height *h*. The figure illustrates scores obtained using both *Kurtosis Split* and *Random Split* criterion.

Results show again two distinct patterns for *RHF* and *XSTREAM* with respect to *ISO*. While the first two boast a slight performance decrease over the decreasing number of anomalies in the dataset (for all their hyperparameters), *ISO*'s default hyper-parameter show a faster deterioration, as it becomes increasingly more likely those anomalies to be missed in the sample. To tackle such an issue, one could try to increase the ψ hyper-parameter. This would result in a better AP score with $\psi = 2048$ and $\psi = 4096$, when the number of anomalies is smaller than 50% (right side of the plot). However, using a higher ψ value turns out to be counterproductive when the number of anomalous points is larger (left side of the plot, e.g. AP≈ 0.7 on the original dataset - 100% of the anomalies - for both $\psi = 2048/4096$). As a result, it is unclear how to determine an optimal value for such a hyper-parameter.

Overall, our studies show RHF_K and *XSTREAM* to be two effective algorithms for the anomaly detection task as they achieve, on average, high AP scores on both public and private datasets. Both of them cope well with noisy features, while deliver consistently good results under a wide range of values for their hyperparameters.

3.4.5 HyperParameters Tuning

RHF uses two input hyperparameters: the max height *h* which determines the η number of *leafs* and the number of trees *t*. As in most of the ensemble methods, we use *t = 100* trees while empirically study the behavior of *h*. Our method is somehow linked to histograms, as we split the data into η leaves. Therefore, we employ the widely used rules of thumb in determining the number of leaves (i.e. bins in histograms). In particular Sturge's [138] rule of thumb $k = \lceil 1 + log_2 n \rceil$

suggests that the number of bins should increase logarithmically in the number of points. We study *RHF*'s performance for increasing $h \in [1, 8]$ which defines $\eta \in [2, 256]$ comparing the results obtained using both *Random Split* and *Kurtosis Split*. The results depicted in Fig. 3.6 show two takeaways: i) *RHF*'s performance smoothly vary over *h* and ii) on average, *Kurtosis Split* consistently outperforms the *Random Split*.

Regarding the maximum height *h* hyper-parameter, we observe from Fig. 3.6 that: i) the AP benefits from increasing *h*, however, ii) the AP reaches its maximum value already when *h = 4*; iii) the number of leaves $\eta = 16/32$ defined by *h=4/5* is consistent with Sturge's rule of thumb which would recommend to use $K = 14$ bins for smaller datasets (e.g. 5000 points) while $K = 21$ for bigger ones (e.g. 620000 points). We set and recommend thus *h = 5*, as it produces results not distinguishable from *h = 4* and should handle properly also larger datasets.

Fig. 3.7 shows the performance of the algorithms when increasing the number of trees t in the forest. As expected, the results show relatively high variance for smaller values of t while they stabilize when $t > 50$. We suggest $t = 100$ as commonly used in ensemble models.

We finally measure the empirically the running complexity of RHF_K for increasing *n*, *d*, *t* and *h*. Table 3.2 shows such times for all the datasets in our comparison and for increasing values of $h \in [3, 5, 7]$ (using $t = 100$) and $t \in [30, 60, 90]$ (using $h = 5$). The table contains moreover the running times of *ISO* ($\psi = 256$) and *XSTREAM* ($K = 100$, $C = 100$, $l = 15$). To compare the running of different methods please refer to column $h = 5$ in which RHF_K uses its default hyperparameters ($h = 5$ and $t = 100$).

The table, ordered by n, shows that indeed RHF_K running times linearly increase in *n, d, h, t*. Overall the running times exhibit two takeaways: i) robustness to noisy features comes at a cost. Both RHF_K and *XSTREAM* are slower than *ISO* when the number of dimensions is high; ii) RHF_K is not only always faster than *XSTREAM* but it can outperform it by up to 50 times in some datasets, while being 14 times faster on average.

Furthermore, we show in Fig 3.8 the running times for increasing *n*, *d*, *t* and *h* while keeping fixed the remaining hyperparameters. The plots show that execution time linearly increases with increasing input sizes (*n* and *d*) as well as for incresing height *h* and ensemble size *t*.

	n	d	h=3	h=5	h=7	t=30	t=60	t=90	ISO	XSTREAM
vertebral	240	6	0.009±0.001	0.012±0.0	0.018±0.001	0.004±0.001	0.008±0.001	0.012±0.001	0.22±0.018	0.493±0.038
ionosphere	351	33	0.04±0.003	0.064±0.007	0.096±0.008	0.018±0.002	0.04±0.004	0.05±0.005	0.239±0.023	1.011±0.059
wbc	378	30	0.039±0.004	0.066±0.005	0.097±0.007	0.02±0.002	0.043±0.002	0.05±0.005	0.256±0.012	1.211±0.095
arrhytmia	452	274	0.374±0.053	0.717±0.126	1.045±0.178	0.227±0.018	0.444±0.071	0.686±0.094	0.378±0.012	2.231±0.205
breastcancer	683	9	0.026±0.001	0.043±0.003	0.061±0.003	0.012±0.001	0.024±0.002	0.034±0.003	0.198±0.021	1.906±0.212
pima	768	8	0.027±0.002	0.045±0.002	0.063±0.001	0.013±0.001	0.027±0.001	0.033±0.003	0.225±0.021	1.906±0.218
penglobal	809	16	0.045±0.004	0.073±0.006	0.116±0.006	0.02±0.002	0.045±0.004	0.063±0.007	0.273±0.013	2.362±0.15
kdd_finger	1033	3	0.017±0.0	0.027±0.001	0.037±0.001	0.007±0.001	0.012±0.001	0.023±0.002	0.244±0.018	1.506±0.168
yeast	1191	8	0.041±0.002	0.06±0.005	0.093±0.002	0.017±0.002	0.032±0.003	0.06±0.003	0.272±0.021	2.491±0.177
vowels	1456	12	0.06±0.005	0.116±0.005	0.166±0.004	0.032±0.002	0.054±0.006	0.082±0.009	0.271±0.022	3.579±0.168
cardio	1831	21	0.145±0.007	0.239±0.011	0.338±0.018	0.061±0.006	0.137±0.011	0.19±0.018	0.202±0.009	4.465±0.242
abalone	1920	7	0.06±0.001	0.099±0.002	0.13±0.006	0.028±0.0	0.047±0.004	0.06±0.006	0.215±0.021	3.672±0.237
kdd_ftp_distinct	2876	3	0.033±0.004	0.071±0.002	0.093±0.006	0.015±0.002	0.036±0.004	0.065±0.002	0.384±0.017	3.421±0.143
musk	3062	166	4.001±0.462	11.602±1.49	15.392±2.209	1.857±0.281	5.436±1.029	9.287±1.015	1.233±0.25	14.817±0.467
thyroid	3772	6	0.09±0.008	0.135±0.01	0.197±0.015	0.047±0.003	0.085±0.007	0.125±0.013	0.355±0.028	6.914±0.272
spambase	4601	57	1.574±0.246	3.389±0.593	4.56±0.823	0.758±0.108	2.507±0.462	4.282±0.926	0.945±0.106	7.281±0.39
wine	4898	11	0.201±0.015	0.314±0.024	0.369±0.036	0.081±0.009	0.184±0.016	0.297±0.021	0.535±0.027	9.264±0.469
satellite	5100	36	0.756±0.091	1.411±0.165	2.155±0.344	0.348±0.045	0.903±0.091	1.003±0.177	0.795±0.071	12.233±0.502
kdd_ftp	5214	3	0.068±0.007	0.134±0.001	0.143±0.014	0.028±0.003	0.077±0.001	0.098±0.01	0.363±0.041	6.268±0.262
satimages	5803	36	0.793±0.105	1.392±0.267	2.921±0.346	0.443±0.052	0.66±0.116	1.152±0.18	0.776±0.094	14.189±0.64
annthyroid	7200	6	0.169±0.018	0.248±0.022	0.422±0.009	0.073±0.007	0.17±0.012	0.213±0.024	0.611±0.058	13.063±0.494
mnist	7603	100	7.1±0.919	13.525±1.798	22.096±3.271	3.238±0.556	8.389±1.692	13.482±2.167	2.35±0.43	35.233±1.352
mammography	11183	6	0.207±0.024	0.426±0.034	0.624±0.051	0.128±0.01	0.233±0.022	0.391±0.032	0.68±0.053	17.445±0.656
kdd_other	12844	3	0.177±0.014	0.302±0.011	0.323±0.034	0.092±0.001	0.157±0.014	0.204±0.022	0.705±0.05	15.697±0.626
magicgamma	19020	10	0.56±0.047	1.274±0.136	2.175±0.08	0.306±0.033	0.723±0.072	1.103±0.111	1.817±0.248	39.245±1.638
shuttle	49097	9	2.784±0.388	4.008±0.675	5.497±0.935	1.013±0.156	2.271±0.28	2.431±0.283	4.413±0.624	67.466±2.044
aloi	50000	27	48.809±7.767	43.89±12.505	68.517±13.232	22.279±2.152	34.525±6.833	37.907±9.08	4.357±0.743	93.076±3.446
wikiqoe	55932	17	29.765±3.779	43.361±6.654	43.489±8.843	14.206±1.894	24.894±4.598	26.367±6.483	5.767±1.002	72.863±2.668
kdd_smtp_distinct	71257	3	1.216±0.052	1.422±0.169	1.849±0.234	0.525±0.047	0.883±0.097	1.014±0.055	4.314±0.667	79.992±2.782
smtp29	96554	29	81.095±13.567	94.375±19.87	91.099±27.896	41.095±6.895	51.058±14.207	60.92±15.263	8.863±1.453	143.405±6.839
kdd_smtp	96554	3	1.379±0.186	2.011±0.34	3.033±0.345	0.531±0.055	1.2±0.133	1.755±0.2	4.132±0.336	110.532±4.152
kdd_http_distinct	222027	3	4.221±0.512	8.013±0.608	10.36±0.532	2.056±0.213	4.18±0.21	5.967±0.765	7.497±0.615	247.883±9.316
mulcross	262144	4	8.926±1.451	23.597±1.986	31.115±5.396	6.792±1.137	12.806±2.431	16.836±1.147	10.139±1.343	453.331±29.574
cover	286048	10	44.564±4.098	61.941±7.911	75.937±10.824	23.85±2.916	34.951±5.235	42.301±6.137	15.503±2.411	450.391±20.046
http_logged	567498	3	11.912±1.116	19.775±2.023	23.971±2.144	7.188±0.646	14.114±0.978	16.97±1.816	20.262±3.364	606.666±12.539
kdd99	620098	29	269.341±45.138	379.767±58.012	515.339±65.546	133.328±24.344	208.811±23.779	318.45±41.242	37.559±5.411	777.502±15.695
http29	623091	29	224.884±39.142	419.309±47.147	501.436±63.773	117.957±23.109	207.724±23.533	289.49±29.192	33.5±1.987	778.882±12.502
kdd_http	623091	3	14.377±1.219	20.666±1.056	26.081±2.952	5.684±0.763	12.174±0.944	16.217±1.374	21.686±2.608	642.201±9.454

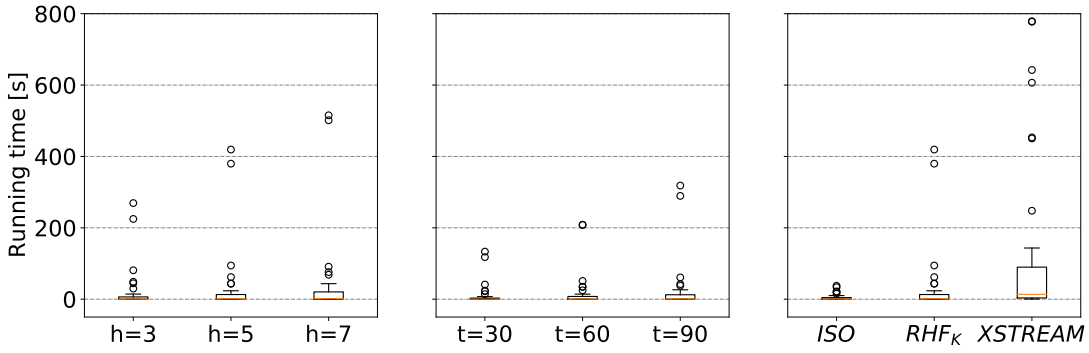


Table 3.2: Running times of the RHF_K algorithm for increasing number of instances n, number of dimension d, max tree height $h \in [3, 5, 7]$ and number of trees $t \in [30, 60, 90]$. The table contains also the running time of *ISO* ($\psi = 256$) and *XSTREAM* (parameters). The boxplots under the table illustrate the running times for increasing tree height *h* (left), number of trees *t* (middle) and the overall comparison of *ISO,* RHF_K and *XSTREAM* running times when the default parameters are used (right).

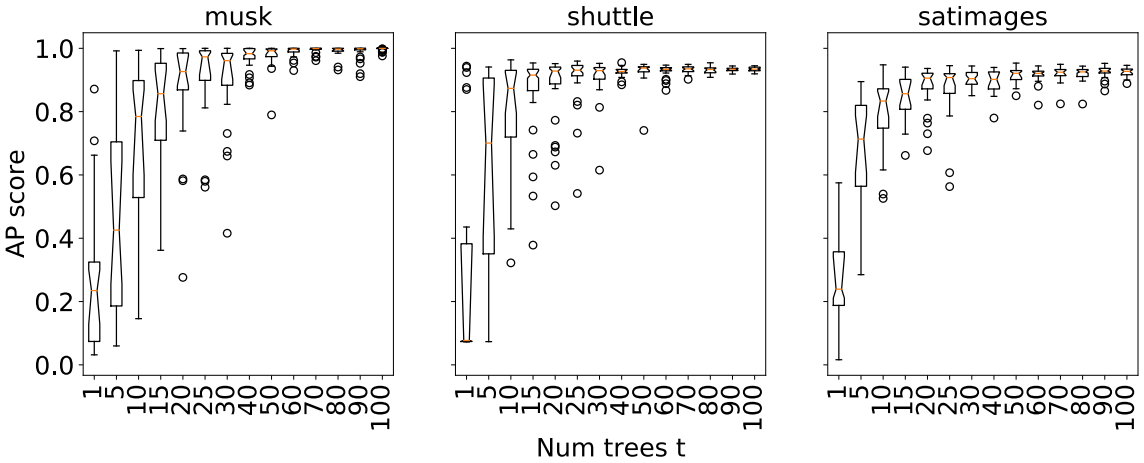


Figure 3.7: HyperParameters tuning: Average Precision score for increasing number of trees *t* on *http_logged* (left), *shuttle* (middle) and *breastcancer* (right) datasets. We observe that performances converge already for small number of trees.

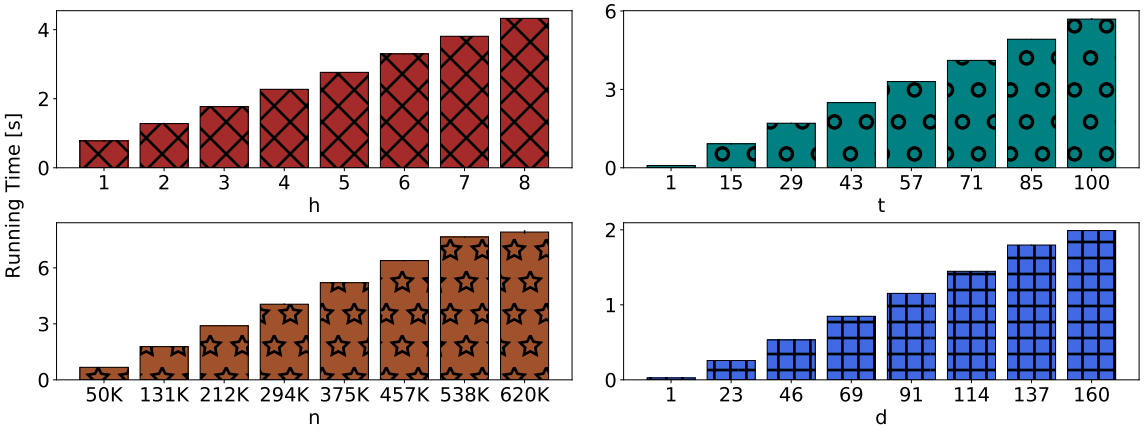


Figure 3.8: Scalability analysis: Empirical Running time complexity analysis for increasing increasing *n*, *d*, *t* and *h* while keeping fixed the remaining hyperparameters. The plots show that execution time linearly increases with increasing input sizes (*n* and *d*) as well as for incresing height *h* and ensemble size *t*.

3.5 Conclusions

We present a novel anomaly detection method called *Random Histogram Forest (RHF)*, which builds a random forest while using the *Kurtosis* score as splitting criterion. The anomaly score of each instance is computed as the information content of the *leaf* it belongs to. We provide an extensive experimental evaluation on 38 public datasets, including all datasets used as benchmarks for anomaly detection, to the best of our knowledge and 64 private ones. Our experimental evaluation shows that our approach outperforms the other approaches in terms of average precision, while being simple and intuitive. Moreover the performance of our algorithm are consistently good over a wide range of values for their hyperparameters, while it requires only two hyperparameters. Finally, our proposed *Kurtosis Split* shows to be effective in high dimensional datasets while maintaning linear running time in the size of the input dataset.

Chapter 4

Online Anomaly Detection Leveraging Stream-Based Clustering and Real-Time Telemetry

In this chapter we describe the proposed *stream* unsupervised algorithm called *ODS* that leverages DenStream, an unsupervised clustering technique, and apply it to measurements collected from real network equipment studying so a *monitoring* application. We first introduce the new challenges in the monitoring application in Section 4.1 and analyze existing monitoring methods in Section 4.2. We then describe the algorithm in Section 4.2.2 before explaining in details the experiments done and how data was captured in Section 4.3. The evaluations are done in Section 4.5. Finally, a summary of our finding and remarks are given in Section 4.7.

4.1 Introduction

Recent technology evolution allows network equipment to continuously stream a wealth of "telemetry" information, which pertains to multiple protocols and layers of the stack, at a very fine spatial-grain and high-frequency. This deluge of telemetry data clearly offers new opportunities for network control and troubleshooting, but also poses a serious challenge for what concerns its real-time processing. We tackle this challenge by applying *streaming machine-learning* techniques to the continuous flow of control and data-plane telemetry data, with the purpose of real-time

detection of anomalies. In particular, we implement an anomaly detection engine that leverages DenStream, an unsupervised clustering technique, and apply it to features collected from a large-scale testbed comprising tens of routers traversed up to 3 Terabit/s worth of real application traffic. We contrast DenStream with batch algorithms such as DBScan and Local Outlier Factor (LOF), as well as stream algorithms such as the windowed version of DBScan, ExactSTORM, Continuous Outlier Detection (COD) and Robust Random Cut Forest (RRCF). Our experimental campaign compares these seven algorithms under both accuracy and computational complexity viewpoints: results testify that DenStream (i) achieves detection results on par with RRCF, the best performing algorithm and (ii) is significantly faster than other approaches, notably over two orders of magnitude faster than RRCF. In spirit with the recent trend toward reproducibility of results, we make our code available as open source to the scientific community.

Nowadays network Operations and Management (OAM) increasingly relies on the ability to stream and process, in near real-time, useful "features" from network equipment. An integral part of the OAM task is, e.g., to ascertain whether the operational conditions are *normal* or *anomalous* and intervene, when needed, by quickly repairing eventual problems.

Simple Network Management Protocol (SNMP) has long been the de facto standard to gather fairly coarse information from the network management, control and data planes. Consequently, SNMP has been used for anomaly detection for long time [143]. In the SNMP paradigm, the server initiates the data collection from hundreds of devices, with a pull-based approach, at traditionally low frequency (i.e., in the order of minutes). More recently, Model-driven telemetry (MDT) [150, 9, 8, 10] has emerged as an interesting alternative to SNMP: instead of having to periodically poll at a low rate (as in SNMP), under MDT subscribers receive continuous stream of operating state information in a standard structured format. In addition to supporting periodic export, MDT further enables to trigger data publication when specific conditions are met.

Rather typically, a common workflow to several vendors (such as Cisco [9], Arista [8] and Huawei [10]) is to express features via Yet Another Next Gen (YANG) [24, 5] data models, encoded with the Google Protocol Buffer (GPB) format, that are then transmitted via the Google Remote Procedure Call (GRPC) protocol. While the use of standard formats and protocol for their export is very desirable, and while the abundance of information is desirable for fine-grained monitoring,

it becomes necessary to also process MDT data *as it is streamed* – a challenging task at the heart of our work. First and foremost, under MDT the data must be processed *in real-time*, which puts a practical cap on the algorithmic complexity. Second, as the data *is streamed continuously*, no assumption on data distributions or length can be made a priori. Third, *data cannot be stored* and algorithms have to perform a single pass on it, which significantly limits the algorithmic design space.

4.2 Related Work

This section overviews related work focused on outlier and novelty detection in computer networks. In particular, Sec. 4.2.1 presents a taxonomy of the relevant work from the network domain viewpoint, whereas Sec. 4.2.2 introduces background material concerning the unsupervised clustering techniques that are relevant for our work.

4.2.1 Outlier detection in computer networks

As already discussed in detail in Section 1, Hawkins [71], defines an outlier as "*an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism*." Generally, anomalies are categorized as point, contextual or group outliers: *Global Point outlier* is an object that significantly deviates from all the objects in the dataset; *Contextual outlier* is an object that significantly deviates from the objects in a context (eg. period in a timeseries); *Group outlier* is a subset of objects that significantly deviates from the entire dataset (i.e., set of instances that do not follow regular patterns in the dataset). The events we consider in this work (see Sec. 4.3) belong to the point and group outlier groups.

In several domains (fraud/intrusion detection, public health etc.) it is assumed that the number of anomalous objects is (much) smaller than the normal objects. For this reason, several methods generate a *normal* (baseline) model of the system and label *anomalous* all the objects that are significantly different. These methods fall into three main groups, namely *(i) statistical*, *(ii) supervised* and *(iii) unsupervised* learning.

(i) Statistical methods use probabilistic models to detect changes in the data. Principal Component Analysis (PCA) is used in [87] to detect anomalous traffic volume in backbone networks

by reducing the n-space of variables into a k-subspace corresponding to the *normal* behavior and a m-subspace corresponding to *anomalies* and *noise*. The subspace method is used in [74] to detect BGP anomalies, extracting the amount of update messages from raw BGP updates every 10 minutes and processing the data in batches of 200 samples. Similarly, [42] and [59] extract features from raw BGP updates (BGP volume, Autonomous System (AS)-Path, etc.) every 5 minutes and perform anomaly detection using the Generalized Likelihood Ratio Test (GLRT) and the t-test respectively. Finally [155] contrasts the covariance matrix of *n* objects against a *normal* precomputed covariance matrix to detect flooding attacks in the *KDD'99* [4] dataset while [135], using the same dataset, proposes a Principal Component Classifier (PCC) yielding an anomaly score for both major and minor subspace components. These methods are interesting but inherently obscure for the human operator that needs to interpret the results, as the semantic of the original feature domain is lost due to the projection transformation – a significant matter of concern that render the techniques less appealing for practical purposes [1, 107].

(ii) Supervised methods learn both *normal* and *anomalous* behaviors from *labeled data* and then classify each new object *normal* or *anomalous* depending on which class fits to. Due to the lack of data, most of the methods use *KDD'99* and *NSL-KDD* [140] datasets or have to manually label private datasets. Among the most used algorithms there are Decision Trees (C4.5) [121], Support Vector Machines (SVM) [38] and Artificial Neural Networks (ANN) [147]. Authors of [148] obtains a $2\times$ SVM training time reduction using the *NSL-KDD* dataset augmented and transformed by the logarithm marginal density transformation while [115] and [139] use the *KDD'99* dataset to evaluate their ANN models which reduce false positive rate and minimize overall computational overhead respectively. Regarding BGP events, [89] trains and tests a decision tree (C4.5) using features extracted every minute from RouteViews and RIPE NCC archive during known misconfiguration events, Slammer worms and electricity blackouts. The same source, sampled every 30 seconds, is used by [41] which compares different algorithms as decision trees, Naive Bayes and SVM. They process the data using a sliding window of 10 samples, corresponding to 5 minutes, which slides of two minutes every time. Their system raise an alarm if at least 60% of the samples in the window are labeled anomalous allowing them to detect events as early as four minutes. The importance of BGP features is studied in [18] applying both Fisher [36] and mRMR [70] feature selection techniques, which concludes that 65% of the selected features are *volume-based* (i.e. BGP announcements, IGP packets, EGP packets etc.) and that these are

more relevant and perform better than *AS-path* features. Supervised approaches are however of little portability across datasets, where features and labels differ, and the applicability of such models is therefore limited to the very specific use-case under study.

(iii) Unsupervised methods could prove particularly useful to detect outliers as they are able to find unknown patterns without using labeled data: outliers are identified as items different from the previously found patterns. Best known approaches [123, 88, 105, 98, 33, 102, 46, 101, 117] uses distance (or density) to group together similar objects and label *anomalous* those far from the neighbors, and can be further categorized into *batch* vs *stream* methods. *Batch* algorithms (such as K-Means [95], DBScan [52], Local Outlier Factor (LOF) [28]), require the access to the *entire dataset at once* (to compute centroids-objects distances or pairwise distances) and iteratively converge to a final solution. *Stream* algorithms (such as iGDCA [108] and DenStream [32]) are instead designed to build, maintain and update models incrementally *at each new sample*. The above classes of work are closer to ours and deserve a deeper look.

Batch unsupervised methods are used in [123, 105, 88, 98, 33] for outlier detection. For instance, [123] applies k-NN, a distance-based method to detect anomalies in wireless sensor networks, whereas [105] uses K-Means to detect anomalous flows (i.e. counters of bytes, packets etc.). Several algorithms (i.e. unsupervised SVM, LOF, k-NN) are instead compared on the DARPA dataset in [88], asserting that the best performing one is LOF – which we thus include in the comparison. Density-based sub-space clustering methods are used in [98], combining evidence accumulation to identify anomalies. To reduce the high computational complexity of such batch methods, [33] utilizes a discrete time-sliding window to extract and aggregate different flow-resolution levels, in time slots of fixed length ΔT – a reasonable compromise approach that we also consider in this work.

Stream unsupervised algorithms have been used more rarely for network anomaly detection [46, 102, 101, 117]. Authors in [46] employ a discrete time-sliding window and an incremental grid clustering algorithm to detect anomaly traffic in the core network of a Spanish Internet Service Provider (ISP). Closer to our work are [102, 101] that employ DenStream at the application and network layers respectively. In particular, [102] use DenStream to successfully detect Twitter spam using a (tiny) dataset containing approximately 3000 normal and 200 manually labeled

spam entries. Authors in [101] cluster normal vs anomalous packets in the DARPA dataset operating in the data plane, and directly leverage *packet payload*, using the numerical value of each byte HTTP payloads as input features. As such, both domains of application in [102, 101] are rather far from the control-plane telemetry use-case of this paper, and neither [102] nor [101] carry on a systematic evaluation of multiple algorithms as we do in this work.

Additionally, from a practical viewpoint [101] requires continuous hyperparameter tuning and furthermore assuming *prior knowledge* of the percentage of anomalous packets – and cannot thus be readily deployed. In contrast, we make no assumption on the data and further propose principled and automated tuning methodology – that are robust to environmental condition changes.

4.2.2 Overview of clustering algorithms

We now provide background information on the clustering algorithms that we will be using as building blocks for our system in this paper. A summary of the algorithms compared in this work, is present in Table 4.1, along with their main hyperparameters. All the algorithms used in this work are explained in detail in Chapter 2. In particular DBScan and LOF batch algorithms are described in Section 2.6 while ExactSTORM, COD and RRCF stream algorithms in Section 2.7. We point out that the ultimate goals of some of these algorithms is to perform clustering: so while this section briefly covers each algorithm, we defer to Sec. 5.3 a more formal description of our methodology to leverage clustering output for anomaly detection purposes. We introduce now *wDBScan*, a windowed version of DBScan previously used by [33] and the DenStream algorithm which is the building block of our anomaly detection engine.

wDBScan is the windowed version of DBScan. Similar to [33], the algorithm is applied to a batch of samples of length w at a time. When a new sample is available, the window advances by one step, removing the oldest sample and adding the newest one. Thus, in addition to DBScan's ϵ and $MinPts$, it adds the window size w hyperparameter. The complexity of this approach then becomes $\mathcal{O}\left(w\ n \log\left(w\right)\right)$.

DenStream is a clustering algorithm proposed by Cao et al. [32]. It is an algorithm designed for data streams, which extends the density-based strategy introduced in DBScan making it viable for stream model construction. First of all, the algorithm uses a damped window model to weight

the samples: older ones become less important than newer ones via a fading function

$$f(t) = 2^{-\lambda t}, \lambda > 0 \tag{4.1}$$

where λ is the aging hyperparameter. The main idea of the algorithm is the introduction of the so called *micro-clusters (mc)*, i.e., group of close points $p_{i_1}, ..., p_{i_n}$ with creation time stamps $T_{i_1}, ..., T_{i_n}$. A *mc* is defined as a (w, c, r) where *w* is the weight, *c* is the center and *r* is the radius of the *mc*. The weight *w* is given by the number of elements in the *mc* weighed by their generation time T_{ij} with respect to the current time t:

$$w = \sum_{j}^{n} f(t - T_{i_j}) \tag{4.2}$$

Similarly,

$$c = \frac{1}{w} \sum_{j}^{n} f(t - T_{i_j}) p_{i_j} \tag{4.3}$$

$$r = \frac{1}{w} \sum_{j}^{n} f(t - T_{i_j}) dist(c, p_{i_j}) \tag{4.4}$$

where $dist(c, p_{i_j})$ is the euclidean distance between point p_{i_j} and the center c. By breaking clusters into *mcs*, DenStream allows to dynamically construct clusters of arbitrary shapes. The *mc* weight *w* plays a key role in the model construction, as it discriminates between *outlier* ($w < \beta\mu$) vs *core* ($w > \beta\mu$) micro-clusters (where β and μ are free hyperparameters).

When a new sample is available, DenStream (i) merges it to the nearest *core mc* provided that the radius of the merged cluster does not exceed a given threshold ϵ; otherwise, DenStream (ii) attempts to merge the point to the closest *outlier mc*, and (iii) a new outlier *mc* is finally created by the point if the merge fails.

Not only *mcs* are easy to maintain *incrementally* at each new data point, but notice that model construction is a continuous process in DenStream: an outlier *mc* can indeed become a core *mc* when its weight increases as new points are added to it. Similarly a core *mc* becomes an outlier *mc* (and ultimately vanishes) if no new data points are added for long periods. The authors show that the minimal time span for a core *mc* fading into an outlier one is $Tp = \frac{1}{\lambda} log(\frac{\beta\mu}{\beta\mu - 1})$ therefore it is natural to check them every *Tp* time periods. Two *offline* phases can be found in DenStream: *Initialization* and *Generating Final Clusters*. The authors propose indeed to obtain the initial *mc* as the output of DBScan applied to the first *InitN* points, called *buffer*, and then maintain them

Table 4.1: Algorithms compared in this work

Algorithm	Type	HyperParameters
DBScan	Batch	ϵ, *MinPTS*
LOF	Batch	*n_neighbors*, *contamination*
wDBScan	Windowed	ϵ, *MinPTS*, *w*
ExactSTORM	Stream	R, K, w
COD	Stream	R, K, w
RRCF	Stream	t, w, contamination
DenStream	Stream	ϵ, λ, β, μ

incrementally. Similarly, they propose to obtain the *final clusters*, when requested, applying again DBScan to the set of core *mcs* considering them as a virtual point located at the center of the *mc*.

4.3 Testbed and Datasets

We study and compare the proposed method using publicly available datasets [6], we have gathered and released previously in [117]. The datasets have been collected in a state of the art testbed, comprising tens of real routers, running real protocols and traversed by Tbps traffic (Sec.4.3.1). The testbed is used in several experiments where anomalous events are injected in randomly chosen nodes in a controlled fashion (Sec.4.3.2). In turn, these controlled anomalies affect the stream of telemetry features (Sec.4.3.3), as we illustrate for the sake of clarity (Sec.4.3.4).

4.3.1 Testbed

The dataset is extracted from a testbed replicating a traditional clos topology of a CSP datacenter shown in Fig.4.1. For redundancy, each leaf is connected to each spine via 4$\times$100Gbps fiber links, so that the nodes have 25 interfaces on average. On the operational level, the datacenter is designed with BGP as the only routing protocol, following guidelines in [110].

Though the testbed does not involve real users, it does use real equipment, protocols and applications typical of production networks. We thus disregard experiments collected under no traffic load (mostly useful for testing) and limitedly consider those where real application mixtures

Table 4.2: Experimental datasets available at [6]

Experiment ID	Traffic Load	No. Anomalies	Duration	Used for
2	1 Tbps	11	1 h	Tuning parameters (Sec. 4.4)
3	1 Tbps	8	0.55 h	Tuning parameters (Sec. 4.4)
5	1 Tbps	12	2 h	Test parameters (Sec. 4.5)
9	2.9 Tbps	5	0.75 h	Test parameteres (Sec. 4.5)
10	2 Tbps	5	0.55 h	Test parameters (Sec. 4.5)

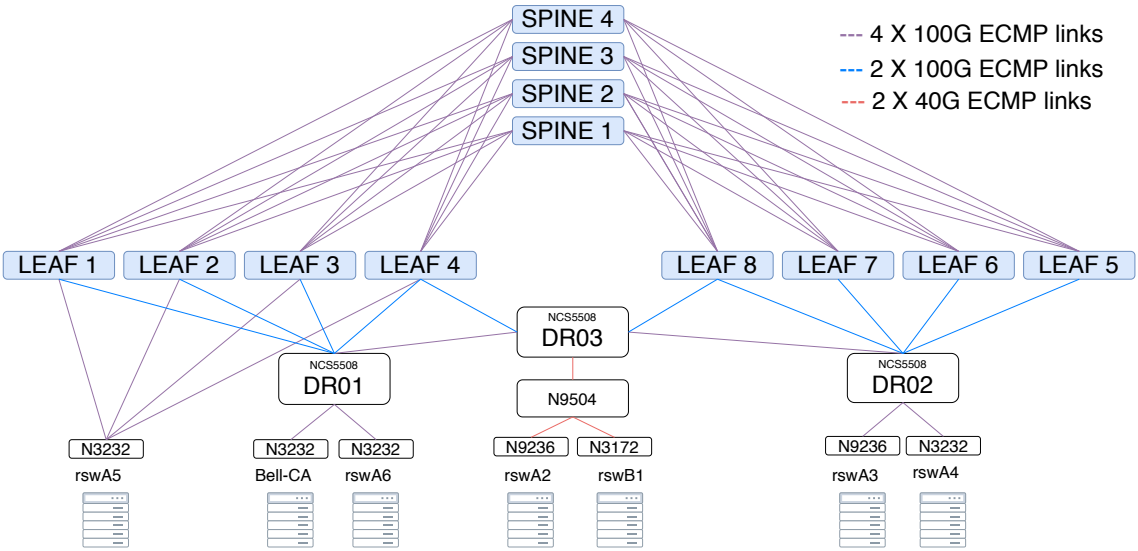


Figure 4.1: Testbed replicating a traditional clos topology of a CSP datacenter

are generated from servers in the racks connected to the ToR switches (the Nexus 2/3/5000 and 9000 series) to generate up to 3 Tbps of aggregated traffic (a mixture of TCP, AMR-WP VoIP and G.711a calls, Skype-1050P, Blue Ray and 4K YouTube streaming).

4.3.2 Data collection

Multiple experiments, listed in Table 4.2, with different characteristics and scenarios are performed. While minute-level telemetry collection is generally practice in the industry, the dataset we use in this work has been gathered with the fastest sampling period supported by the products, namely of ΔT=5 seconds. Every ΔT, each of the N nodes stream a snapshot of its F features to the collector: each experiment is a point $X \in \mathbb{R}^{NSF}$ where N is the number of nodes, S is the number of collected samples during that experiment and F the number of features. All the available features are described by YANG models [11] and then extracted, decoded and stored by Pipeline [7] as compressed CSV files. The repository [6] contains experiments undergoing differ-

ent traffic *load* (from 0 to 3 Tbps), different *number* of anomalies injected (from 0 to few 10s) and different anomalies *types*, such as BGP port flapping (Link Failure - Point anomalies), BGP leaks and BGP clears (Direct Unintended BGP Anomalies - Collective Anomalies). An interesting point is how to ensure that the synthetic anomaly injection process yields to outliers that retain similarities with anomalies found in the real-world. While a systematic quantitative evaluation of the anomaly injection process is outside the scope of this paper, it is possible to provide preliminary qualitative insights on this point. Notice that these synthetically injected anomalies represent the type of BGP anomalies that are typically found in real-data [65]; additionally, such anomalies are injected by actioning on the protocol (e.g., automatically activating/deactivating links for flapping, purposely misconfiguring tables for leaks, and resetting tables for clears) as it would happen in case of real connectivity problem (flapping) or "fat finger" (leaks, clears), so to trigger real protocol reaction. As such, while the temporal patterns of the anomalies are likely unrealistic (i.e., since they are periodical and more frequent that what it can be expected) by separating anomalies by a long enough time, we can ensure anomalies are roughly independent, and as such should independently trigger alarms. Overall, the synthetic injection process is expected to provide a sufficiently realistic benchmark.

The working condition of the system is classified in two categories, i.e., *normal* vs *anomalous*. The system works by default in normal mode, and each experiment starts with a normal period (lasting at least 40 samples), after which controlled anomalous events are injected at randomized node locations. Depending on the dataset, the anomalies are injected by spacing them by 300 seconds or more and all of them are tracked in a ground truth database available in the same repository. The groundtruth file includes the *root node* in which the event is injected, the *timestamp* and the *type*. We point out that we do not leverage ground truth information to build our data-driven models (i.e., as one would to in case of supervised classification), but rather use ground truth only to assess the performance of the unsupervised methods.

Whereas the start time of the anomalous event is known, the event duration is not deterministic: the event injection triggers the BGP update process, after which BGP converges to a new stable state. We discuss with product line experts to set an expected event duration: based on expert knowledge related to both protocol dynamics (i.e., the convergence process of BGP), as well as business objectives (i.e., the ability to gather actionable alarms on a timescale interpretable by

humans), network experts consider an event ended 3 minutes after its injection time.

Clearly, as any threshold that can be set arbitrarily, its tuning may impact algorithmic performance evaluation: for instance, setting a very short event duration (sub-minute) would raise several events that would be wrongly counted as "false alarms" (since BGP did not converge yet in practice); conversely, setting a too long duration (e.g., larger than the interval between two consecutive injections) could lead to superposed event. Based on properties of the injection process, and on our preliminary observations of the timeseries, we concur with experts that 3 minutes is an advisable event duration for this datasets also from the viewpoint of machine learning experts.

4.3.3 Telemetry features

The streamed KPI (aka features in machine learning terms) available in the testbed are a subset of the YANG [5] state of the devices, exported by GRPC to an inbound collector. In a nutshell, YANG models define a hierarchy (i.e., tree) of data that can be used for configuration, state sharing and notifications; in the model, each node has a name, and either a *value* or a set of child nodes. At the same time, it is worth stressing that the YANG hierarchy of devices in the testbed comprises over 378,000 lines, describing a hierarchy of over 45,000 features, with *nearly 5,000 types pertaining to the BGP protocol alone*. From a machine learning viewpoint, it would be counter-productive, due to the curse of dimensionality, to apply any clustering algorithm to such a highly dimensional data. Additionally, from a network-expert viewpoint, collecting and exporting features consumes CPU and bandwidth resources: as such, it is impossible to collect, for all nodes and interfaces, the totality of the supported features – which rules out the possibility to conduct classic "feature selection" algorithms. Product line experts configured the testbed to collect the most relevant control and data plane KPIs according to their domain knowledge, and we therefore take the resulting set of features (reported in Appendix A for completeness) as a given. However, it is well known that not all features are equally important in machine learning terms: by discarding constant or categorical features from the full set of available features, we finally extract a subset of 82 non-trivial features (reported in Appendix B for completeness).

4.3.4 Dataset at a glance

For the sake of clarity, we illustrate samples of the dataset in Fig. 4.2, to exemplify the types of KPI signals and anomalies present in the dataset from spatial and temporal angles.

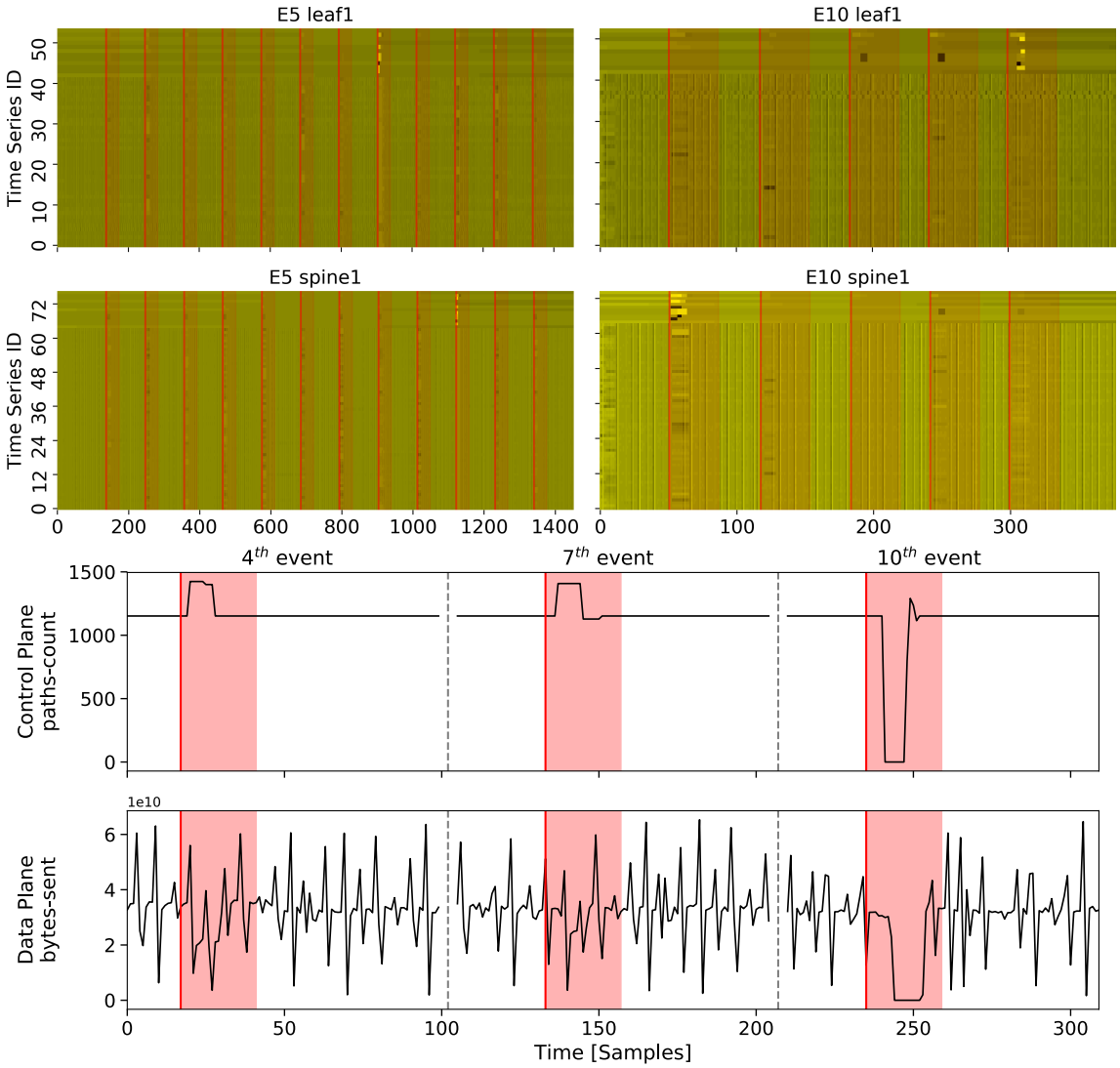


Figure 4.2: Dataset at a glance: Top plots depict example of the Multivariate Time Series as heatmaps for *leaf1* and *spine1* on E5 (plots on the left) and E10 (plots on the right). Bottom plot reports temporal evolution for sample features and annotated ground truth for node *spine1* on E5: control-plane *paths-count* (top), vs data-plane *bytes-sent* on interface HundredGig0/0/0/0 (bottom).

Spatio-temporal view

We start from a heatmap representation of the multivariate data collection in the top of Fig. 4.2: x-axis represent the time, y-axis represents different features whose values are encoded as colors. To portray two different datasets and nodes, we select *leaf1* and *spine1* extracted from experiments E5 and E10. It is easy to observe that in E5, all the 12 BGP clear events have a noticeable impact on *spine1* features, and to a smaller extent, on *leaf1*. While it is visually possible to distinguish the first four events in *spine1*, the same does not hold for *leaf1*: this happens since these events are injected in nodes that are directly connected with *spine1* but are at 2 hops away from *leaf1*. Thus, *leaf1* features are most noticeably affected when an event is injected in a topologically nearby node. Heatmaps on the right show that the events injected in E10 have even a less noticeable impact: link failures indeed impact only one (out of four) direct links between two nodes, and consequently only some of the features related to that particular link are affected, which can be hard to detect.

Temporal view

Events are more easily noticeable by considering a temporal view, on the bottom part of Fig. 4.2, that shows an example of CP (*paths-counts*) and DP (*bytes-sent*) features for *spine1* in E5. The ground truth is represented with a vertical red line representing the anomaly injection time and a shaded window for the anomaly duration, and we depict only 3 out of the 12 injected events for the sake of readability. From these few examples, it can be expected that accurately detecting all events, from all nodes, can be a quite challenging due to the nature of the events, that can yield to weak signals for some nodes and anomaly type.

4.4 Methodology

Traditional clustering-based approaches, e.g. DBScan are mainly designed to produce clusters rather than detecting outliers or other types of anomalies, which is our ultimate goal. As such, in this section we first specify how we move from clustering to outlier detection Sec. 4.4.1. We next illustrate in Sec. 4.4-B/C the careful hyperparameter selection procedure we followed to ensure fair performance comparison in Sec. 4.5.

4.4.1 From clustering to anomaly detection

All the considered methods succeed in making a distinction between *normal* samples (belonging to a cluster) and *outliers/noise*. We further use these peculiarities to trigger anomaly detection for each algorithm. In particular, a sample is considered *anomalous*

(i) by DBScan, if it cannot be merged to any cluster;

(ii) by wDBScan, if by adding it to the time-sliding window, it cannot be clustered together with the previous samples;

(iii) by LOF, ExactSTORM, COD and RRCF, if it is assigned an *anomalous* label

(iv) by DenStream, if it is successfully merged to an outlier-*mc*, or a new outlier-*mc* needs to be created for that sample.

Note that the exact composition and size of clusters is affected by the algorithms hyper-paramers: e.g., the minimum size of a cluster is either *explicitly* (e.g., as for DBScan via $MinPTS$, cfr. Sec.4.4.2) or *implicitly* specified (e.g., as for ODS, cfr. Sec.4.4.3)

For the sake of illustration, we report the pseudo-code (merging, promotion and pruning phases) of the DenStream algorithm, together with the proposed changes for label assignment in ODS (emphasised and between [square brackets]) in Algorithm. 2. In particular, we propose two major changes with respect to the original version of DenStream to be found in the initialization phase and in the offline clustering part. While, in the initialization phase (line 1), the original version of DenStream uses DBScan to obtain the micro-clusters and then start maintaining them incrementally, we cluster together the first S samples of the stream assuming they are event free and not contaminated by anomalies. By doing so, we obtain a cluster of *normal* samples (normal working condition of the system). We remove, similarly, the offline part of DenStream (line 31) in which DBScan is applied to cluster together micro-clusters. Our goal is to find out if the samples, as soon as they are available, are *normal* or *anomalous*; we need so to know only if the samples can be merged to existing *normal-mc* or not. Instead, we do not need to group together different micro-clusters, which may represent different normal states, to obtain a macro cluster of *normality*. Other changes pertain to automated tuning, which we detail in Sec. 4.4.3.

Algorithm 2: DenStream [and *ODS*]

Initialization with DBScan [skipped in ODS]
while *ns (new sample)* **do**
  find closest core *mc* and try to merge *ns*
  if $r_c < \epsilon$ $[< \bar{r} + k_r\sigma_r]$ **then**
    merge *ns* to core *mc*
    [return label $normal$, r_c]
  else
    find closest outlier *mc* and try to merge *ns*;
    if $r_o < \epsilon$ $[\bar{r} + k_r\sigma_r]$ **then**
      merge *ns* to outlier *mc*
      if outlier *mc* weigth $> \beta\mu$ $[< \beta/(1 - 2^{-\lambda})]$ **then**
        promote outlier *mc* to core *mc*
      end if
    else
      generate new outlier *mc* by *ns*
    end if
    [return label $anomalous$, r_c]
  end if
  apply fading function $2^{-\lambda \cdot t}$ to all *mcs*
  if t mod $T_p == 0$ **then**
    for each core *mc* **do**
      if $w_p < \beta\mu$ $[< \beta/(1 - 2^{-\lambda})]$ **then**
        remove core *mc*
      end if
    end for
    for each outlier *mc* **do**
      if $w_o < \beta\mu$ $[< \beta/(1 - 2^{-\lambda})]$ **then**
        remove outlier *mc*
      end if
    end for
  end if
  DBScan on core *mcs* [skipped in ODS]
end while

Table 4.3: Hyperparameters: Number and range of combinations tested for each method and final selection

Method (num. combinations)	Hyperparameter Name	[Range]@step	Selected Value
DBScan (500)	ϵ	$[1, 20]$, @1	6
	$MinPts$	$[2, 50]$, @2	18
LOF (1960)	$n\_neighbors$	$[1, 50]$, @1	24
	$contamination$	$[0.001, 0.2]$, @0.005	0.065
wDBScan (8000)	ϵ	$[1, 20]$, @1	9
	$MinPts$	$[2, 50]$, @2	3
	w	$[20, 100]$, @5	80
ExactSTORM (3040)	R	$[1, 20]$, @1	10
	w	$[10, 100]$, @5	95
	K	$[2, 10]$, @1	2
COD (3040)	R	$[1, 20]$, @1	12.5
	w	$[10, 100]$, @5	50
	K	$[2, 10]$, @1	5
RRCF (800)	t	-	100
	w	$[5, 100$, @5	95
	$contamination$	$[0.001, 0.2]$, @0.005	0.03
ODS (135)	ϵ	dynamic	$\bar{r} \pm k_r \sigma_r, k_r = 3$
	λ	$[0.01, 0.45]$, @0.03	0.125
	β	$[0.1, 1]$, @0.1	0.4

4.4.2 Hyperparameter selection (DBScan, LOF, wDBScan, ExactSTORM, COD and RRCF)

For all the methods, we first perform a hyperparameter selection phase, to select hyperparameters yielding to good performance as measured by classic metrics from information retrieval (i.e., precision, recall and F_β scores). Given that anomalies are rare, we use the $F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$, setting $\beta = 0.5$ to account for imbalance by non-linearly interpolating precision and recall. In particular, we use E2 and E3 from Tab. 4.2 for hyperparameter selection. Generalization capabilities of the tuned algorithms will be tested on entirely different datasets in Sec. 4.5. The full set of hyperparameters explored, along the total number of combination tested per protocol, and the resulting selection is summarized in Tab. 4.3.

We use classic *grid optimization*, i.e., an exhaustive search, to find the hyperparameters that reach the best performance. We do not perform however a blindly search, as it is of fundamental importance the research of the hyperparameters in the correct intervals and magnitudes, therefore we follow the best practices suggested by the authors in the determination of grid boundaries.

Grid boundaries

For DBScan ϵ and *MinPTS* hyperparameters, we perform a grid search varying $\epsilon \in [1, 20]$ with a unit step and $minPTS \in [2, 50]$ with step equal to 2, for a total of 500 different hyperparameters combination.

wDBscan follows the same principles for what concerns $\epsilon \in [1, 20]$ and $minPTS \in [2, 50]$. The window size *w* is searched in $w \in [20, 100]$ in steps of 5 units, which is equivalent to time windows ranging from 1 minute to 8 minutes (in line with BGP time spans), obtaining so a total of 8000 different hyperparameters tested for each node.

The number of neighbors used by LOF is searched in $n\_neighbors \in [1, 50]$ with unit step, while we explore $contamination \in [0.001, 0.2]$ with a step equal to 0.005, obtaining a total of 1960 combinations.

For what concerns ExactSTORM and COD, we perform the *grid search* varying $R \in [1, 20]$ and $k \in [2, 10]$ with a unit step while $w \in [10, 100]$ in steps of 5 units obtaining so a total of 3040 hyperparameters tested for each node.

The number of trees used by *RRCF* is set to $t = 100$ as it is commonly used in ensemble models. We explore the tree size $w \in [5, 100]$ in steps of 5 units while exploring $contamination \in [0.001, 0.2]$ with a step equal to 0.005, obtaining a total of 800 combinations.

Grid search results

Fig. 4.3 shows a heatmap of the $F_{0.5}$ score for DBScan (top left plot), LOF (top right plot), wDBScan (middle left plot), ExactSTORM (middle right plot), COD (bottom left plot) and RRCF (bottom right plot). Clearly, hyperparameter selection is to select the point (or points in a region) that maximizes the $F_{0.5}$ score. For each algorithm, we represent the hyperparameter space of two hyperparameters as a heatmap, to convey an idea of the algorithm stability to (even slight) hyperparameter changes (in the region). We highlight the final hyperparameters choice directly in each plot, i.e., at intersection of the dashed lines.

We observe that DBScan performs the best for $3 < \epsilon < 8$ and $10 < MinPTS < 30$. We select

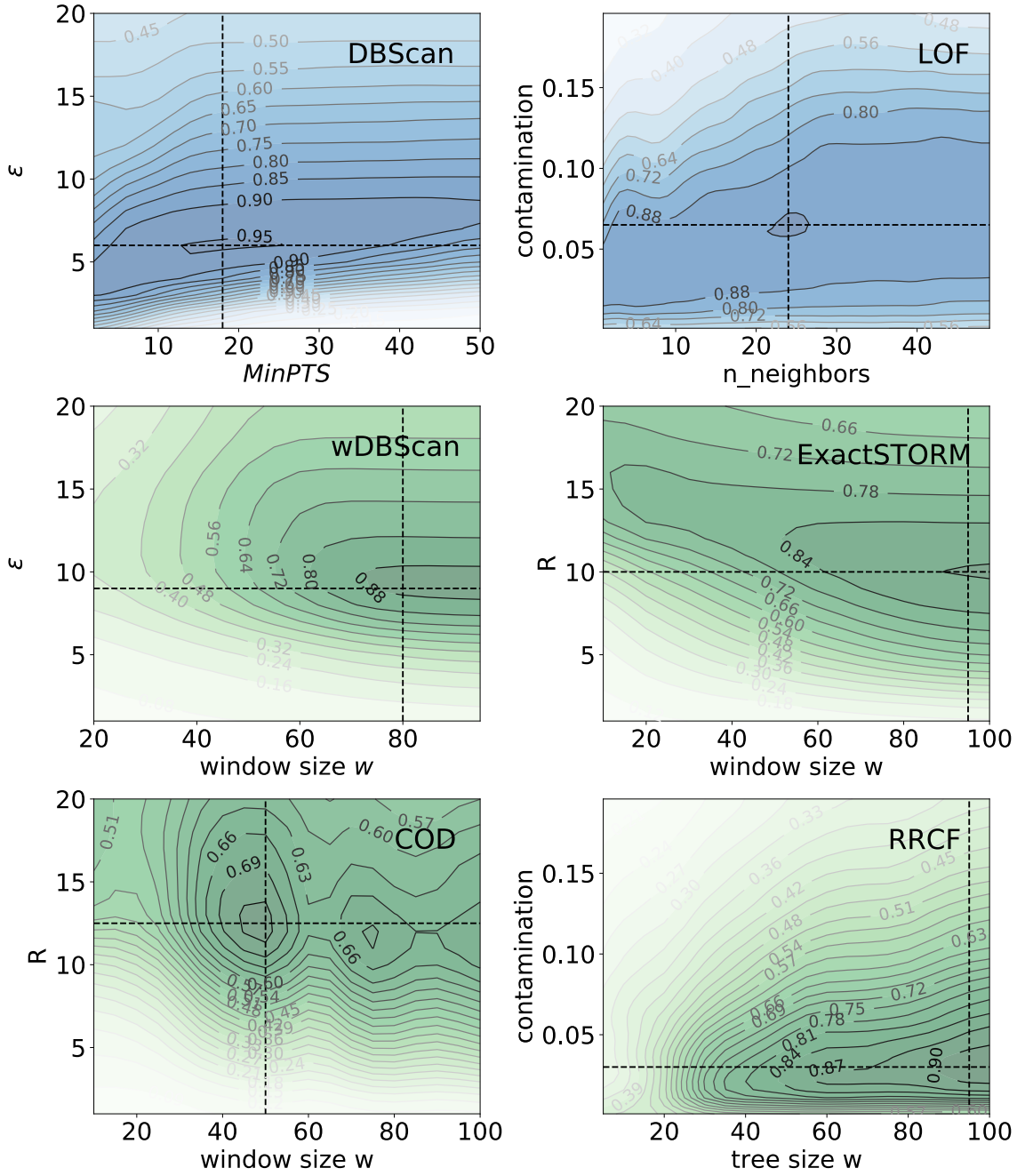


Figure 4.3: Hyperparameter selection: $F_{0.5}$ heatmap for DBScan (top left), LOF (top right), wDBScan (middle left), ExactSTORM (middle right), COD (bottom left) and RRCF (bottom right). Detailed hyperparameters in Tab. 4.3. Selected hyperparameters at the intersection of the dashed lines.

$\epsilon = 6$ and $MinPTS = 18$. We also notice that $F_{0.5}$ changes quickly in ϵ, and is slowly varying in $MinPTS$.

From the top right plot, we observe that the region for which LOF produces the best results

are $0.06 < contamination < 0.07$ while $20 < n\_neighbors < 30$. We select $contamination = 0.065$ and $n\_neighbors = 24$ which is close to the default parameter ($n\_neighbors = 20$).

Heatmaps in middle plots clearly show that the window size need to be $w > 70$ for both wDBScan and ExactSTORM. In the wDBScan case though, we remark that the use of a smaller time-horizon affects the $(MinPTS, \epsilon)$ heatmap so that the best selection appears to fall for $MinPTS < 6$ (much smaller that in the DBScan case) and $\epsilon \approx 9$ (slightly larger than for DBScan). We select $\epsilon = 9$, $w = 80$ and $MinPTS = 3$. We select $R = 10$, $w = 95$ and $k = 2$ for ExactSTORM.

Finally bottom plots show that $R = 12.5$, $w = 50$ and $k = 5$ are the best hyperparameters for COD while tree size $w = 95$ and $contamination = 0.03$ are the best ones for RRCF.

4.4.3 Hyperparameter selection (ODS)

We point out that, as reported in Tab. 4.3, the total number of hyperparameter explored is smaller (135) than in the other cases: this should provide not only a fair, but an expected conservative performance assessment of ODS performance. At the same time, since DenStream is less well known and ODS build on it, we present a more comprehensive explanation of its hyperparametrization. Particularly, we use ingenuity to:

(i) simplify hyperparameter selection by lumping factors whenever possible (as for μ^+), as well as

(ii) proposing dynamic parameterless settings based on statistical properties (for ϵ) and

(iii) resorting to grid search for the remaining ones (λ, β).

Maximum weight μ^+

The weight parameter μ is used jointly with the potential factor β to decide when a given outlier *mc* becomes a new core *mc* (particularly, when $w > \mu\beta$). Given the exponential fading function $f(t) = 2^{-\lambda t}$, and considering a fixed-rate sampling as in our case, the maximum weight a micro-cluster can reach is $\mu^+ = \sum_j f(j) = \frac{1}{1-2^{-\lambda}}$ (since $|f(t)| < 1$ for $\lambda > 0$) which solely depends on λ. By setting $\mu = \mu^+$ we therefore reduce the parameter cardinality, obtaining the new minimal time span for a normal cluster fading into an outlier one $T_p = \lceil \frac{1}{\lambda} log_2(\frac{1}{\beta}) \rceil$, obtained by the equation

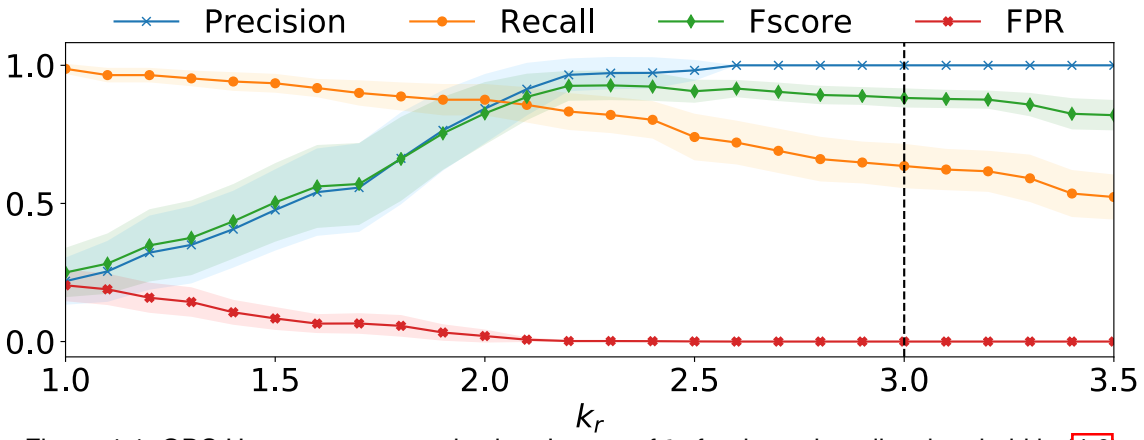


Figure 4.4: ODS Hyperparameter selection: Impact of k_r for dynamic radius threshold in (4.6)

$2^{-\lambda T_p}\mu^+ = \mu^+\beta$ and the rule for outlier micro-cluster *mc* promotion becomes:

$$w > \beta/(1 - 2^{-\lambda}) \tag{4.5}$$

Radius threshold ϵ

The radius threshold is the most important parameter of the algorithm as it delimits the *anomalous* threshold. On the one hand, one may suggest a *fixed* selection of the parameter, that is to compute it as the radius of the cluster obtained merging together the data of an experiment in which no anomalies are injected (i.e. E0 and E1 - baselines). This is the approach we originally followed in [117]. At the same time, we argue that a *fixed* selection of the parameter introduces *portability* issues: as it is necessary to generate baselines for each possible combination of topology, traffic loads and BGP policies, this making the choice fragile and impractical. On the other hand, one could advocate for a *dynamic* selection of the parameter, that is automatically computed as the radius of the cluster obtained merging together the first S samples at the beginning of the model construction and keep updating the threshold as new samples are available. We follow this second path and dynamically set the radius threshold as:

$$\epsilon(k) = \bar{r} \pm k_r\sigma_r \tag{4.6}$$

where $\bar{r}$ is the incremental estimation of the radius mean while σ_r is the incremental estimation [112] of the radius standard deviation, and k_r an arbitrary parameter that allows to control (more precisely, upper bound) the false alarm rate.

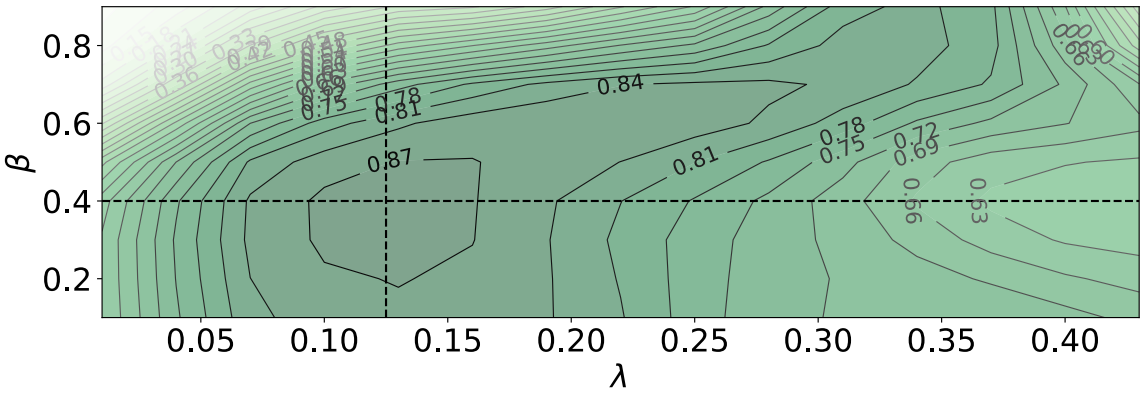


Figure 4.5: ODS Hyperparameter selection: $F_{0.5}$ score for increasing fading factor λ and potential factor β applied on E2 and E3

We observe from (4.3) and (4.4) that the radius has a gamma type distribution (as it is the square root of sums of positives values). Without further assumptions on the radius distribution, we use Chebyshev's inequality:

$$Pr\left(|X-\mu| \geq k\sigma\right) \leq \frac{1}{k^2}, \text{ where } k > 0 \tag{4.7}$$

which states that for a random variable with finite expected value μ and variance σ^2, it is possible to compute a lower bound of the probability that values lie inside the interval $(\mu - k\sigma,\ \mu + k\sigma)$. For example, for $k = 3$, the interval contains at least 88.89% of the population, upper bounding false alarm rate to at most 11%. We should stress that the bound is however not tight: to confirm this, we report in Fig. 4.4, the false positive rate (along with precision, recall, and $F_{0.5}$) as a function of k_r, which is extremely low already for $k_r > 2$. To conservatively evaluate ODS, in the following we set $k_r = 3$.

Fading λ and Potential β factors

Both λ and β have a physical interpretation and play a key role in the model construction. λ is a time-related parameter that tunes the timescales at which *old samples should be considered as totally independent from the current system state*. Once λ is fixed, the potential factor β has a geometric interpretation, as *it determines the minimum number of samples needed for outlier mc promotion to core mc*. We point out that a domain expert could be tempted to select λ and β according to physical properties of the network, whereas a machine learning expert would select λ and β as a result of data analysis and a *grid search* procedure. We adopt both viewpoints in what follows.

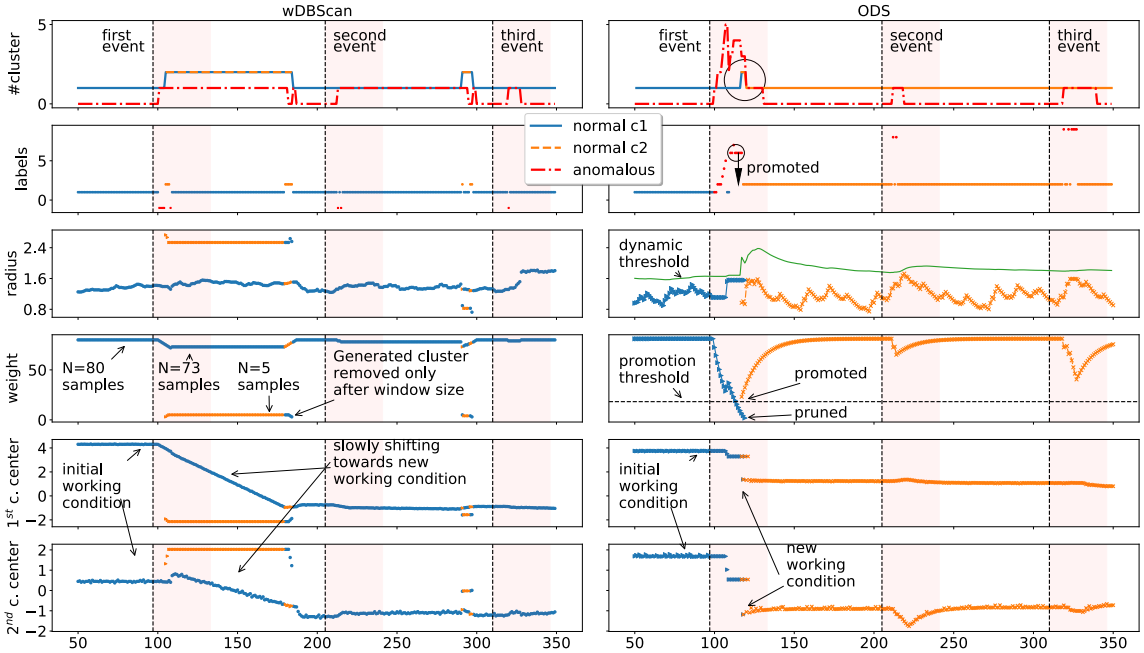
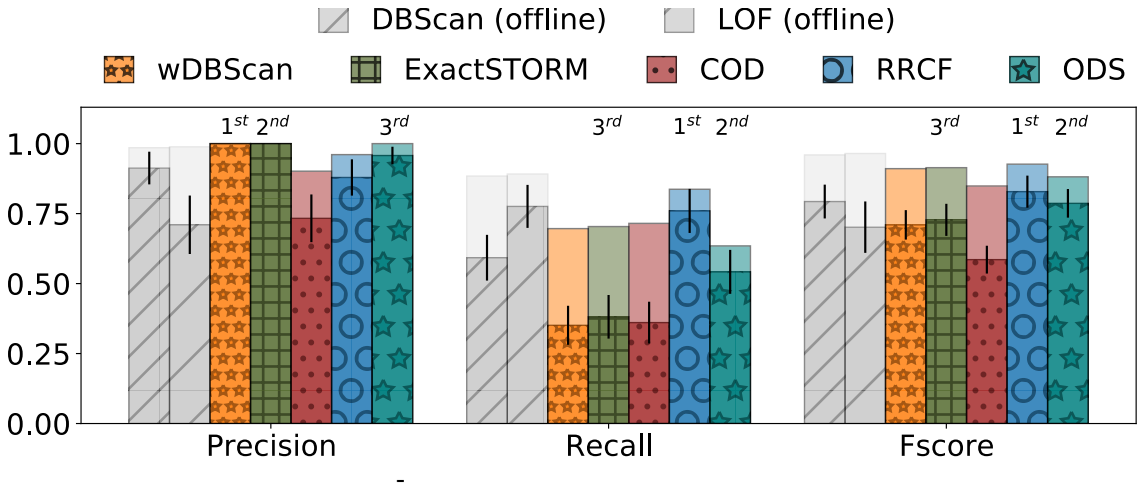


Figure 4.6: wDBScan (left) vs ODS (right) model evolution over time. The top two rows discriminate *normal* vs *anomalous* clusters: the top row reports the number of *normal* vs *anomalous* clusters returned by the models, the second row reports the cluster ID to which each samples is merged to. The third and fourth rows show the radius and the weight evolution of each *normal* cluster respectively, while the two bottom ones depict the evolution of the first and second components extracted from the center of the clusters.

For instance, a network domain expert could decide to require that an outlier *mc* should have at least 3 samples before becoming a normal *mc* and set λ according to the expected convergence time of the BGP protocol. For example, choosing λ such that a sample's contribution decays 99% after 5 minutes means $2^{-\lambda \cdot 60} = 10^{-2}$ or $\lambda \approx 0.111$ while the promotion threshold, requiring at least 3 samples, is $\beta \cdot \mu^+ = \sum_{i=0}^{3-1} 2^{-\lambda i} \approx 2.78$.

We perform a grid search to question these choices, reporting in Fig. 4.5 the $F_{0.5}$ score for varying $\lambda \in [0.01, 0.45]$ and $\beta \in [0, 1]$. Several important takeaways can be gathered from the figure. First, the performance metrics are smoothly varying over λ and β with a fairly large region (best $F_{0.5}$ score obtained for $\lambda \in [0.10, 0.16]$ and $\beta \in [0.2, 0.5]$ approximately).

Second, performances degrades for both increasing λ and β. A too high decay factor λ leads to giving too much importance to the most recent samples while a too high β, instead, translates in a too high weight threshold. We select $\lambda = 0.125$ and $\beta = 0.4$. Notice that with $\lambda = 0.125$, $\mu^+ \approx 12$ and for $0.2 < \beta < 0.5$ the *weight* promotion threshold ranges from $2.4 < \beta \cdot \mu^+ < 6$ which translates in clusters whose weight is composed by at least 3 to 8 samples and which are in line



-

Dataset	Algorithm	Precision		Recall		$F_{0.5}$score	
Mean Tuning (E2/3)	*DBScan*	0.99 ± 0.02		0.88±0.05		0.96±0.02	
	LOF	0.99±0.02		0.89±0.05		0.96±0.02	
	wDBScan	1.00±0.00	1st	0.70±0.07	3rd	0.91±0.03	2nd
	ExactSTORM	1.00±0.00	1st	0.70±0.07		0.91±0.02	2nd
	COD	0.90±0.06		0.72±0.07	2nd	0.85±0.06	
	RRCF	0.96±0.05		0.84±0.06	1st	0.93±0.04	1st
	ODS	1.00±0.00	1st	0.64±0.08		0.88±0.04	
Mean Test (E5/9/10)	*DBScan*	0.91±0.06		0.59±0.08		0.79±0.06	
	LOF	0.71±0.10		0.78±0.08		0.70±0.09	
	wDBScan	1.00±0.00	1st	0.35±0.07		0.71±0.08	
	ExactSTORM	1.00±0.00	1st	0.38±0.08	3rd	0.72±0.05	3rd
	COD	0.73±0.09		0.36±0.07		0.58±0.08	
	RRCF	0.88±0.06		0.76±0.08	1st	0.83±0.06	*1st*
	ODS	0.96±0.03	3rd	0.54±0.08	2nd	0.79±0.05	2nd

Figure 4.7: Algorithms Performance Comparison: Precision, Recall and $F_{0.5}$ score. Figure and table report the average performance on the *testing* (full opacity, foreground bars) vs *tuning* (light opacity, background bars) dataset. The top-3 among the stream algorithm are explicitly annotated.

with the expert domain choices. These findings confirm that indeed the hyperparameters of the algorithm respect their physical interpretation and can be set accordingly.

A last remark is worth making: while we have seen that performance smoothly varies on β and λ, we point out that their selection is still primarily correlated with the telemetry sampling rate: as such, for very different sampling timescales (eg. subsecond or minutes), a new sensitivity analysis is recommended.

4.5 Performance evaluation

We now carefully compare the methods, using the hyperparameters selected in the tuning phase on datasets (E2, E3), on previously unseen datasets (E5, E9, E10). We start by illustrating one example of execution of the system in Sec. 4.5.1. We systematically compare performance of the algorithms in terms of several information retrieval metrics in Sec. 4.5.2. We finally contrast algorithms in terms of complexity and execution times in Sec. 4.5.3.

4.5.1 Model evolution over time

To better illustrate intrinsic differences of stream-learning vs batch algorithms, we portray the evolution of two sample models. In particular, we select ODS and the windowed version of DBScan (wDBScan) and depict their anomaly detection processes with the help of Fig. 4.6, using the initial portion of dataset E5 as an example. In particular, we ought to recall that whereas in the ODS case a *single model* evolve over time, in wDBScan each of the different windows yield to a *different model*. In other words, in ODS model evolution over time is smooth by design, whereas in wDBScan the similarity between outputs of models that are run on consecutive input windows, merely stem from similarity in the input data, as the time component is not otherwise explicitly exploited. In spite of the above difference, it is possible to resort to consistent visualization of the main inner state of these algorithms, such as the number of normal vs outlier clusters, their radius size and center position, etc. that are reported in Fig. 4.6.

In wDBScan, each time the algorithm is run, a unique increasing ID is assigned to each cluster, starting every time from 0: each sample is assigned the ID of the cluster it is merged to, and the value -1 is reserved for outlier clusters. In ODS, a unique increasing ID is assigned to each newly generated *normal-mc*, and a separate ID space is similarly used to track *outlier-mc* clusters. Each sample is then assigned the ID of the cluster it is merged to, and differently from wDBScan, ID is *not* reset across runs.

Plots in the top row of Fig. 4.6 depict the time evolution of the number of normal vs outlier clusters present in the window (wDBScan) or of the number of *normal-mc* vs *outlier-mc* (ODS). We observe that, after the initialization phase, both the models are composed by a single *normal-cluster* and no *outlier* ones. It is easy to observe that, in correspondence to the anomaly injection

periods, the number of outlier clusters increase as expected.

The second plots row represents the ID of the clusters the samples are merged to. We observe that in both the models most of the samples are assigned a normal ID (generating a horizontal line). When an event is injected, algorithms flag samples as anomalous (red dots). In the case of ODS, outlier clusters may get promoted into normal clusters (which is visible, e.g., toward the end of the first event), whereas the old clusters are pruned over time due to the fading function.

The third and fourth rows illustrate the radius and the weight of the normal clusters respectively. One can easily observe from these plots the main differences in the model evolution of the two methods. On the one hand, after the injection of the first event, wDBScan generates a small cluster composed of 5 samples but, as soon as BGP convergence is achieved, returns merging new samples to the original cluster. The newly generated cluster, together with the anomalous samples, continue to be part of the model, even if not used for any purpose: i.e. no new samples are merged to the generated cluster as evidenced by the label ID (2nd plot), by the constant radius (3rd) and weight (4th) and also from the constant center (5th and 6th), which holds until they are excluded from the window. On the other hand, ODS removes old clusters (see the steady decrease of the weight of the cluster before pruning) and updates new ones (see the fast increase of the weight of the cluster after promotion) much more promptly than wDBScan.

The fifth and sixth rows show the first and second center components respectively, extracted through simple Principal Component Analysis (PCA) whose eigenvalues represents approximately 94% (wDBScan) and 77% (ODS) of the center. The two methods clearly behave differently: while ODS generates a new normal cluster (notice the pruning/promotion in the fourth row) in the region of the new working condition, wDBScan slowly drifts towards the latter. This effect depends on the hyperparameters ϵ and w whose modification would however lead to a decrease in wDBScan performance (recall Fig. 4.3).

4.5.2 Detection Performance

For validation, we rely on a set (E5, E9, E10) of experiments that are independent from those used for tuning (E2, E3), and are additionally well suited to stress test generalization capabilities of the studied methods. In particular, E5 is very similar to the tuning ones both in terms of traffic

load (1 Tbps) as well as anomalies (BGP leaks) so it allows one to test the performances of the models in scenarios similar to those of the tuning phase. *E5 thus constitutes a good stress test of the generalization capabilities of the selected hyperparameter.*

E9 and E10 instead undergo different traffic loads (2.9 Tbps and 2 Tbps respectively) as well as different injected anomalies (Port Flaps caused by *interface shutdown from terminal* and *link failures caused by fiber pull* respectively). The latter ones allows thus one to test the performance of the models in scenarios very different from those used in the tuning phase. Additionally, while the events injected in E5 are severe ones and affect the operational status of an entire node (i.e. the node losses all the BGP tables and is no longer able to forward data on all the interfaces) and all the direct neighbors, the events in E9 and E10 affect a single interface which results in the loss of only one out of the four links between two nodes. We expect a mitigation of the effects produced in the network by the anomalies injected in E9 and E10 and for this reason we expect not all the nodes to detect the events. *E9 and E10 constitutes a good stress test of generalization capabilities of the algorithm.*

We compare algorithmic performance using several information retrieval metrics, notably Precision, Recall and $F_{0.5}$score in Fig. 4.7 and Accuracy, Markedness and Informedness in Fig. 4.8. We report the mean and confidence interval of each metric for the 7 algorithms tested, explicitly contrasting tuning (light opacity, background bars) and testing datasest (full opacity foreground bars). The information is additionally tabulated in detail in the figure, and an explicit annotation of the algorithm rank (limited to stream algorithms) for each metrics is additionally reported to ease interpretation of the results.

From a high-level viewpoint, considering the testing datasets we observe that RRCF stand out as being ranked 1st on four metrics (Recall, $F_{0.5}$, Accuracy and Informedness) – which makes it a very good choice. ODS instead stand out as the only algorithm being systematically ranked 2nd or 3rd on all 6 metrics – which makes it a robust choice.

In more details, we observe that the hyperparameters selected in the tuning phase yield to reasonably good performance in the test datasets as well, albeit performance diminish for all methods. This behavior is expected, as we know already E9 and E10 contain anomalies which

Table 4.4: Detection peformance: in-depth RRCF vs ODS comparison

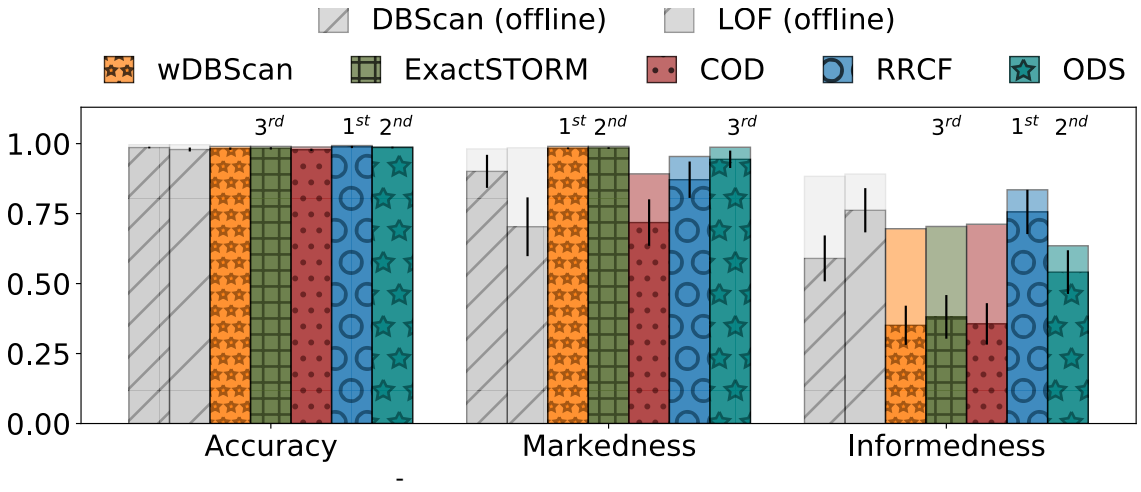
Metric	ODS	RRCF
Precision@3	0.899±0.063	0.898±0.021
Precision@5	0.776±0.088	0.786±0.024
Area Under the Curve (AUC)	0.967±0.016	0.989± 0.002
Average Precision (AP)	0.785±0.075	0.850± 0.017

are less disruptive and concern a single link, and are thus more difficult to detect. Discrepancy between tuning and testing phase is particularly visible considering the mean $F_{0.5}$ score over all datasets, where the degradation appears more severe for DBScan, wDBScan, LOF, ExactSTORM and COD. Investigating further, we find that DBScan and LOF performance degradation is due to low portability of parameter configuration as a function of the traffic load, which in turn has an impact on the distance between the samples. wDBScan, ExactSTORM and COD precision is consistent with that of the tuning phase, however, as a downside of the tradeoff, they achieve 0.35, 0.38 and 0.36 recall respectively.

In summary, RRCF and ODS appears to be a first and second choice respectively as for detection performance are concerned. Additionally, RRCF and ODS both appear to limit discrepancy with respect to the tuning phase, hinting to the fact that their hyperparameters hardly fall into overfitting. Interestingly, RRCF and ODS stand at opposite sides in the recall (RRCF is top-1) vs precision tradeoff (RRCF is not even in the top-3), which makes both of them interesting options. For instance, in the case of ODS, precision could be traded for recall decreasing k_r, however this would increase the number of false alarms, which is not desirable in operational settings in our opinion, as otherwise alerts are unreliable and would thus simply be ignored. To further appreciate the differences between RRCF and ODS, we additionally contrast further metrics such as Precision@K, area under the received operating characteristics curve (AUC) and average precision (AP) in Tab.4.4. From the comparison it emerges clearly that RRCF and ODS are practically indistinguishable for (at least) the top-5 events in each datasets, as shown by the Precision@3 and Precision@5 metrics, with RRCF exhibiting a better average precision over the whole set of anomalies.

4.5.3 Computational Complexity

With respect to SNMP polling in periods of 5 minutes, the 5 second sampling rate in the dataset considered in this work constitutes a 60$\times$ increase in the data velocity. In case future technolog-



Dataset	Algorithm	Accuracy		Markedness		Informedness	
Mean Tuning (E2/3)	*DBScan*	0.996±0.002		0.981±0.021		0.883±0.049	
	LOF	0.996±0.002		0.985±0.017		0.891±0.047	
	wDBScan	0.990±0.003	2nd	0.990±0.003	1st	0.696±0.069	3rd
	ExactSTORM	0.990±0.003		0.990±0.003	3rd	0.704±0.068	
	COD	0.988±0.003	3rd	0.892±0.064		0.712±0.065	2nd
	RRCF	0.993±0.003	1st	0.954±0.046		0.835±0.059	1st
	ODS	0.988±0.003	3rd	0.987±0.003	2nd	0.635±0.081	
Mean Test (E5/9/10)	*DBScan*	0.986±0.003		0.901±0.059		0.590±0.082	
	LOF	0.979±0.007		0.703±0.105		0.762±0.079	
	wDBScan	0.982±0.004		0.983±0.003	2nd	0.351±0.070	
	ExactSTORM	0.983±0.004	3rd	0.984±0.003	1st	0.381±0.078	3rd
	COD	0.978±0.004		0.718±0.083		0.356±0.074	
	RRCF	0.988±0.004	1st	0.871±0.065		0.756±0.079	1st
	ODS	0.986±0.003	2nd	0.944±0.031	3rd	0.541±0.078	2nd

Figure 4.8: Algorithms Performance Comparison: Accuracy, Markedness and Informedness. Figure and table reports the average performance on the *testing* (full opacity, foreground bars) vs *tuning* (light opacity, background bars) dataset. The top-3 among the stream algorithm are explicitly annotated.

ical releases will reduce the sampling period further (to subsecond timescales), a major limiting factor will be then represented by the processing capabilities of the device and collectors. Under this angle, it is clear that computational complexity is of uttermost importance from a deployment perspective.

Yet, most of the well known methods used today in the literature are in large part unsuitable to process data fast enough both because they are computationally complex and have heavy resources requirements. In this section we experimentally measure the time complexity of *stream* vs *batch* algorithms. We replicate E5 100$\times$ times, obtaining a stream composed by approximately 150K samples: by varying the length of the stream, we study the execution time trend of each

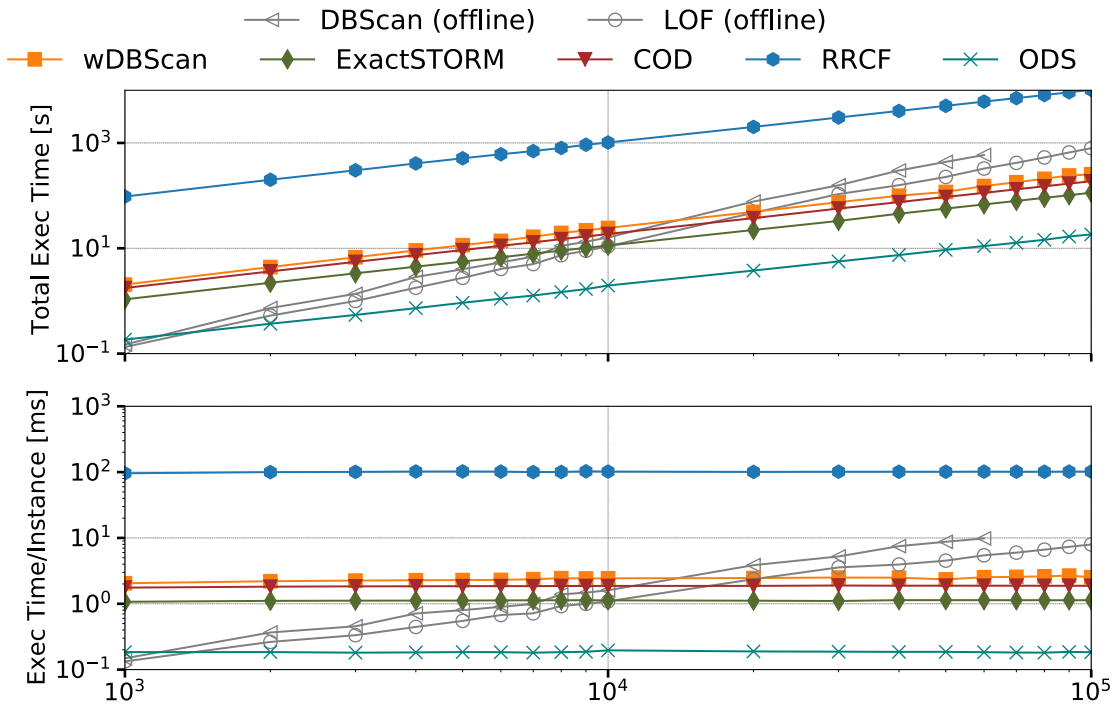


Figure 4.9: Algorithm complexity: total execution time in seconds (top) and execution time per instance (bottom) for DBScan, LOF, wDBScan, ExactSTORM, COD, RRCF and ODS as a function of the dataset size. DBScan measurements are not complete as it runs out of memory for values greater than those shown in the plot.

method.

All the experiments are run on a server equipped with Linux Debian 10, Intel Xeon E5-1620 with 3.60 GHz CPUs and 32GB RAM. The scripts, available at [2, 3], use Python 3.8.3 [145], numpy 1.19.1 [109], pandas 1.05 [99] and scikit-learn 0.23.1 [53]. We use moreover RRCF's python implementation 0.4.3 present at [22].

Fig. 4.9 reports as a function of the stream length, the total execution time (top) and the average processing time of a single instance (bottom). ODS is the fastest one due to its linear complexity followed by ExactSTORM, COD, wDBScan, LOF and DBScan. DBScan is the one demanding most resources and reaches the memory limit for streams longer than 60K samples.

The bottom plot shows the processing time per instance. While DBScan and LOF show increasing processing time per instance per increasing stream size, stream algorithms process the data in fixed amount of time per instance. Even by restricting our attention to stream algorithms only, ODS stands out as being significantly faster than the others, followed by ExactSTORM

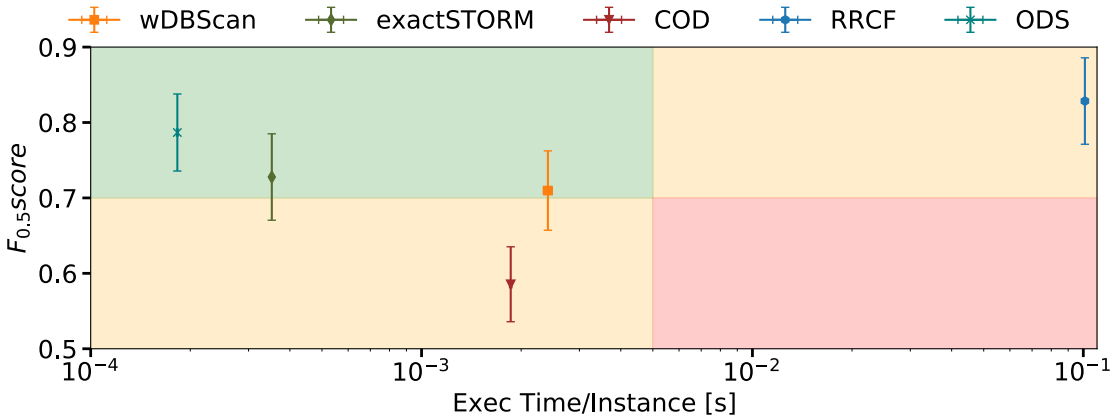


Figure 4.10: Execution time per Instance (x-axis) vs Detection performance $F_{0.5}score$ (y-axis). The plot is annotated with semi-planes to better highlight the desirable corners of the design space: ODS sits at an interesting operational point for being significantly faster than all the algorithms tested and second only to RRCF in terms of information retrieval metrics ($F_{0.5}$ in this plot).

(2×slower), COD (10×), wDBScan (13×) and RRCF (550×). By comparing the processing time and the sampling rate it is possible to establish an upper bound of the maximum sampling rate. Observing that the elapsed time per-sample is roughly 0.20 ms, our released ODS Python implementation is able to process approximately 5000 samples per second. Conversely, RRCF execution time requires about 100ms per sample, which caps its processing rate to at most 10 samples per second.

We summarize the complexity vs detection performance tradeoff in Fig.4.10 as a scatter plot of the execution time (in second, on the x-axis) vs the $F_{0.5}$ score performance (y-axis). Note that we use xyerrorbars, but the execution times confidence interval is very tight, and thus not visible due to the logarithmic x-axis scale. The plot is annotated with semi-planes (split so to halve the x-axis and y-axis ranges), to better highlight the desirable corners of the design space: top-left corner indicates algorithms that are both fast and good (green shading), whereas top-right corner indicates good but slow algorithms (yellow shading), bottom-left indicates fast but poorly performing algorithms (yellow shading) and finally bottom right indicates slot execution and poor performance (red shading). The picture clearly show that ODS sits at an interesting operational point for being significantly faster than all algorithms tested, and second (but nevertheless very close to) only to RRCF in terms of information retrieval metrics (as per Tab.4.4).

4.6 Discussion

We have contrasted a number of clustering methods for anomaly detection in networks. Whereas results testify stream-based approaches to be of interest, we aim at discussing here limitations and caveats to avoid pitfalls in their deployment.

Contextual anomalies. First, DenStream and consequently ODS are based on euclidean distances. We expect ODS to work on anomalies constituted by points *far* in the space, from the *normal* clusters and under no circumstances we do expect it to be able to detect *contextual* anomalies (e.g., such as absence of a periodic peak in a periodic signal). This suggests that techniques such as those studied here, should be complemented by others shall contextual anomalies be relevant in the deployment scenario.

Curse of dimensionality. The algorithm presented is of course not immune to the curse of dimensionality. This is likely to happen in practice whenever one would attempt to build a single model, aggregating several nodes and features per node. At the same time, model execution is extremely lightweight, which would allow to run multiple models in parallel, either at node-level (reducing communication complexity, but possibly missing events not detectable from internal measurements) or at feature-subset level (which would require some amount of communication between nodes, but possibly exploiting correlation among features at neighboring nodes).

Hyperparameters tuning. We have observed that hyperparameter tuning can lead to overfit, making deployment of unsupervised techniques difficult in practice, especially for methods whose hyperparameters are closely related to the dataset (e.g. *contamination* in LOF or ϵ neighborhood in DBScan).

Instead, even though DenStream (and thus ODS) relies on four hyperparameters, we have seen that it is possible to reduce their number by lumping some (e.g. by setting $\mu = \mu^+$) and by dynamically setting others (e..g, so that the radius threshold $\epsilon = \bar{r} + k_r\sigma_r$ contains the bulk of the radius distribution).

At the same time, the fading factor λ (reduces gradually the importance of the samples) and the potential factor β (delimits the size of a *normal* cluster) must be selected with care. In particular, they are indeed bounded by $w > \beta/(1 - 2^{-\lambda})$ and therefore must be set taking into account their relationship and physical interpretation. For example, once λ is set, choosing a too high β could

lead to the degenerate case in which the weight threshold is too high, and the model fails to build and maintain *normal-mcs*.

Autonomicity level ODS is not intended to completely replace a human network operator, but on the contrary it is a tool designed to facilitate his job. For instance, while ODS can update the model over time, it has to be initialized by a conscious operator, providing anomalous-free samples at at bootstrap. In turn, while ODS operates in the unmodified feature domain, there is a further need of explainable attention focus mechanisms [1] to let the human operator focus on the important features that triggered an event detection, an aspect orthogonal to our work.

4.7 Conclusion

The recent emergence of model-driven telemetry opens new challenges for anomaly detection, and particularly makes the use of stream-based unsupervised machine learning tools very appealing. In this paper we develop, implement and open-source the ODS anomaly detection engine, based on the online clustering algorithm DenStream. We thoroughly analyze ODS on datasets gathered on BGP-only datacenter network loaded with up to 3 Tbps aggregated traffic, and extensively compare it with a set of batch (DBScan, LOF), windowed (wDBScan) and stream techniques (ExactSTORM, COD and RRCF).

Our results show that despite ODS is apparently plagued with several hyperparameters inherited from DenStream, their selection is quite straightforward, and performance are robust to inner hyperparameter selection. Additionally, ODS is significantly faster than any of the tested algorithms, and second only to RRCF (yet very close to it) in terms of detection performance. Overall, the above results suggest ODS as a particularly lightweight and suitable algorithm for stream-mode real-time network anomaly detection.

4.7.1 Available features

The available features are uniquely identified via their full YANG name, which in turn derives from the concatenation of an an *EncodingPath* with a *Leaf* among the available ones for that path (*Leafset*). Tab. 4.5 reports the full list of features used, that can be ascribed to either Data Plane (DP), for EncodingPaths matching `infra-statsd-oper` and `fib-common-oper` categories,

Table 4.5: Available telemetry features

(CP) EncodingPath = Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ ip-rib-route-table-name/protocol/bgp/as/information

Leafset = {
active-routes-count
backup-routes-count
deleted-routes-count
paths-count
protocol-route-memory
routes-counts
}

(CP) EncodingPath = Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance /instance-active/default-vrf/ process-info

Leafset = {
global_established-neighbors-count-total
global_neighbors-count-total
global_nexthop-count
global_restart-count
performance-statistics_global_configuration-items-processed
performance-statistics_global_ipv4rib-server_is-rib-connection-up
performance-statistics_global_ipv4rib-server_rib-connection-up-count
performance-statistics_vrf_inbound-update- messages
vrf_neighbors-count
vrf_network-count
vrf_path-count
vrf_update-messages-received
}

(DP) EncodingPath = Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/ generic-counters

Leafset = {
bytes-received
bytes-sent
}

or Control Plane (CP), for EncodingPaths matching the `ip-rib-ipv4-oper` and `ipv4-bgp-oper` categories. Extracting the full list of features per each interface and node yield about 750 data-plane and 25 control-plane features overall per dataset.

4.7.2 Selected features

The full set of available features contains (i) redundant and totally correlated features (e.g. *free-memory* and *occupied-memory*), as well as (ii) categorical features design (e.g. *af-name*, *as* etc.) or (iii) features that are constant in our experiments (e.g., *output-queue-drops*). Out of the available features, we thus discard duplicated, categorical or constant features. Based on this process, we ultimately retain 64 DP and 18 CP features, for a total of 82 features. Notably DP features relate to the generic counters (i.e., whose EncodingPath match `interface/generic-counters`), whereas CP features relate to `bgp/as/information` and `vrf/process-info` EncodingPaths, which have proven useful for BGP anomaly detection in previous studies [18].

Chapter 5

AutoAD

In this chapter we describe `autoAD` an automated unsupervised Anomaly Detection framework which selects for a given unlabeled dataset the best performing model, as well as its optimal configuration for its hyperparameters. We analyze the most recent studies of automated anomaly detection in Section 5.2 and introduce the proposed strategy in Section 5.3. The evaluations are done in Section 5.4 while a summary of our finding and remarks are given in Section 5.5.

5.1 Introduction

Over the last decade, we witnessed the proliferation of several machine learning algorithms capable of solving different tasks for the most diverse applications. Often, for an algorithm to be effective, significant human effort is required, in particular for hyperparameter tuning and data cleaning. Recently, there have been increasing efforts in alleviating such a burden, with the goal of making machine learning algorithms easier to use for researchers with varying levels of expertise. Nevertheless, the question of whether an efficient and a fully generalizable automated Machine Learning (autoML) framework is possible remains unanswered. We present `autoAD`, an autoML framework for unsupervised anomaly detection. By leveraging a pool of different anomaly detection algorithms (called ensemble), each one coming with its own hyperparameter search space, our framework automatically selects the best performing approach, while determining an optimal configuration for its hyperparameters on a given dataset. Our extensive experimental evaluation, conducted on a rich collection of datasets, shows the substantial gains that can be achieved with `autoAD` compared to existing autoML methods such as *SelectV*, *BoostSelect* and

the *Naive* method.

Often, to unravel the full potential of machine learning, some human expertise and domain knowledge are required in order to select the most effective algorithm, to tune its hyperparameters, as well as to perform data cleaning (feature selection, dimensionality reduction, etc.). This is often a tedious and non-trivial task for researchers with all levels of expertise. To alleviate such a burden for researchers, several techniques have been proposed under the *automated Machine Learning (autoML)* research direction [26, 77]. One of the goals of autoML is to select the most effective model and its best configuration for its hyperparameters, on given a dataset.

Currently, most efforts in autoML are devoted to supervised machine learning, with very few studies dealing with unsupervised tasks. Recently, [127] proposed a method called *Vertical Selection (SelectV)* that exploits correlation analysis among scores from different methods to select the best algorithms. *SelectV* first unifies the scores of all the considered methods into a unique *target* vector (treated as *pseudo ground-truth*). Subsequently it builds a new ensemble containing the methods that better correlate to *target*. [31] extends the correlation analysis introduced by *SelectV* using a boosting strategy and present *BoostSelect*. Their idea is to perform boosting (change weights into the correlation analysis) upon the inclusion of a new member into the ensemble. *BoostSelect* tries to reduce the importance of those instances (outliers) that have already been identified by any ensemble member to promote diversity among chosen methods.

Such methods however are limited by their initialization: the unification of the scores into a *target* vector. Consider, for example, an ensemble composed mostly by poorly performing models with only a few well performing ones. In such a scenario the *target* is composed largely by poorly performing methods. As a result, the well performing algorithms weakly correlate to the target and might be discarded into the correlation analysis. As a consequence we observe that, in this scenario, the *Naive* ensemble (all methods are equally weighted) outperforms both *SelectV* and *BoostSelect*. Moreover, as stated by the authors, *BoostSelect* performance strongly depends on the choice of the input hyperparameters on some datasets.

To tackle the aforementioned limitations, we propose `autoAD`, an automated unsupervised Anomaly Detection framework which selects for a given unlabeled dataset the best performing

model, as well as its optimal configuration for its hyperparameters. Instead of using correlation techniques, we analyze the output of each method individually searching for evidence of proper outlier detection. In this direction we focus on the most anomalous instances and test if the dataset is easier to compress by removing them. Motivated by the established supervised autoML frameworks, *e.g.*, , `Auto-WEKA` [85] and `Auto-Sklearn` [55], the proposed `autoAD` framework involves different outlier detection algorithms and a definition of their corresponding hyperparameter search spaces. Given an input dataset, `autoAD` applies the algorithms with their different hyperparameter configurations in parallel, afterwards it evaluates the performance using the unsupervised evaluation strategy mentioned before.

The main contributions of this chapter are summarized as follows:

- We develop `autoAD`, a framework for automated unsupervised anomaly detection, which, to the best of our knowledge, represents the first autoML approach for unsupervised anomaly detection which does not use correlation analysis.
- We propose an unsupervised metric strategy that permits the evaluation of anomaly detectors in a fully unsupervised way by removing outliers and using statistical measures, such as variance, on the remaining normal instances.
- We conduct an extensive experimental evaluation on a diverse set of datasets which shows that `autoAD` achieves significantly better performances than other proposed methods.
- To foster reproducibility, the code and datasets employed in our work are available at `http://bit.ly/3mdT6qu`.

The reminder of this work is organized as follows. In Section 5.2, we detail related work. Section 5.3 presents the description of our proposed framework and the employed methodology for unsupervised anomaly detection evaluation. In Section 5.4, we show the experimental results and discussions. We finally end this chapter by drawing conclusions and future directions in Section 5.5.

5.2 Related Work

The performance of a given machine learning method depends on the quality of the algorithm as well as its hyperparameterization, a task which is sometimes difficult to fix to the optimal values. AutoML [26, 77] is a new topic that supports researchers and practitioners with the tedious work of manually designing machine learning pipelines, which include performing algorithm selection and tuning hyperparameters. AutoML can be also viewed as the process that makes machine learning easier by avoiding manual hyperparameters tuning for both machine learning experts and non-experts.

5.2.1 Supervised Automated Machine Learning

This very hot topic has attracted several researchers during the recent years due to the importance of its application. In fact, the basic autoML algorithms have been initially proposed for the supervised learning and are discussed in recent surveys on autoML and its open challenges [48, 54, 73]. Examples of well-known autoML approaches are (i) *irace* [94] that uses an iterated racing procedure where the worst configurations are replaced by new ones for each iteration (race); (ii) *SMAC* [76] that performs a Bayesian optimization in conjunction with a simple racing mechanism on the instances to efficiently decide which of two configurations performs better; and (iii) *ParamILS* [75] which is based on an iterated local search that starts by evaluating the default and some other configurations on a subset of instances, then the best configuration will be maintained and tested on a different subset of data.

Throughout the last five years, multiple tools and systems have been developed which serve autoML [73, 81, 83]. For instance, `Auto-Sklearn`[1] [55] and `Auto-WEKA`[2] [85] which are two automated systems that have been implemented on top of the well-known machine learning softwares `scikit-learn` and `WEKA`, respectively. These tools and techniques exclusively deal with supervised methods where ground truth labels are used during the model selection and the hyperparameters tuning processes.

[1] https://www.automl.org/automl/auto-sklearn/
[2] http://www.cs.ubc.ca/labs/beta/Projects/autoweka/

5.2.2 Unsupervised Automated Machine Learning

For what concerns unsupervised methods for automated anomaly detection we observe two different branches in tackling such problem. On one hand previous works have investigated the use of meta-learning [146, 30, 157] techniques. Such methods, start by extracting meta-data from the datasets and then, given a test dataset, search for the most similar dataset and its corresponding best performing algorithm. Meta-learners thus require a collection of historical anomaly detection datasets and historical performances of the models on such datasets to map models with meta-data.

On the other hand, completely unsupervised strategies for anomaly detection do not require any training phases or labels. Both the strategies have pros and cons: on one hand meta-learners are extremely fast as they require an offline tuning phase which is not critical and an online prediction phase in which meta-data is extracted from the test dataset. However, such methods might suffer when used on test datasets with relatively low similarities with respect to training datasets. Moreover, collecting new datasets and labels could result to be expensive in terms of human effort. On the other hand, completely unsupervised methods are very slow compared to meta-learners. They require indeed to run all the algorithms on the test dataset, process outputs and aggregate results. However, being completely unsupervised they do not require prior training and could perform well also on unseen datasets. In this study we consider only completely unsupervised models leaving unsupervised vs meta-learning comparison for future work.

Among the completely unsupervised strategies for automated selection of methods and hyperparameters we cite *SelectV* and *BoostSelect*. They are both based on correlation analysis of the output scores.

Vertical Selection (SelectV) [127]: Selects the ensemble components through correlation analysis among the score lists from different methods. Given a set of anomaly score lists *S*, SelectV first averages the probability scores across lists to construct a *target* vector, known as *pseudo ground-truth*. Subsequently, it initializes a new ensemble *E* with the list $l \in S$ that has the highest weighted Pearson correlation to *target*. In computing the correlation, the weights used for the list elements are equal to $\frac{1}{r}$, where *r* is the rank of an element in target when sorted in descending order, i.e., the more anomalous elements receive higher weight. Next, *SelectV* sorts the remain-

ing lists $S \setminus l$ in descending order by their correlation to the current "prediction" of the ensemble, which is defined as the average probability of lists in the ensemble *E*. The method tests whether adding the top list to the ensemble would increase the correlation of the prediction to target. If the correlation improves by this addition, the ensemble is updated and the remaining lists are reordered by their correlation to the updated prediction. As such, a list gets either included or discarded at each iteration until all lists are processed.

BoostSelect [31]: Similarly to *SelectV*, it constructs a *target* vector by combining the scores of all available components in the ensemble. From this target vector, the algorithm preliminarily assumes the top $\lfloor n \cdot t \rfloor$ objects (ranked by their combined score) to be outliers, where n is the dataset size and $0 < t \ll 1$ is a hyperparameter capturing the expected percentage of outliers in the dataset (i.e., there are $K = \lfloor n \cdot t \rfloor$ outliers assumed to be present). The target vector thus becomes a binary vector, listing 1 for outliers and 0 for a inlier and serves as pseudo ground truth for the boosting approach. *BoostSelect* sets then the weights for Pearson correlation to $\frac{1}{2K}$ for outliers and $\frac{1}{2(n-K)}$ for inliers. Such values are only the initial weights as they will be updated by the boosting procedure. The potential ensemble members are sorted according to their weighted Pearson correlation to the target vector. The candidate that is most similar to the target vector is chosen as the first ensemble member. Remaining potential ensemble members are iteratively resorted in ascending order according to their similarity to the current prediction of the ensemble. Potential members are included if their inclusion would increase the similarity of the ensemble prediction to the target vector, otherwise they are discarded. After the insertion of a potential member into the ensemble the boosting procedure is performed. This phase reduces the weights by the input hyperparameter $0 < d < 1$ (drop rate) of outliers that have already been identified by any ensemble member.

5.3 The Proposed `autoAD` Framework

In this section, we present an automated unsupervised anomaly detection framework that aims to find the best performing algorithms for a given unlabeled dataset by weighing them using quality metrics measurements.

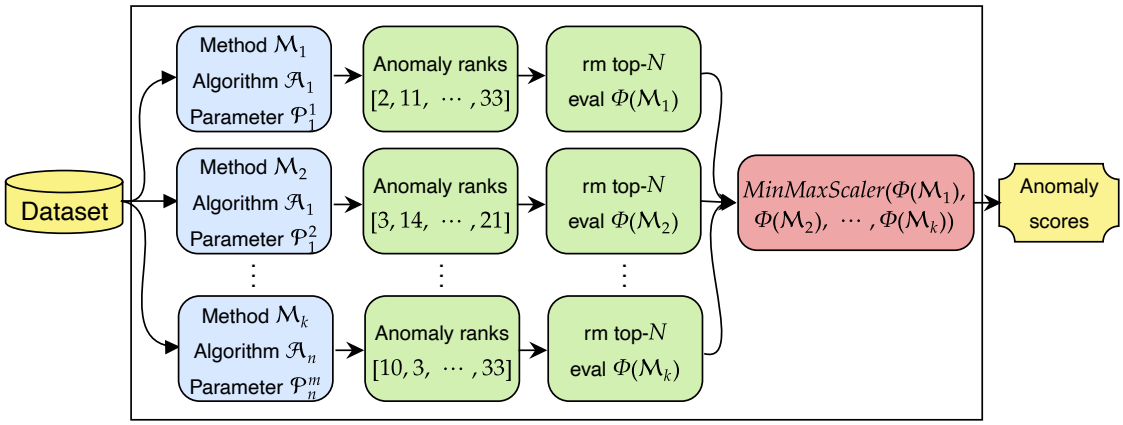


Figure 5.1: Overview of the `autoAD` framework. Given the input dataset and the set of algorithms with their corresponding predefined search space, we apply separately each method, as the combination $(\mathcal{A}_i, \mathcal{P}_i^j)$. Once we finish processing each of the method, we rank the output anomaly scores (as shown in the first green column cells) in order to remove (*rm*) the top-N anomalous instances. We therefore evaluate (*eval*) the performance of each method using a quality measure, ϕ, on the remaining normal instances. We assign to each method a weight proportional to its performance. Hence, the final anomaly scores are computed based on the initial scores and the weight assigned to each method.

An effective automated framework for unsupervised anomaly detection should be composed of a set of different unsupervised detectors with distinct configurations. The proposed `autoAD` framework consists in four key steps: (i) application of the anomaly detection algorithms where each one outputs the anomaly scores of instances given a dataset; (ii) removing the top N anomalous instances, *i.e.*, instances with the highest anomaly scores; (iii) assigning a weight to each algorithm using a given evaluation metric; and (iv) obtaining the final anomaly scores for each instance. An overview of `autoAD` is given in Fig. 5.1.

5.3.1 Application of Anomaly Detection

Each outlier detection method in our automated framework is composed of the algorithm and its hyperparameter configuration. Let $(\mathcal{A}, \mathcal{P})$ be the method $\mathcal{M}$ that uses algorithm $\mathcal{A}$ with hyperparameters $\mathcal{P}$. We define $\mathbb{M}$ the set of methods which is composed of $k = |\mathbb{M}|$ combinations of each algorithm tuned with each of its corresponding hyperparameter as follows:

$$\mathbb{M} = \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_k = (\mathcal{A}_1, \mathcal{P}_1^1), (\mathcal{A}_1, \mathcal{P}_1^2), ..., (\mathcal{A}_n, \mathcal{P}_n^m) \;\; \forall \mathcal{A}_i \in \mathbb{A}, \mathcal{P}_i^j \in \mathbb{P},$$

where $\mathbb{A}$ is a set of anomaly detection algorithms and $\mathbb{P}$ is a set of predefined hyperparameters

for the algorithms. Each algorithm $\mathcal{A}_i$ is used with a different hyperparameter configuration j in our framework.

Given an input unlabeled dataset, each method in the set $\mathbb{M}$, $(\mathcal{A}_i, \mathcal{P}_i^j)$, is simultaneously used to build a model and accordingly compute the output anomaly score of each instance.

In the following steps, we gradually present our key strategy in evaluating unsupervised detectors in a fully unsupervised way that mostly corresponds to performance in terms of AUC/AP criteria (see empirical studies in Section 5.4). In this framework, we used several algorithms discussed in Section 2.6 such as Random Histogram Forest (RHF) [119], Isolation Forest (ISO) [93], Probabilistic Principal Component Analysis (PPCA) [144], Histogram-based Outlier Score (HBOS) [61], K-NN [20] and Local Outlier Factor (LOF) [28] obtaining so $\mathbb{A} = \{RHF, ISO, PPCA, HBOS, K-NN\}$. More algorithms can be easily added with their hyperparameters' space to our `autoAD` framework.

5.3.2 Anomalies Removal

Anomaly algorithms assign a score to each instance in a given dataset that can be used to rank instances depending on their level of outlierness, *i.e.*, the anomalies are ranked before the non-anomalous instances (the most anomalous instance has the biggest score and is ranked first). Algorithm 3 presents the pseudocode of how we remove anomalies. In fact, the score result obtained from each method in `autoAD` will be ranked from the highest to the smallest one (line 1, Algorithm 3). In this work, we propose a quality measure that does not require data with

Algorithm 3: AnomaliesRemoval(X, ms_i, R)

Input: X: dataset ; ms_i: anomaly scores of method i; R: list of N-top ranked instances to remove drawn from $\mathcal{U}(0, 10\%)$
Output: Quality measure ϕ_i

1: Ranks = SortRank(ms_i)// `Sorted instances according to their anomaly scores in descending order`
2: $\phi_i = 0$
3: **for** all $N \in R$ **do**
4: $X_{filtered}$ = filter(X $\setminus Ranks$[1:N]) // `Remove top` N `most anomalous instances from` X
5: ϕ_i += QualityMeasure($X_{filtered}$) // `Compute the quality measure on the filtered dataset and aggregate over` R
6: **end for**
7: return ϕ_i

known labels. This evaluation strategy starts by removing the top N anomalous instances and

evaluates the remaining "normal" instances using a quality metric. Some questions naturally arise: *How many instances of the top ranked ones are actually anomalies? and how can we fix* N*?* This can be tricky as it involves a hyperparameter that controls the number of anomalies. In the unsupervised context, we assume that the right percentage of anomalies in a given dataset is unknown. An envisaged solution to answer these questions consists in picking a random value

$$N = \mathcal{U}(0, 10\%),$$

and removing the N top ranked instances, where N is between $[0, 10\%]$.

To avoid picking an unreasonable value, this process is repeated $r = 100$ times (line 3, Algorithm 3), *i.e.*, we do 100 runs and randomly select N for each time. In the end, we average the results of the different runs to appropriately capture the anomalies.

Once we remove the N-top ranked anomalies (line 4, Algorithm 3), we separately compute the performance of all the methods (line 5, Algorithm 3) using a quality measure ϕ, such as variance or Error Sum of Squares (SSE), on the remaining – *supposed to be* – normal non-anomalous instances, as depicted in Fig. 5.2. The final value of ϕ, for each method in our `autoAD`, is used to normalize the instances' score obtained in Section 5.3.1.

Quality metrics. Several quality metrics can be defined and used in the `autoAD` framework. As there are many types of anomalies (*e.g.*, *global*, *local* or *contextual*), it is possible to design quality metrics whose goal is to target a particular kind of anomalies. In this work we focus mainly on *global* anomalies that usually lie in the tails of the data distribution. Starting from these characteristics, we define two quality measures that serve the automated method to understand which method better compresses the data after removing the N-top ranked instances.

SSE: The simplest measurement one can compute on the – supposed to be – normal instances is the SSE. The method whose SSE measurement is the smallest have to be considered the best as it better compresses the data after removing the anomalous instances. By computing such a measure, we assume that normal instances are drawn from a unique cluster (not always true). Moreover, SSE can be weak when dealing with high dimensional datasets.

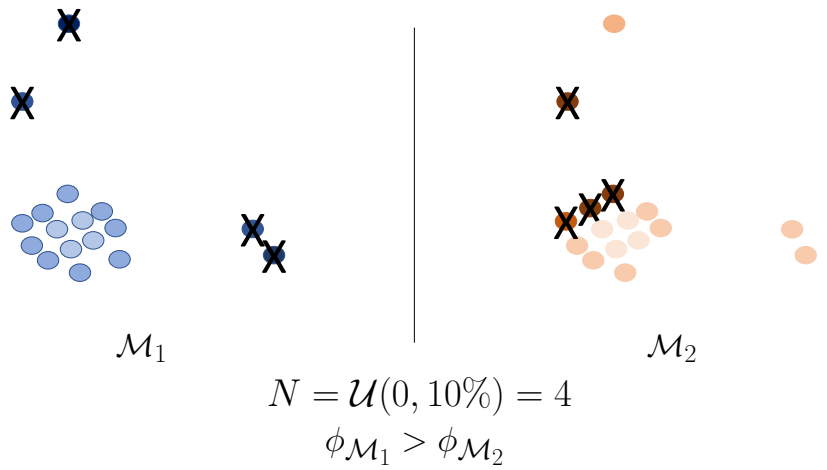


Figure 5.2: *Anomalies Removal* example applied on two different methods $\mathcal{M}_1$ and $\mathcal{M}_2$. The two methods produce different anomaly scores and ranks as illustrated by the color of each instance (darker colors indicate higher anomaly scores). By removing, for example, $N = 4$ most anomalous instances and applying a quality metric on the remaining instances $\mathcal{M}_1$ proves to be better than $\mathcal{M}_2$.

VAR: The variance is one of the most used dispersion metrics. Unlike the previous measure, the variance is dimension-wise and can be obtained by computing it on each dimension and aggregating the scores. Similarly to the previous case, the method whose aggregated variance is the smallest, after removing the N-top ranked anomalies, has to be considered the best method.

5.3.3 Algorithm Weighting

Each method in $\mathbb{M}$ produces its own measure ϕ obtaining so

$$\Phi = \{\phi_1,\ \phi_2,\ \phi_3,\ \ldots\ \phi_k\}.$$

The best method in the set is the one obtaining the best quality measure (*e.g.*, the method whose variance is the smallest after removing the N-top ranked instances) while the worst one is the one obtaining the worst quality measure.

A weight proportional to the measure ϕ_i is assigned to each method $\mathcal{M}_i$ by normalizing (min/max) the set of measures Φ.

$$W = MinMaxScaler(\Phi) = \{w_1,\ w_2\ w_3\ ...\ w_k\}.$$

The best quality measure (*e.g.*, the minimum variance) originates the best weight $w_i = 1$, the worst one $w_i = 0$ while the remaining ones $w_i \in [0, 1]$.

5.3.4 Final Anomaly Scores

The final anomaly scores take into account the initial anomaly scores of each method in $\mathbb{M}$ and the weights assigned by the quality measure in the previous step. As the output anomaly scores of each algorithm can be homogeneous and represented in very different ranges we scale all of them between 0 and 1. Subsequently, the anomaly scores of each method undergo the weighing process in which the weights produced in the quality measure phase are used. The final anomaly score is obtained aggregating the results over all the methods.

$$S = \sum_{i=1}^{|\mathbb{M}|} ms_i w_i,$$

where ms_i corresponds to the initial anomaly score of method i while w_i corresponds to its weight.

5.4 Experimental Evaluation

We conduct an extensive experimental evaluation to assess the effectiveness of our approach. All algorithms developed in our work are publicly available, so as to foster reproducibility [3]. We implemented our algorithms in Python 3.8.

5.4.1 Datasets

We consider a diverse set of datasets coming from different data sources, with different size and anomaly ratio. Most of them have been widely used as benchmarks when evaluating anomaly detection algorithms. In particular, we consider 16 datasets that are publicly available at the UCI [47] or ODDS [126] repositories. The size of our datasets range from 351 to 623091 instances (n), while the number of dimensions vary from 3 to 274 (d). The anomaly ratio is between 0.03% up to 35.9%. Similarly to [119], we use well-known datasets such as *ionosphere, arrhytmia, satellite,*

[3] The source code and datasets employed in our analysis can be found at `http://bit.ly/3mdT6qu`.

dataset	n	d	anomalies(%)	dataset	n	d	anomalies(%)
ionosphere	351	33	126 (35.9%)	shuttle	49097	9	3511 (7.15%)
wbc	378	30	21 (5.56%)	smtp29	96554	29	1183 (1.23%)
arrhytmia	452	274	66 (14.6%)	http_distinct	222027	3	75 (0.03%)
cardio	1831	21	176 (9.61%)	mulcross	262144	4	26214 (10.0%)
musk	3062	166	97 (3.17%)	cover	286048	10	2747 (0.96%)
satellite	5100	36	75 (1.47%)	http_logged	567498	3	2211 (0.39%)
satimages	5803	36	71 (1.22%)	kdd99	620098	29	1052 (0.17%)
mnist	7603	100	700 (9.21%)	http29	623091	29	4045 (0.65%)

Table 5.1: Overview of the datasets. For each dataset, we have the number of instances n, number of dimensions d, and number of anomalies (in $\%$).

mnist, shuttle, mulcross and some extracted from the *KDD99* dataset. Table 5.1 presents a brief summary of the datasets used.

5.4.2 Methods and Hyperparameters

In our approach `autoAD`, we use RHF, ISO, PPCA, HBOS, LOF and K-NN algorithms as the main anomaly detection engines. As RHF and ISO have consistently proven to be one of the most effective algorithms for unsupervised anomaly detection [119, 45, 49] we consider 8 different values for their main hyperparameter, namely, *sampling size* ψ for ISO and *maximum height* h for RHF. In both cases, we employ the same number of trees $t = 100$. Based on hyperparameters range, authors' insights and considerations done in the original paper of the two algorithms, we select $h \in [1, 8]$ for RHF and $\psi \in [32, 64, 128, ..., 4096]$ for ISO. For what concerns the remaining algorithms we use $K = \sqrt{n}$ as suggested by the authors as input hyperparameter for HBOS while using $K = 50$ neighbors for proximity based methods K-NN and LOF. We consider thus an initial ensemble of 20 different methods which are selected from all the three classes of algorithms: *Proximity*, *Probabilistic* and *Ensemble/Isolation* based.

We compare `autoAD` against the *Naive* ensemble (in which all methods are weighted equally) and also against the two previously described methods *SelectV* and *BoostSelect*. The latter uses the hyperparameters suggested by authors which are drop rate *d = 0.75* and threshold *t = 5%*.

Method	Median Gain	$\{RHF + ISO\}_{dft}$			
		Win Rate	Med Gain	Loss Rate	Med Loss
autoAD-SSE	+18%	43.75%	+27.87%	37.5%	-1.6%
autoAD-VAR	+23%	31.25%	+35.92%	31.25%	-1%

Table 5.2: The overall median gain of autoAD using the three different quality measures with respect to *SelectV*, the second best performing method. The Win/Loss rate indicates the number of datasets in which autoAD is statistically better/worse. The Med Gain/Loss indicates the median AP gain in the winning/worsening datasets.

5.4.3 Results

All the findings in this section are reported as a result of 30 independent runs for each of the considered strategies: *Naive*, *SelectV*, *BoostSelect* and the *AnomaliesRemoval* function discussed and presented in Algorithm 3.

We first show in Table 5.2 a summary of the results achieved by autoAD using the two quality metrics SSE and VAR versus *Naive*, the second best performing method. Table 5.3 reports subsequently a full comparison of all approaches and datasets considered indicating the mean AP score complemented with a 0.95 confidence interval. The best method is awarded using the two sample Kolmogorov-Smirnov test ($\alpha = 0.05$) [136] under the null hypothesis that the two distributions are identical while the Welch's two-tailed t-test [149] is used to test if they have the same mean.

From Table 5.2 we observe that both SSE and VAR quality measures produce a median gain of 18% (SSE) and 23% in terms of AP with respect to *Naive*, the second best strategy. Considering, on one hand, the datasets in which the proposed method is better than the *Naive* one, we observe a median AP gain of at least 28% (SSE) up to 36% (VAR). On the other hand, when the latter performs better that autoAD the median AP loss is limited to less than 2%.

Such gains and losses have to be found in Table 5.3 which reports the results for each dataset and strategy. For example, in the *kdd99* dataset, *Naive*, *BoostSelect* and *SelectV* strategies reach a very close AP score of 0.594 ± 0.005, 0.559 ± 0.011 and 0.606 ± 0.005 respectively while both autoAD-SSE and autoAD-VAR improve the scores to 0.756 ± 0.011 and 0.808 ± 0.009 respectively. Similar results can be observed in the *mulcross* dataset in which the first three strategies obtain

	autoAD-SSE	autoAD-VAR	Naive	SelectV	BoostSelect
ionosphere	0.808 ± 0.001	0.809 ± 0.001	0.801 ± 0.001	0.803 ± 0.002	0.789 ± 0.005
wbc	0.583 ± 0.004	0.585 ± 0.005	0.595 ± 0.004	0.593 ± 0.004	0.571 ± 0.01
arrhytmia	0.461 ± 0.003	0.456 ± 0.003	0.454 ± 0.003	0.455 ± 0.003	0.454 ± 0.005
cardio	0.582 ± 0.004	0.578 ± 0.004	0.574 ± 0.004	0.576 ± 0.005	0.56 ± 0.007
musk	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.999 ± 0.001
satellite	0.648 ± 0.002	0.647 ± 0.002	0.663 ± 0.002	0.656 ± 0.003	0.676 ± 0.003
satimages	0.925 ± 0.001	0.926 ± 0.001	0.936 ± 0.001	0.935 ± 0.002	0.902 ± 0.019
mnist	0.326 ± 0.004	0.326 ± 0.004	0.324 ± 0.004	0.338 ± 0.008	0.336 ± 0.007
shuttle	0.954 ± 0.001	0.952 ± 0.001	0.966 ± 0.0	0.951 ± 0.005	0.97 ± 0.002
smtp29	0.973 ± 0.001	0.972 ± 0.001	0.982 ± 0.0	0.971 ± 0.004	0.977 ± 0.003
http_distinct	0.585 ± 0.008	0.641 ± 0.009	0.649 ± 0.01	0.626 ± 0.029	0.015 ± 0.002
mulcross	0.813 ± 0.007	0.819 ± 0.006	0.635 ± 0.007	0.637 ± 0.013	0.712 ± 0.019
cover	0.057 ± 0.001	0.057 ± 0.001	0.054 ± 0.001	0.068 ± 0.003	0.053 ± 0.002
http_logged	0.961 ± 0.001	0.964 ± 0.001	0.962 ± 0.001	0.954 ± 0.008	0.798 ± 0.062
kdd99	0.756 ± 0.011	0.808 ± 0.009	0.594 ± 0.005	0.601 ± 0.02	0.559 ± 0.011
http29	0.764 ± 0.01	0.77 ± 0.015	0.591 ± 0.006	0.617 ± 0.033	0.518 ± 0.011
average	0.699	**0.706**	0.673	0.673	0.618
median	0.760	**0.788**	0.641	0.631	0.623

Table 5.3: Average precision scores of all approaches on all the datasets. autoAD-SSE and autoAD-VAR represent the autoAD algorithm using the two different quality measures. *Naive* is the ensemble composed by all 20 methods equally weighted. **Bold** values are the best between autoAD-VAR and *Naive*, the second best strategy

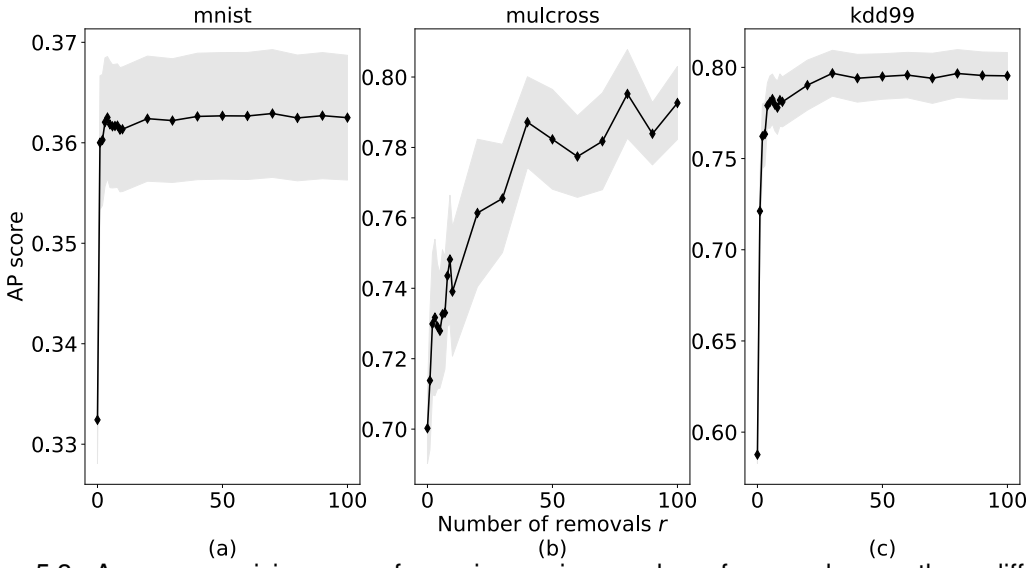


Figure 5.3: Average precision score for an increasing number of removals r on three different datasets, (a) mnist, (b) mulcross, and (c) kdd99 datasets, respectively.

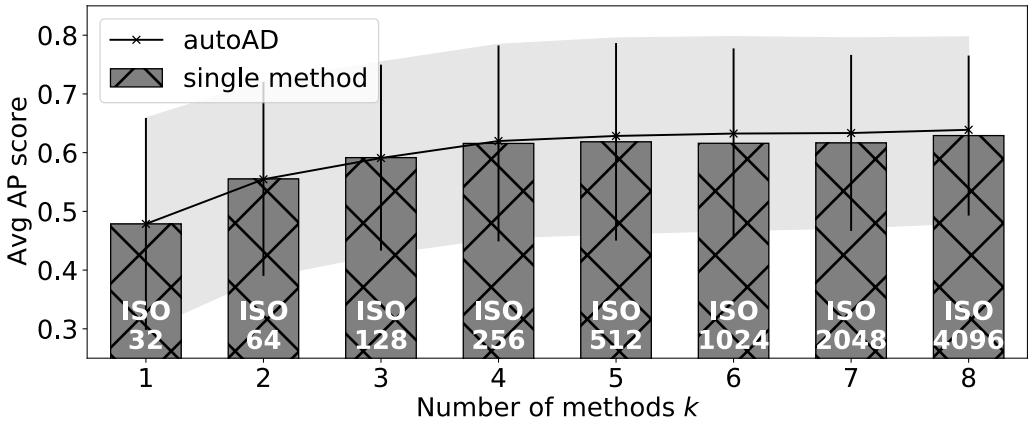


Figure 5.4: Average precision scores with a different number of methods.

once more similar AP scores of 0.635 ± 0.007, 0.712 ± 0.019 and 0.612 ± 0.007 respectively while the proposed methods raise them to 0.813 ± 0.007 and 0.819 ± 0.006 respectively.

On the contrary, when `autoAD`-SSE and `autoAD`-VAR perform worse than the competitors the losses are very limited. For instance, the *Naive* method is the best performing method on the smtp29 dataset (0.982 ± 0.001) while `autoAD`-SSE (0.973 ± 0.001) and `autoAD`-VAR (0.972 ± 0.001) are only 1% worse. Similarly, losses are contained on *satellite* (2%), *satimages* (1%), *shuttle* (1.2%) and *wbc* (1%).

We observe that overall the *Naive* method is on par (or slightly better) with performances generated by *SelectV* and *BoostSelect*. This is an expected behaviour as these two strategies

generate an initial *target* vector aggregating the results of all the methods considered. As a consequence, the correlation analysis selects methods that are average-performing ones. Our strategy instead independently weights each method (based on evidence of dispersion) and results are not influenced by other methods in the initial ensemble.

To understand such improvement, we study the impact of our main hyperparameters, the number of removals r and the number of methods k, on the precision performance. Fig. 5.3 shows how the AP score fluctuates when changing r on three datasets. We notice that when r increases (where we remove the instances with the highest anomaly scores), the AP score increases accordingly. That is an expected behavior similar to ensemble-based methods where several weak learners are more accurate than a single one. We set $r = 100$ as it is commonly used in ensemble models.

In Fig. 5.4, we present the `autoAD` results in terms of precision while increasing the number of methods used in our framework. We remark that the overall precision tends to be as good as the best method in the ensemble even when only few methods are used. For instance, `autoAD` performs as good as *ISO@256*[4] (best method) when the first four methods are used. By further growing the number of methods, the space of choices broadens and the model starts showing results better than any single method.

Quality measures. The overall unsupervised quality metric ϕ plays a key role in the weighting phase. A good correlation between the latter and the average precision is therefore desirable for the `autoAD` to produce satisfying results.

We report in Fig. 5.5 two examples on kdd99 and mulcross datasets showing the assigned weight *w*, the AP score for each method used and its quality measure ϕ. The methods are ordered according to the quality measure in decreasing order. Taking a deeper look at Fig. 5.5 (a) and (b) with the kdd99 dataset, one can notice that the worst performing method, *i.e.*, *ISO@32* with an AP=0.17, is also the one that produces the worst result in terms of the quality measure ϕ, where the weight $w = 0$. By looking at the quality measure plot from left to right, we observe that decreasing values (and thus higher weight w) correlate with increasing AP scores. Similar result

[4] Algorithm@parameter. Example: $ISO@256$ is the iForest algorithm with $\psi = 256$.

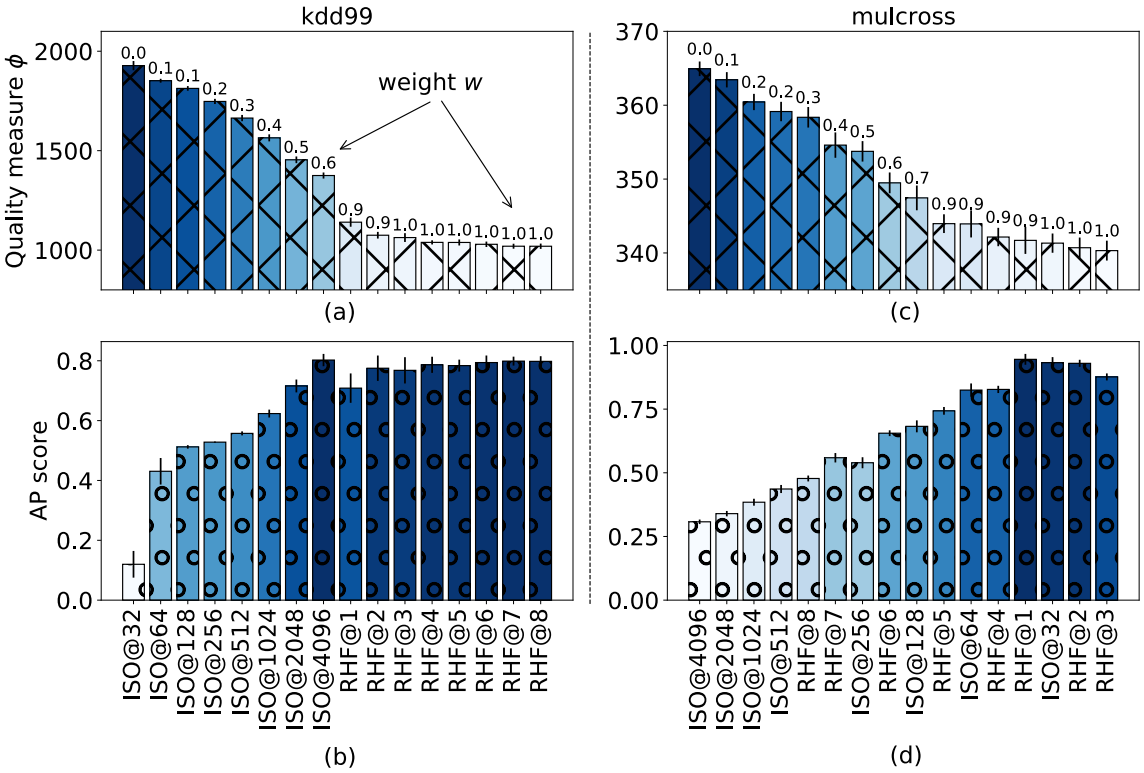


Figure 5.5: Correlation between the variance-quality measure ϕ with (a) kdd99, (c) mulcross, and the average precision score with (b) kdd99, (d) mulcross of each method. The methods are presented in a decreasing order according to ϕ.

is obtained with the mulcross dataset represented in Fig. 5.5 (c) and (d), as well as with most of the datasets. Similarly, in this case larger weights are assigned to methods having high AP scores.

Interestingly, we remark a contradictory pattern when comparing the weights assigned to each method. For example, with the kdd99 dataset, we obtain better precision when we increase the sampling hyperparameter ψ. The worst result is thus achieved by *ISO@32* while *ISO@4096* gives the best performance. Opposite results were obtained with the mulcross dataset, where *ISO@4096* is the worst and *ISO@32* is one of the best performing methods.

Scalability Analysis. The running time complexity of `autoAD` depends on the number of input methods $k = |\mathbb{M}|$ and the number of removals r drawn at random from $\mathcal{U}(0, 10\%)$.

In Fig. 5.6 we report the running time behavior when we change these two hyperparameters. Fig. 5.6 (a) depicts the running time consumed by `autoAD` and each method in the framework. We can see that the overall running time of `autoAD` increases linearly with time performance of

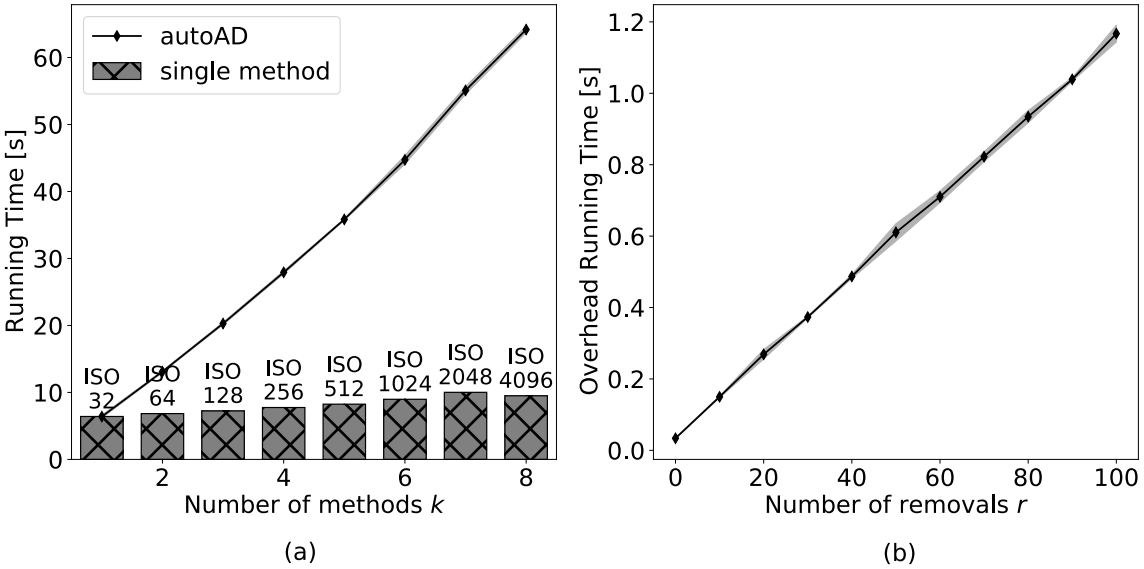


Figure 5.6: Running time. (a) the `autoAD` running time together with the running time of each method (bar plot). (b) the running time with different number of removals r.

each method in the ensemble. On the other hand, Fig. 5.6 (b) shows that when we increase the number of removals r, the execution time of `autoAD` linearly increases accordingly.

Overall, the `autoAD` framework shows a high correlation between the obtained AP scores and the two quality measures. The results obtained using different unsupervised quality metrics show that it is sufficient to focus on the top most anomalous scores of each method to study its reliability and effectiveness. Such strategy does not aggregate anomaly scores from different sources into a unique *target* vector as *SelectV* or *BoostSelect* but threat them independently.

5.5 Concluding Remarks and Future Directions

In this chapter, we presented `autoAD`, the first, to the best of our knowledge, autoML framework for unsupervised anomaly detection based on anomalies removal which produces simple and intuitive quality measures for each method. Experiments conducted on a various set of datasets show substantial gains in terms of average precision score, while showing linear running time in the number of methods and input data size. We release our Python3 parallel implementation, which runs all different methods in parallel, thereby achieving some significant speedup.

Several improvements can be addressed in the future. The framework previously presented indeed has a hyperparameter itself, that is the upper bound $\mathcal{U}(0, 10\%)$ when selecting the top instances to remove. Such hyperparameter is selected according to the common definition of anomaly detection datasets in which it is assumed anomalies to be rare (e.g. $5/10\%$ of the dataset). The selected value however shows to be effective also when there are very low percentages of true anomalies in the input datasets. The datasets in Table 5.1 are composed largely by small fractions of outliers, e.g. *kdd99* contains only 0.17% anomalies while the performance of our proposed method is AP=0.80 compared to SelectV AP=0.60, the second best performing method. Similarly, *http29* contains only 0.65% anomalies while the performances of our method is AP=0.77 compared to AP=0.61 of the second best performing method. Besides, more sophisticated unsupervised metrics can be explored and further studies on the weighting process can be performed. The framework indeed is an ensemble model in which all the algorithms are weighted accordingly; one could instead further study the performance of the framework using only the first best performing method. Notice however that in such scenario, the execution time would not be reduced since all the algorithms require to be run. Under these circumstances the framework remains to be compared to meta-learners which select an algorithm and perform a single run resulting to be faster. Nevertheless, such comparison has to guarantee fairness. It would be unfair to compare the methods on datasets used already in the tuning phase of the meta-learners. Furthermore, fairness has to be guaranteed also in what concerns the pool of algorithms and configurations used.

Chapter 6

Conclusion

Anomaly Detection has become an increasingly important task in Machine Learning. It focuses on identifying faults and detecting malicious activities in diverse application domains, ranging from data security and fraud detection to healthcare. In the last years a great effort has been made in developing novel anomaly detection methods and algorithms whose objective vary on requirements of different applications. For example, algorithms can be designed to target different types of anomalies (e.g. local vs global anomalies) or to process new instances as soon as possible in monitoring applications.

We examined, in this thesis, several key aspects of anomaly detection. We presented a novel *batch algorithm* for *outlier detection* as well as a *time series stream algorithm* for *novelty detection* particularly useful in computer networks monitoring applications. Finally, we investigated *automated algorithms* presenting a novel strategy based on independent tests instead of correlation analysis.

This concluding chapter summarizes the contributions of this thesis.

6.1 Summary of Contributions

After the discussion and clarification of *anomaly detection*, taxonomy of the anomaly types, strategies and algorithm classes given in the first two chapters of this thesis, we first presented in Chapter 3 Random Histogram Forest (RHF), an unsupervised batch algorithm for *outlier detection*. RHF is an ensemble model which builds a random forest while using the *Kurtosis* score as splitting criterion. The anomaly score of each instance is computed as the information content of the *leaf* it belongs to. We provide an extensive experimental evaluation on 38 public datasets and 64 private ones. Our experimental evaluation shows that our approach outperforms the other approaches in terms of average precision, while being simple and intuitive. Moreover the performance of our algorithm are consistently good over a wide range of values for their hyperparameters, while it requires only two hyperparameters. Finally, our proposed *Kurtosis Split* shows to be effective in high dimensional datasets while maintaining linear running time in the size of the input dataset.

In Chapter 4 we proposed ODS, a *time series* unsupervised engine that leverages DenStream, an online clustering method. We apply it to measurements collected from real network equipment, gathered on BGP-only datacenter network loaded with up to 3 Tbps aggregated traffic, and extensively compare it with a set of stream techniques such as *ExactSTORM, COD* and *RRCF*. Our results show that despite ODS is apparently plagued with several hyperparameters inherited from DenStream, their selection is quite straightforward, and performance are robust to inner hyperparameter selection. Additionally, ODS is significantly faster than any of the tested algorithms, and second only to RRCF (yet very close to it) in terms of detection performance. Overall, the above results suggest ODS as a particularly lightweight and suitable algorithm for stream-mode real-time network anomaly detection in monitoring applications.

Finally, in Chapter 5, we presented `autoAD`, the first, to the best of our knowledge, autoML framework for unsupervised anomaly detection based on anomalies removal which produces simple and intuitive quality measures for each method. Experiments conducted on a various set of datasets show substantial gains in terms of average precision score, while showing linear running time in the number of methods and input data size. `autoAD` shows significant gains when compared with existing automated strategies as *Naive*, *SelectV* or *BoostSelect* which are based con correlation analysis.

Bibliography

[1] https://cloud.google.com/explainable-ai.

[2] https://github.com/anrputina/ods-anomalydetection.

[3] https://github.com/anrputina/ODS-2020.

[4] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[5] The yang 1.1 data modeling language. *RFC 7950* , Aug. 2016.

[6] 2018. URL https://github.com/cisco-ie/telemetry.

[7] 2018. URL https://blogs.cisco.com/sp/introducing-pipeline-a-model\-driven-telemetry-collection-service.

[8] https://www.arista.com/en/solutions/telemetry-analytics, 2018.

[9] https://www.cisco.com/c/en/us/solutions/service-provider/cloud-scale-networking-solutions/model-driven-telemetry.html, 2018.

[10] http://support.huawei.com/enterprise/en/doc/EDOC1000173015?section=j006, 2018.

[11] https://github.com/YangModels/yang, 2018.

[12] 2021. URL https://github.com/vatsalsharan/pidforest.

[13] 2021. URL https://cmuxstream.github.io/.

[14] C. C. Aggarwal. *Data Classification: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2014. ISBN 1466586745.

[15] C. C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016. ISBN 3319475770.

[16] S. Agrawal and J. Agrawal. Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60:708–713, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.08.220. URL https://www.sciencedirect.com/science/article/pii/S1877050915023479. Knowledge-Based and Intelligent Information and Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings.

[17] L. Akoglu. Anomaly mining - past, present and future. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4932–4936. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/697. URL https://doi.org/10.24963/ijcai.2021/697. Early Career.

[18] N. M. Al-Rousan and L. Trajković. Machine learning models for classification of BGP anomalies. In *IEEE HPRS*, June 2012. doi: 10.1109/HPSR.2012.6260835.

[19] F. Angiulli and F. Fassetti. Detecting distance-based outliers in streams of data. In *ACM CIKM*, pages 811–820, 2007. doi: 10.1145/1321440.1321552.

[20] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *Principles of Data Mining and Knowledge Discovery*, 2002.

[21] D. Bajusz, A. Rácz, and K. Héberger. Comparison of data fusion methods as consensus scores for ensemble docking. *Molecules*, 24:2690, 07 2019. doi: 10.3390/molecules24152690.

[22] M. Bartos, A. Mullapudi, and S. Troutman. RRCF: Implementation of the Robust Random Cut Forest algorithm for anomaly detection on streams. *The Journal of Open Source Software*, 4(35), 2019.

[23] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[24] M. Bjorklund. YANG - A data modeling language for NETCONF. *RFC 6020*, Oct. 2010.

[25] G. E. P. Box and D. A. Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970. doi: 10.1080/01621459.1970.10481180.

[26] P. B. Brazdil, C. Soares, and J. P. da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003. ISSN 1573-0565.

[27] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00. doi: 10.1145/342009.335388.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2), May 2000. ISSN 0163-5808. doi: 10.1145/335191.335388.

[29] J. P. Burg. A new analysis technique for time series data. *NATO Advanced Study Institute on Signal Processing, 1968.*, 1968.

[30] E. Burnaev, P. Erofeev, and D. Smolyakov. Model selection for anomaly detection. In *ICMV*, 2015.

[31] G. O. Campos, A. Zimek, and W. Meira. An unsupervised boosting strategy for outlier detection ensembles. In D. Phung, V. S. Tseng, G. I. Webb, B. Ho, M. Ganji, and L. Rashidi, editors, *Advances in Knowledge Discovery and Data Mining*, pages 564–576, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93034-3.

[32] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM Conference on Data Mining*, 2006.

[33] P. Casas Hernandez, J. Mazel, and P. Owezarski. Unsupervised Network Intruison Detection Systems: Detecting the Unknown without Knowledge. *Computer Communications*, 35 (7):772–783, 2012.

[34] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15, 2009.

[35] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: Special issue on learning from imbalanced data sets. *SIGKDD Explor. Newsl.*, 6(1):1–6, June 2004. ISSN 1931-0145. doi: 10.1145/1007730.1007733. URL https://doi.org/10.1145/1007730.1007733.

[36] Y.-W. Chen and C.-J. Lin. *Combining SVMs with Various Feature Selection Strategies*, pages 315–324. Springer Berlin Heidelberg, 2006.

[37] Z. Chen, Z. Peng, X. Zou, and H. Sun. Deep learning based anomaly detection for muti-dimensional time series: A survey. In W. Lu, Y. Zhang, W. Wen, H. Yan, and C. Li, editors, *Cyber Security*, pages 71–92, Singapore, 2022. Springer Singapore. ISBN 978-981-16-9229-1.

[38] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995. ISSN 1573-0565. doi: 10.1007/BF00994018.

[39] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, 2006. doi: 10.1145/1143844.1143874.

[40] C. De Stefano, C. Sansone, and M. Vento. To reject or not to reject: that is the question-an answer in case of neural classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(1):84–94, 2000. doi: 10.1109/5326.827457.

[41] I. O. de Urbina Cazenave, E. Köşlük, and M. C. Ganiz. An anomaly detection framework for BGP. In *International Symposium on Innovations in Intelligent Systems and Applications*, June 2011. doi: 10.1109/INISTA.2011.5946083.

[42] S. Deshpande, M. Thottan, T. K. Ho, and B. Sikdar. An online mechanism for bgp instability detection and analysis. *IEEE Transactions on Computers*, 58(11):1470–1484, Nov 2009. doi: 10.1109/TC.2009.91.

[43] M. M. Deza and E. Deza. *Encyclopedia of Distances*, pages 1–583. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00234-2. doi: 10.1007/978-3-642-00234-2_1. URL https://doi.org/10.1007/978-3-642-00234-2_1.

[44] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406 – 421, 2018. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.09.037.

[45] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, 2018.

[46] J. Dromard, G. Roudière, and P. Owezarski. Online and scalable unsupervised network anomaly detection method. *IEEE Transactions on Network and Service Management*, 14 (1):34–47, 2017. ISSN 1932-4537. doi: 10.1109/TNSM.2016.2627340.

[47] D. Dua and C. Graff. UCI machine learning repository, 2017.

[48] R. Elshawi, M. Maher, and S. Sakr. Automated machine learning: State-of-the-art and open challenges. 2019.

[49] A. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong. A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158*, 2015.

[50] A. Emmott, S. Das, et al. A meta-analysis of the anomaly detection problem. *arXiv: Artificial Intelligence*, 2015.

[51] W. Enders. Applied econometric time series. *Technometrics*, 46:264 – 264, 1994.

[52] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *AAAI KDD*, 1996.

[53] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[54] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. 2019.

[55] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. 2019.

[56] B. Frenay and M. Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014. doi: 10.1109/TNNLS.2013.2292894.

[57] R. Fujimaki, T. Yairi, and K. Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 401–410, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 159593135X. doi: 10.1145/1081870.1081917. URL https://doi.org/10.1145/1081870.1081917.

[58] J. Gan and Y. Tao. Dbscan revisited: Mis-claim, un-fixability, and approximation. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015.

[59] M. C. Ganiz, S. Kanitkar, M. C. Chuah, and W. M. Pottenger. Detection of interdomain routing anomalies based on higher-order path analysis. In *ICDM*, 2006. doi: 10.1109/ICDM.2006.52.

[60] E. S. Gardner. Exponential smoothing: The state of the art—part ii. *International Journal of Forecasting*, 22(4):637–666, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.03.005. URL https://www.sciencedirect.com/science/article/pii/S0169207006000392.

[61] M. Goldstein and A. Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. 2012.

[62] M. Goldstein and S. Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 2016.

[63] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[64] P. Gopalan, V. Sharan, and U. Wieder. Pidforest: Anomaly detection via partial identification. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alch'e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/eb6dc8aba23375061b6f07b137617096-Paper.pdf.

[65] T. Green, A. Lambert, C. Pelsser, and D. Rossi. Leveraging Inter-domain Stability for BGP Dynamics Analysis. In *International Conference on Passive and Active Network Measurement (PAM)*, 2018.

[66] F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969. doi: 10.1080/00401706.1969.10490657. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1969.10490657.

[67] S. Guha, N. Mishra, G. Roy, and O. Schrijvers. Robust random cut forest based anomaly detection on streams. In *ICML*, page 2712–2721, 2016.

[68] A. Gunawan. A faster algorithm for dbscan. In *Master's thesis, Technische University Eindhoven*, March 2013.

[69] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, Sept 2014. ISSN 1041-4347.

[70] Hanchuan Peng, Fuhui Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, Aug 2005. doi: 10.1109/TPAMI.2005.159.

[71] D. Hawkins. *Identification of Outliers*. Chapman and Hall, 1980.

[72] D. Hawkins. *Identification of Outliers*. Chapman and Hall, 1980.

[73] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 2021.

[74] Y. Huang, N. Feamster, A. Lakhina, and J. J. Xu. Diagnosing network disruptions with network-wide analysis. *SIGMETRICS Perform. Eval. Rev.*, 35(1):61–72, June 2007. ISSN 0163-5999. doi: 10.1145/1269899.1254890.

[75] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: an automatic algorithm configuration framework. *JAIR*, 2009.

[76] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, 2011.

[77] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning*. Springer, 2019.

[78] R. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice, 3rd edition, OTexts*. 2021.

[79] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. Performance anomaly detection and bottleneck identification. *ACM Comput. Surv.*, 48(1), July 2015. ISSN 0360-0300. doi: 10.1145/2791120. URL https://doi.org/10.1145/2791120.

[80] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, pages 429–449, 2002.

[81] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *ACM SIGKDD*, 2019.

[82] M. V. Joshi, R. C. Agarwal, and V. Kumar. Mining needle in a haystack: Classifying rare classes via two-phase rule induction. *SIGMOD Rec.*, 30(2):91–102, May 2001. ISSN 0163-5808. doi: 10.1145/376284.375673. URL https://doi.org/10.1145/376284.375673.

[83] B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, 2014.

[84] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. Continuous monitoring of distance-based outliers over data streams. In *IEEE ICDE*, pages 135–146, 2011.

[85] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *JMLR*, 2017.

[86] K.-H. Lai, D. Zha, J. Xu, and Y. Zhao. Revisiting time series outlier detection: Definitions and benchmarks. 2021.

[87] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. *SIGCOMM Comput. Commun. Rev.*, 34, Aug. 2004. ISSN 0146-4833. doi: 10.1145/1030194.1015492.

[88] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *SIAM International Conference on Data Mining*, volume 3, 05 2003. doi: 10.1137/1.9781611972733.3.

[89] J. Li, D. Dou, Z. Wu, S. Kim, and V. Agarwal. An internet routing forensics framework for discovering rules of abnormal BGP events. *ACM SIGCOMM Computer Communication Review*, 35:55–66, 10 2005. doi: 10.1145/1096536.1096542.

[90] J. Li, W. Pedrycz, and I. Jamal. Multivariate time series anomaly detection: A framework of hidden markov models. *Applied Soft Computing*, 60:229–240, 2017. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2017.06.035. URL https://www.sciencedirect.com/science/article/pii/S1568494617303782.

[91] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[92] F. T. Liu, K. M. Ting, and Z. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

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

[94] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 2016.

[95] J. MacQueen. Some methods for classification and analysis of multivariate observations. University of California Press, 1967.

[96] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[97] E. Manzoor, H. Lamba, and L. Akoglu. Xstream: Outlier detection in feature-evolving data streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 1963–1972, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220107. URL https://doi.org/10.1145/3219819.3220107.

[98] J. Mazel, P. Casas, R. Fontugne, K. Fukuda, and P. Owezarski. Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection. *International Journal of Network Management*, 25(5):283–305. doi: 10.1002/nem.1903.

[99] W. McKinney et al. Data structures for statistical computing in python. In *9th Python in Science Conference*, volume 445, pages 51–56, 2010.

[100] K. G. Mehrotra, C. K. Mohan, and H. Huang. *Anomaly Detection Principles and Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2017. ISBN 3319675249.

[101] Z. Miller, W. Deitrick, and W. Hu. Anomalous network packet detection using data stream mining. *J. Information Security*, 2(4), 2011.

[102] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang. Twitter spammer detection using data stream clustering. *Information Sciences*, 260:64 – 73, 2014. ISSN 0020-0255.

[103] J. J. A. Moors. The meaning of kurtosis: Darlington reexamined. *The American Statistician*, 40(4):283–284, 1986. doi: 10.1080/00031305.1986.10475415.

[104] T. Myron. *Thermostatics and thermodynamics : an introduction to energy, information and states of matter, with engineering applications / by Myron Tribus*. University series in basic engineering. D. Van Nostrand, Princeton [etc, cop. 1961.

[105] G. Münz, S. Li, and G. Carle. Traffic anomaly detection using kmeans clustering. In *In GI/ITG Workshop MMBnet*, 2007.

[106] J. M. Navarro and D. Rossi. Hurra! human readable router anomaly detection. In *2020 32nd International Teletraffic Congress (ITC 32)*, pages 19–28, 2020. doi: 10.1109/ITC3249928.2020.00011.

[107] J. M. Navarro and D. Rossi. Hurra: Human-readable router anomaly detection. In *International Teletraffic Congress (ITC32)*, 2020.

[108] C. Ning, C. An, and L.-X. Zhou. An incremental grid density-based clustering algorithm. 1313:1–7, 01 2002.

[109] T. E. Oliphant. *A guide to NumPy*. Trelgol Publishing USA, 2006.

[110] P. Lapukhov, A. Premji, J. Mitchell. Use of BGP for Routing in Large-Scale Data Centers. In *RFC7938*, Aug. 2016.

[111] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), mar 2021. ISSN 0360-0300. doi: 10.1145/3439950. URL https://doi.org/10.1145/3439950.

[112] P. Pébay, T. B. Terriberry, H. Kolla, and J. Bennett. Numerically stable, scalable formulas for parallel and online computation of higher-order multivariate central moments with arbitrary weights. *Computational Statistics*, 31(4):1305–1325, Dec 2016. ISSN 1613-9658. doi: 10.1007/s00180-015-0637-z.

[113] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*.

[114] T. Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102, 07 2015. doi: 10.1007/s10994-015-5521-0.

[115] G. Poojitha, K. N. Kumar, and P. J. Reddy. Intrusion detection using artificial neural network. In *IEEE Conference on Computing, Communication and Networking Technologies*, July 2010. doi: 10.1109/ICCCNT.2010.5592568.

[116] A. Putina, 2021. URL https://github.com/anrputina/rhf.

[117] A. Putina, D. Rossi, A. Bifet, S. Barth, D. Pletcher, C. Precup, and P. Nivaggioli. Telemetry-based stream-learning of bgp anomalies. In *ACM SIGCOMM, Big-DAMA workshop*, 2018.

[118] A. Putina, M. Sozio, D. Rossi, and J. M. Navarro. Random histogram forest for unsupervised anomaly detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1226–1231, 2020. doi: 10.1109/ICDM50108.2020.00154.

[119] A. Putina, M. Sozio, D. Rossi, and J. M. Navarro. Random histogram forest for unsupervised anomaly detection. In *International Conference on Data Mining (ICDM)*, pages 1226–1231. IEEE, 2020.

[120] G. Quellec, M. Lamard, M. Cozic, G. Coatrieux, and G. Cazuguel. Multiple-instance learning for anomaly detection in digital mammography. *IEEE Transactions on Medical Imaging*, 35 (7):1604–1614, 2016. doi: 10.1109/TMI.2016.2521442.

[121] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[122] S. Rajasegarar, C. Leckie, and M. Palaniswami. Anomaly detection in wireless sensor networks. *IEEE Wireless Communications*, 15(4):34–40, 2008. doi: 10.1109/MWC.2008.4599219.

[123] S. Rajasegarar, C. Leckie, and M. Palaniswami. Hyperspherical cluster based distributed anomaly detection in wireless sensor networks. *Journal of Parallel and Distributed Computing*, 74(1), 2014. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2013.09.005.

[124] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 2000. ISSN 0163-5808. doi: 10.1145/335191.335437.

[125] G. Ratsch, S. Mika, B. Scholkopf, and K.-R. Muller. Constructing boosting algorithms from svms: an application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1184–1199, 2002. doi: 10.1109/TPAMI.2002.1033211.

[126] S. Rayana. ODDS library [http://odds.cs.stonybrook.edu], 2016.

[127] S. Rayana and L. Akoglu. Less is more: Building selective anomaly ensembles, 2015.

[128] F. Salutari, D. Da Hora, G. Dubuc, and D. Rossi. Analyzing wikipedia users' perceived quality of experience: A large-scale study. *IEEE Transactions on Network and Service Management*, 2020.

[129] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, 2000. doi: 10.1147/rd.441.0206.

[130] S. Sathe and C. C. Aggarwal. Subspace outlier detection in linear time with randomized hashing. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 459–468, 2016. doi: 10.1109/ICDM.2016.0057.

[131] B. Schölkopf, R. Williamson, et al. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, 1999.

[132] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

[133] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[134] D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, and S. T. Edwards. Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data, 2017.

[135] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang. A novel anomaly detection scheme based on principal component classifier. In *IEEE Foundations and New Directions of Data Mining*, 2003.

[136] N. Smirnov. Table for Estimating the Goodness of Fit of Empirical Distributions. *The Annals of Mathematical Statistics*, 19(2):279 – 281, 1948. doi: 10.1214/aoms/1177730256. URL https://doi.org/10.1214/aoms/1177730256.

[137] I. Steinwart, D. Hush, and C. Scovel. A classification framework for anomaly detection. *Journal of Machine Learning Research*, 6(8):211–232, 2005. URL http://jmlr.org/papers/v6/steinwart05a.html.

[138] H. A. Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66, 1926. doi: 10.1080/01621459.1926.10502161.

[139] B. Subba, S. Biswas, and S. Karmakar. A neural network based system for intrusion detection and attack classification. In *22nd National Conference on Communication (NCC)*, pages 1–6, March 2016. doi: 10.1109/NCC.2016.7561088.

[140] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *IEEE CISDA*, CISDA'09, pages 53–58. IEEE Press, 2009. ISBN 978-1-4244-3763-4.

[141] D. Tax. One-class classification; concept-learning in the absence of counter-examples. 01 2001.

[142] J. P. Theiler and D. M. Cai. Resampling approach for anomaly detection in multispectral images. In S. S. Shen and P. E. Lewis, editors, *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery IX*, volume 5093 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 230–240, Sept. 2003. doi: 10.1117/12.487069.

[143] M. Thottan and C. Ji. Anomaly detection in ip networks. *IEEE Transactions on signal processing*, 51(8):2191–2204, 2003.

[144] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society*, 1999.

[145] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[146] J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[147] H. Wang and B. Raj. On the origin of deep learning, 2017.

[148] H. Wang, J. Gu, and S. Wang. An effective intrusion detection framework based on svm with feature augmentation. *Knowledge-Based Systems*, 09 2017. doi: 10.1016/j.knosys.2017.09.014.

[149] B. L. WELCH. THE GENERALIZATION OF 'STUDENT'S' PROBLEM WHEN SEVERAL DIFFERENT POPULATION VARLANCES ARE INVOLVED. *Biometrika*, 34(1-2):28–35, 01 1947. ISSN 0006-3444. doi: 10.1093/biomet/34.1-2.28. URL https://doi.org/10.1093/biomet/34.1-2.28.

[150] Q. Wu, J. Strassner, A. Farrel, and L. Zhang. Network telemetry and big data analysis. *IETF draft-wu-t2trg-network-telemetry-00*, Mar 2016.

[151] H. Xiong, G. Pandey, M. Steinbach, and V. Kumar. Enhancing data analysis with noise removal. *IEEE Transactions on Knowledge and Data Engineering*, 18(3):304–319, 2006. doi: 10.1109/TKDE.2006.46.

[152] K. Yamanishi and J.-i. Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 676–681, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 158113567X. doi: 10.1145/775047.775148. URL https://doi.org/10.1145/775047.775148.

[153] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000. ISBN 1581132336. doi: 10.1145/347090.347160.

[154] N. Ye. A markov chain model of temporal behavior for anomaly detection. 07 2000.

[155] D. S. Yeung, S. Jin, and X. Wang. Covariance-matrix modeling and detecting various flooding attacks. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(2), 2007. ISSN 1083-4427. doi: 10.1109/TSMCA.2006.889480.

[156] Y. Zhao, Z. Nasrullah, and Z. Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 2019.

[157] Y. Zhao, R. A. Rossi, and L. Akoglu. Automating outlier detection via meta-learning. *arXiv preprint arXiv:2009.10606*, 2020.

[158] C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17. ISBN 9781450348874. doi: 10.1145/3097983.3098052.

[159] M. Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science*, 2, 2004.

[160] R. Zuech, T. Khoshgoftaar, and R. Wald. Intrusion detection and big heterogeneous data: a survey. *Journal of Big Data*, 2:1–41, 2015.

ECOLE
DOCTORALE

Titre : Détection non supervisée d'anomalies : méthodes et applications

Mots clés : non supervisé, détection d'anomalies, apprentissage automatisé

Résumé : Une anomalie (également connue sous le nom de *outlier*) est une instance qui s'écarte de manière significative du reste des données et est définie par Hawkins comme "*une observation, qui s'écarte tellement des autres observations qu'elle éveille les soupçons qu'il a été généré par un mécanisme différent*."
La détection d'anomalies (également connue sous le nom de détection de valeurs aberrantes ou de nouveauté) est donc le domaine de l'apprentissage automatique et de l'exploration de données dans le but d'identifier les instances dont les caractéristiques semblent être incohérentes avec le reste de l'ensemble de données. Dans de nombreuses applications, distinguer correctement l'ensemble des points de données anormaux (*outliers*) de l'ensemble des points normaux (*inliers*) s'avère très important. Une première application est le nettoyage des données, c'est-à-dire l'identification des mesures bruyantes et fallacieuses dans un ensemble de données avant d'appliquer davantage les algorithmes d'apprentissage.
Cependant, avec la croissance explosive du volume de données pouvant être collectées à partir de diverses sources, par exemple les transactions par carte, les connexions Internet, les mesures de température, etc., l'utilisation de la détection d'anomalies devient une tâche autonome cruciale pour la surveillance continue des systèmes. Dans ce contexte, la détection d'anomalies peut être utilisée pour détecter des attaques d'intrusion en cours, des réseaux de capteurs défaillants ou des masses cancéreuses.
La thèse propose d'abord une approche basée sur un collection d'arbres pour la détection non supervisée d'anomalies, appelée *Random Histogram Forest (RHF)*. L'algorithme résout le problème de la dimensionnalité en utilisant le quatrième moment central (alias *kurtosis*) dans la construction du modèle en bénéficiant d'un temps d'exécution linéaire. Un moteur de détection d'anomalies basé sur le stream, appelé *ODS*, qui exploite DenStream, une technique de clustering non supervisée est présenté par la suite et enfin un moteur de détection automatisée d'anomalies qui allège l'effort humain requis lorsqu'il s'agit de plusieurs algorithmes et hyper-paramètres est présenté en dernier contribution.

Title : Unsupervised Anomaly Detection: Methods and applications

Keywords : unsupervised learning, anomaly detection, automated learning

Abstract : An anomaly (also known as *outlier*) is an instance that significantly deviates from the rest of the input data and being defined by Hawkins as "*an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism*."
Anomaly detection (also known as outlier or novelty detection) is thus the machine learning and data mining field with the purpose of identifying those instances whose features appear to be inconsistent with the remainder of the dataset. In many applications, correctly distinguishing the set of anomalous data points (*outliers*) from the set of normal ones (*inliers*) proves to be very important. A first application is data cleaning, i.e., identifying noisy and fallacious measurement in a dataset before further applying learning algorithms.
However, with the explosive growth of data volume collectable from various sources, e.g., card transactions, internet connections, temperature measurements, etc. the use of anomaly detection becomes a crucial stand-alone task for continuous monitoring of the systems. In this context, anomaly detection can be used to detect ongoing intrusion attacks, faulty sensor networks or cancerous masses.
The thesis proposes first a batch tree-based approach for unsupervised anomaly detection, called *Random Histogram Forest (RHF)*. The algorithm solves the curse of dimensionality problem using the fourth central moment (aka *kurtosis*) in the model construction while boasting linear running time. A stream based anomaly detection engine, called *ODS*, that leverages DenStream, an unsupervised clustering technique is presented subsequently and finally Automated Anomaly Detection engine which alleviates the human effort required when dealing with several algorithm and hyper-parameters is presented as last contribution.

Institut Polytechnique de Paris
91120 Palaiseau, France