Détection distribuée d'anomalies dans les flux de données

Maurras Ulbricht Togbe

Sorbonne Université

Ecole Doctorale Informatique, Télécommunications et Electronique (EDITE)

Laboratoire d'Informatique, Signal et Image, Électronique et Télécommunications (LISITE/ISEP)

Thèse présentée par **Maurras Ulbricht TOGBE**

Soutenue le **13 décembre 2022**

En vue de l'obtention du grade de docteur de la Sorbonne Université

Discipline **Informatique**
Version soumise aux examinateurs

Détection distribuée d'anomalies dans les flux de données

Composition du jury

Directrice de thèse	Mme Raja CHIKY	ISEP
Co-Encadrants de thèse	Mme Yousra CHABCHOUB	ISEP
	M. Aliou BOLY	Université Cheikh Anta Diop de Dakar
Rapporteurs	Mme Amel BOUZEGHOUB	Télécom SudParis
	M. Eric GRESSIER-SOUDAN	CNAM
Examinateurs	Mme Christine FRICKER	INRIA
	M. Mustapha LEBBAH	Université de Versailles Saint-Quentin-en-Yvelines

À Pierre et Marie-Odette TOGBE

À Théophile TOGBE
(Paix à ton âme)

Don't Worry Be Happy.

Meher Baba

Résumé

La détection d'anomalies est une problématique importante dans de nombreux domaines d'application comme la santé, les transports, l'industrie, etc. Par exemple, la cybercriminalité peut causer des pertes économiques considérables et menacer la survie des entreprises. Il s'agit d'un sujet d'actualité qui tente de répondre à la demande toujours croissante dans différents domaines tels que la détection d'intrusion, la détection de fraude, etc.

Dans cette thèse, nous avons présenté un état de l'art complet sur les algorithmes de détection d'anomalies. Nous avons proposé une classification de ces méthodes basée sur le type de jeux de données (batch, flux, séries temporelles, graphes, etc.), le domaine d'application et l'approche considérée (statistique, classification, clustering, etc.). Nous nous sommes concentrés ensuite sur trois algorithmes : Local Outlier Factory (LOF), One-Class Support Vector Machine (OC-SVM) et Isolation Forest (IForest), que nous avons testés sur différents jeux de données afin de comparer leurs performances.

Puis, nous nous sommes focalisés sur IForest, un algorithme non supervisé performant, basé sur une forêt d'arbres binaires pour isoler les anomalies. Nous avons présenté une étude approfondie et complète de IForest en évaluant l'impact de ses paramètres (nombre d'arbres, taille de l'échantillon et seuil de décision) sur l'efficacité de la détection et sur le temps d'exécution. Nous avons également étudié l'intérêt d'inclure des anomalies dans la phase d'entraînement.

Afin d'explorer les limites de IForest, nous avons réalisé différentes expérimentations sur des jeux de données réels largement utilisés dans la littérature et aussi sur des jeux de données synthétiques avec des distributions non-triviales. Nous avons conçu des jeux de données multidimensionnelles où les anomalies sont portées par plusieurs dimensions simultanément. De plus, nous avons fait varier la densité et la distance entre les anomalies et les données normales pour tester différentes situations. Nous avons comparé les performances de IForest et celles de sa version étendue appelée Extended IForest (EIF). Enfin, nous avons conçu et validé une nouvelle version de IForest, Majority Voting IForest (MVIForest), qui se base sur les différentes décisions des arbres individuels plutôt que sur une décision globale de la forêt. Les expériences montrent que MVIForest a un temps d'exécution plus court que IForest, avec presque la même performance pour la détection.

Dans cette thèse, nous avons également abordé la problématique de la détection d'anomalies dans les flux de données. Nous avons proposé une implémentation de Isolation Forest for Anomaly detection in Streaming Data (IForest ASD), une variante de IForest pour les flux de données. Cette implémentation est basée sur scikit-multiflow (River), un framework d'apprentissage automatique open source pour les flux de données. Nous avons comparé notre implémentation à un algorithme de détection d'anomalies de référence pour les flux de données appelé Half-Space Trees. Nous avons également étendu l'algorithme IForest ASD pour gérer la détection du concept drift en y incluant des méthodes de détection de drift bien connues comme ADaptive WINdowing (ADWIN) et Kolmogorov Smirnov WINdowing (KSWIN). Les expériences montrent que nos extensions consomment moins de ressources que IForest ASD avec des performances similaires voire supérieures pour certains jeux de données.

Enfin, la détection distribuée d'anomalies en utilisant IForest basé sur le framework Apache Spark a été étudiée et évaluée.

Toutes nos propositions ont été validées avec des expérimentations sur différents jeux de données réels et synthétiques à travers différentes métriques utilisées en machine learning et qui sont adaptées aux jeux de données non équilibrés.

Mots clés : détection d'anomalies, isolation forest, algorithme distribué, flux de données, concept drift

Abstract

Anomaly detection is an important issue in many application areas such as healthcare, transportation, industry, etc. For example, cybercrime can cause considerable economic losses and threaten the survival of companies. This is a hot topic that tries to meet the ever increasing demand in different areas such as intrusion detection, fraud detection, etc.

In this thesis, we have presented a complete state of the art on anomaly detection algorithms. We proposed a classification of these methods based on the type of datasets (batch, streams, time series, graphs, etc.), the application domain and the approach considered (statistical, classification, clustering, etc.). We then focused on three algorithms: Local Outlier Factory (LOF), One-Class Support Vector Machine (OC-SVM) and Isolation Forest (IForest), which we tested on different datasets in order to compare their performances.

Then, we focused on IForest, a powerful unsupervised algorithm based on a binary tree forest to isolate anomalies. We provided a deep and comprehensive study of IForest by evaluating the impact of its parameters (number of trees, sample size and decision threshold) on the detection efficiency and on the execution time. We also studied the interest of including anomalies in the training phase.

In order to explore the limitations of IForest, we performed different experiments on real datasets widely used in the literature and also on synthetic datasets with non-trivial distributions. We designed multidimensional datasets where anomalies are carried by several dimensions simultaneously. In addition, we varied the density and distance between the anomalies and the normal data to test different situations. We compared the performance of IForest and its extended version called Extended IForest (EIF). Finally, we designed and validated a new version of IForest, Majority Voting IForest (MVIForest), which is based on the different decisions of individual trees rather than a global forest decision. Experiments show that MVIForest has a shorter execution time than IForest, with almost the same performance for detection.

In this thesis, we also addressed the problem of anomaly detection in data streams. We proposed an implementation of Isolation Forest for Anomaly detection in Streaming Data (IForest ASD), a variant of IForest for data streams. This implementation is based on scikit-multiflow (River), an open source machine learning framework for data streams. We compared our implementation to a state-of-the-art anomaly detection algorithm for data streams called Half-Space Trees. We also extended the IForest ASD algorithm to handle concept drift detection by including well-known drift detection methods like ADaptive WINdowing (ADWIN) and Kolmogorov Smirnov WINdowing (KSWIN). Experiments show that our extensions consume less resources than IForest ASD with similar or even better performance for some datasets.

Finally, distributed anomaly detection using IForest based on the Apache Spark framework has been studied and evaluated.

All our proposals have been validated with experiments on different real and synthetic datasets through different metrics used in machine learning and which are adapted to unbalanced datasets.

Keywords: anomalies detection, isolation forest, distributed algorithm, data stream, concept drift

Remerciements

Plusieurs personnes ont contribué de près ou de loin à la réalisation de cette thèse. Je voudrais leur adresser toute ma gratitude.

Avant tout, je souhaite remercier Mme Raja CHIKY pour avoir été une directrice de thèse disponible et impliquée dans mes travaux. Elle est toujours présente dès que le besoin se fait sentir. Je voudrais remercier mon encadrante Mme Yousra CHABCHOUB pour la qualité de son encadrement et ses conseils pertinents. Présente, accessible et disponible, elle a toujours montré un grand intérêt à mes travaux et mes idées. Je remercie également mon autre encadrant M. Aliou BOLY. Il a toujours été présent, de bon conseil et cela remonte à très longtemps, avant même le début de la thèse (2015). Merci à vous de m'avoir encadré et d'avoir guidé mes travaux.

Je remercie les rapporteurs de cette thèse, Mme Amel BOUZEGHOUB et M. Eric GRESSIER-SOUDAN pour leurs remarques constructives et suggestions sur la version préliminaire de ce rapport. Je remercie également les membres du jury, Mme Christine FRICKER et M. Mustapha LEBBAH d'avoir accepté de participer au jury de cette thèse.

Je remercie spécialement mes parents, mes frères et sœurs pour leurs disponibilités, conseils et encouragements.

Je remercie tout particulièrement ma femme Christelle VINAWAMON pour son encouragement et sa grande patience.

Je remercie Amadou Fall DIA qui m'a conseillé, soutenu et accompagné à chaque étape de ces années. Je remercie également tous mes amis Husni et Nadège, Kowiwou, Paulin, Michel, Yaël, Néria, Etienne, Angèle, Anne-Michelle, Léa, Amina, Mouinath.

Je tiens à remercier tous les doctorants de l'ISEP avec qui j'ai passé de bons moments. Il m'ont aidé à traverser les moments de doute. Je remercie notamment Mariam pour sa collaboration fructueuse ainsi que Jade, Shufan, Arthur, Guillaume et Katya.

Je remercie également Romuald le sage conseiller et tous les enseignants, enseignants-chercheurs de l'ISEP, de l'UCAD, de Polytech à l'université de Tours et de l'université Paris Nanterre qui m'ont fait découvrir cet univers passionnant

d'enseignement et de recherche.

Merci à toutes et à tous pour tout !

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

Introduction

Sommaire

1.1 Contexte et motivation

Une anomalie est une observation ou groupe d'observations ayant un comportement très différent de la majorité des données (Barnett et Lewis, 1994; Chandola et al., 2009; Aggarwal, 2017). Les anomalies sont par définition très peu nombreuses dans le jeu de données, comparées aux données normales. Ceci constitue le principe de base de la détection d'anomalies. En effet, dans Ted et Ellen (2014), les auteurs considèrent que la détection d'anomalies est basée sur la modélisation de ce qui est normal dans le but de découvrir ce qui ne l'est pas. Les méthodes de détection d'anomalies englobent donc l'ensemble des procédés de modélisation du comportement normal présenté par la grande majorité des observations afin de découvrir les observations ou groupes d'observations qui ont un comportement différent. La détection d'anomalies est un exercice de classification binaire dont les deux classes possibles sont "normale" et "anormale".

La recherche et la classification des anomalies sont un sujet qui a intéressé les scientifiques depuis plusieurs dizaines d'années. À l'origine, cette problématique s'adressait aux statisticiens et l'un des domaines d'application majeurs concernés était la production industrielle, notamment pour la détection d'un défaut de fabrication ou d'une défaillance. Suite à l'émergence des nouvelles technologies, les domaines d'application de la détection d'anomalies se sont multipliés incluant une communauté scientifique bien plus large. Aujourd'hui, la détection d'anomalies est une problématique au croisement du Big data, de la Cybersécurité, et de l'intelligence artificielle.

En effet, avec l'évolution des technologies (réseaux hauts débit, 5G, 6G...) plusieurs nouvelles applications ont vu le jour, notamment les objets connectés et les réseaux sociaux qui produisent de gigantesques volumes de données en continu et avec un débit de plus en plus élevé. Exploiter ces données en temps réel afin d'en extraire les connaissances pour une meilleure aide à la prise de décision est l'objectif premier de plusieurs domaines de l'informatique de nos jours. Des domaines comme le Big data, le data mining, la business intelligence,

les systèmes de recommandation, le Machine learning et l'intelligence artificielle ont vu le jour et sont en constante évolution. Ils se basent tous sur les données générées par les différents outils technologique. La détection d'anomalies dans un grand volume de données est l'un des grands défis de ces domaines de recherche.

A titre d'exemple, dans le domaine de la santé, une anomalie peut prédire ou expliquer une crise cardiaque ou encore un mauvais fonctionnement des capteurs médicaux (Haque et al., 2022 ; Sarangi et Tripathy, 2022). Dans le domaine de la finance, du commerce, de l'assurance ou des impôts, la détection de fraude est une application majeure de la détection d'anomalies (Hilal et al., 2022). L'une des applications les plus répandues est la détection d'intrusion et de cyberattaques dans le domaine de la Cybersécurité (Umer et al., 2022). La détection d'anomalies est également présente dans l'industrie à travers la détection des défauts ou usures pour la maintenance prédictive (Divya et al., 2022) et aussi dans les réseaux de transport à travers la détection d'obstacles, d'accidents, de déviation de trajectoires, etc. (Hu et al., 2022).

La détection d'anomalies s'applique à tout type de données. On peut noter quelques exemples : la détection d'anomalies dans les images pour détecter les catastrophes naturelles (Sublime et Kalinicheva, 2019), ou encore pour diagnostiquer des maladies dans l'imagerie médicale (Shvetsova et al., 2021 ; Tschuchnig et Gadermayr, 2022). La détection d'anomalies s'applique également aux données textuelles à travers les logs (Vervaet, 2021 ; Ryciak et al., 2022 ; Shao et al., 2022) par exemple. Nos travaux dans cette thèse se concentrent particulièrement sur les données numériques multidimensionnelles. À noter que certaines méthodes évoquées dans ce rapport peuvent être appliquées ou facilement adaptées à d'autres types de données.

Les flux de données sont un type de données particulier dont les caractéristiques constituent des contraintes pour la détection d'anomalies. En effet, les flux de données sont illimités, générés à vitesse variable et de façon continue. C'est le cas par exemple des données collectées par les différents capteurs. Ces types de données sont incontournables de nos jours. Les flux de données étant évolutifs et infinis, il faut un traitement en temps réel avec des algorithmes ayant une faible complexité et une faible consommation mémoire. La détection

d'anomalies dans les flux de données doit donc se faire en ligne avec un passage unique sur chaque donnée. Elle doit également tenir compte du concept drift qui est un défi considérable. Plus d'informations ont été données aux chapitres 4 et 5 concernant la détection d'anomalies dans les flux de données.

Par ailleurs, il est à noter que le traitement des données se fait généralement de manière centralisée. En effet, les données sont collectées de différentes sources et sauvegardées sur un serveur central. L'algorithme s'exécute ainsi au niveau de ce serveur. Cette façon de fonctionner n'est pas toujours optimale. On peut donner l'exemple d'un réseau de capteurs où les données sont par défaut distribuées. Un traitement distribué serait donc beaucoup plus adéquat pour ces données.

En somme, avec le développement des domaines d'applications comme l'internet des objets (IoT), la bourse, les transports maritimes, terrestres et aériens, etc. les flux de données sont devenus incontournables. Les données générées ou collectées par ces applications sont par défaut distribuées. Il est donc important d'exploiter ces caractéristiques dans la détection d'anomalies pour une meilleure aide à la prise de décision. D'où notre sujet de thèse intitulé ***Détection distribuée d'anomalies dans les flux de données.***

Défis de la détection d'anomalies

La détection d'anomalies a pour double rôle de modéliser le comportement des données normales et d'aider à décider si les autres données sont anormales. Il n'est pas simple de mener à bien ces deux fonctions car la détection d'anomalies est confrontée à bien de défis. En plus du manque de données labellisées et des contraintes imposées par les flux de données à haut débit nous notons la mauvaise qualité des données, le caractère évolutif des données anormales, l'impact du domaine applicatif et l'effet du swamping et masking que nous expliquons ci-dessous.

La qualité des données

La donnée est une ressource indispensable pour toutes les méthodes de Machine Learning. La performance de ces méthodes est fortement dépendante de la qualité des données traitées. Dans les conditions réelles, les données collectées

ne sont pas toujours propres et prêtes à l'emploi. En effet, la défaillance des capteurs, les conditions de collecte et les erreurs humaines peuvent créer des incohérences dans le jeu de données. Ainsi, le jeu des données collectées peut contenir des valeurs manquantes, des valeurs aberrantes ou du bruit. Une étape de pré-traitement des données est donc indispensable pour améliorer la qualité des données avant d'appliquer des méthodes de détection d'anomalies. Pour pré-traiter les jeux de données, les valeurs manquantes peuvent être remplacées, les données erronées peuvent être supprimées ou corrigées (D'Urso, 2016). Aggarwal (2017) met en lumière sur la différence entre un bruit et une anomalie. Il existe différents traitements réservés aux bruits comme "noise removal" pour la suppression du bruit (Motwani et al., 2004 ; Frénay et Verleysen, 2013) et "noise accommodation" qui analyse et ajuste le bruit (Hampel, 2001 ; Rousseeuw et Hubert, 2018 ; Maronna et al., 2019).

L'évolutivité des anomalies

La détection d'anomalies implique une identification efficace des anomalies qui peuvent se traduire par des attaques dans certains domaines comme la Cybersécurité. Il n'est pas évident de définir et caractériser toutes les attaques possibles car les hackers ne manquent pas d'imagination et d'ingéniosité. Ils ne cessent d'innover et d'exploiter les évolutions technologiques. Ils sont à la recherche constante de moyens, de techniques pour dissimuler leurs attaques. Dans ce cadre nous pouvons citer Chandola et al. : « Lorsque les anomalies sont le résultat d'actions malveillantes, les adversaires malveillants s'adaptent pour que les observations anormales paraissent normales, ce qui complique la tâche de définition du comportement normal. » (Chandola et al., 2009). Le développement de différents outils dont l'utilisation peut être détournée pour la création de faux contenus comme le deepfake est un exemple. Deepfake detection (Yu et al., 2021) est une application du deep learning pour l'identification des vidéos créées grâce à l'intelligence artificielle. Les meilleures méthodes de deep learning sont à une précision de 65% seulement (Rauwerda, 2022), ce qui montre la difficulté de détecter de telles anomalies.

La dépendance du domaine applicatif

Bien que la notion d'anomalie soit théoriquement la même dans tous les domaines, techniquement, elle n'est pas pareille. Généralement, les algorithmes conçus pour un domaine donné ne sont pas toujours adaptés à un autre domaine. La nature des données considérées (images, séries temporelles, données spatiales..) a, en effet, un impact sur le choix de la méthode la plus adaptée. Pareil pour le volet données statiques versus flux de données. De plus le cadre applicatif implique un niveau de sensibilité et des objectifs de détection bien particuliers. Dans certaines applications critiques liées au domaine médical par exemple on souhaite détecter toutes les anomalies et on tolère un niveau élevé de fausses alarmes. Dans d'autres cadres moins exigeants on préfère rater quelques vraies anomalies pour baisser le taux de fausses alarmes.

Swamping et Masking

Le swamping et le masking représentent un vrai défi pour la détection d'anomalies. Le swamping (étouffement) consiste à identifier à tort certaines données normales comme des anomalies à cause d'une faible différence entre ces deux types de données. Concrètement, on dira qu'une donnée a_1 étouffe une deuxième donnée a_2 si a_2 ne peut être considérée comme anormale qu'en présence de a_1. En d'autres termes, après la suppression de a_1, a_2 devient une observation non aberrante donc normale. Le swamping arrive lorsqu'un groupe de données anormales déforme la moyenne et les estimations de la covariance proches de ses valeurs. La distance entre les données normales et la moyenne est donc grande, ce qui fait ressembler ces données à des anomalies. Le masking (masquage) est l'existence d'un grand nombre d'anomalies dissimulant leur propre présence. Lorsqu'un groupe d'anomalies est grand et dense, certaines méthodes pourraient les considérer comme "un autre" comportement normal. Ces données anormales risquent ainsi d'être considérées comme normales. Concrètement, on dira qu'une donnée d_1 masque une seconde donnée d_2 si d_2 ne peut être considérée comme une anomalie que par elle-même et non en présence de d_1. Ainsi, après la suppression de d_1, d_2 apparaît comme une anomalie. Le masking se produit lorsqu'un groupe de données anormales biaise la moyenne et les

estimations de covariance proches de ses valeurs. La distance résultante de la donnée anormale par rapport à la moyenne est donc faible. L'échantillonnage aléatoire est souvent utilisé pour réduire l'effet du swamping et masking.

La détection d'anomalies étant un domaine de grand intérêt de nos jours, il est important de produire une liste des différentes techniques existantes afin de permettre à la communauté (data scientist, data analyst, chercheurs, enseignants, étudiants, etc.) du domaine de ne pas se perdre dans la multitude de ces techniques, d'être au parfum des nouvelles avancées mais également d'aller à l'essentiel selon leur besoin. Plusieurs revues existent dans la littérature mais en général elles ne se concentrent que sur quelques domaines d'application ou sur une catégorie d'algorithmes donnée ou ne sont pas à jour quand aux nouvelles avancées dans le domaine.

Dans cette thèse, nous proposons une large revue des méthodes de détection d'anomalies. Notre revue permet au lecteur une compréhension facile et un choix rapide en fonction de ses contraintes. Etant donné les différents types de données et de domaines existants, il n'est pas aisé de concevoir une méthode générique applicable à tout type de données et à tout type de domaine. Dans cette thèse, nous nous focalisons sur les flux de données. En effet, nous avons analysé une méthode performante et bien connue pour la détection d'anomalies et l'avons adaptée au contexte des flux. De plus, vu que certaines données sont par nature distribuées, nous avons étudié la distribution de certaines méthodes conçues pour un traitement centralisé. Tous les algorithmes développés dans cette thèse sont en Python ou en Scala et n'utilisent que des logiciels totalement ou partiellement open source. Tous nos codes sources sont disponibles sur GitHub pour faciliter la reproductibilité.

1.2 Contributions

L'objectif de cette thèse est d'étudier la détection d'anomalies dans les flux de données et de proposer des solutions pour une meilleure efficacité des méthodes de détection d'anomalies dans un tel contexte. Deux approches ont été explorées dans cette thèse : l'approche centralisée et l'approche distribuée. Premièrement,

nous avons étudié les méthodes centralisées de détection d'anomalies existantes dans la littérature. Ceci nous a permis de choisir une des plus performantes pour laquelle nous avons approfondi l'étude et proposer des améliorations. Deuxièmement, nous avons considéré l'approche distribuée et avons évalué les adaptations existantes pour une meilleure aide au choix.

Nos contributions peuvent être résumées en six points que nous décrivons ci-après.

1.2.1 Etude comparative des méthodes de détection d'anomalies

Après un état de l'art complet des méthodes de détection d'anomalies, nous avons comparé trois des méthodes les plus connues et efficaces spécifiquement conçues pour la détection d'anomalies. Soient les méthodes Local Outlier Factory (LOF), One-Class Support Vector Machine (OC-SVM) et Isolation Forest (IForest). Ces méthodes ont été choisies en fonction de leur efficacité, leur popularité, et leur différence en ce qui concerne le principe de fonctionnement. Nos expérimentations ont porté sur différents jeux de données réels et publics de différentes caractéristiques et qui sont très utilisés par la communauté de détection d'anomalies.

Cette première contribution est un point d'ancrage de nos travaux dans cette thèse. En effet, elle nous a permis de dégager une méthode efficace qui nous servira de base pour la suite des travaux notamment pour une détection distribuée d'anomalies dans les flux de données.

IForest (FT Liu et al., 2008) s'est avérée être une méthode efficace avec une faible complexité. Ceci constitue un avantage primordial pour la détection d'anomalies dans les flux de données par rapport aux autres méthodes considérées. Néanmoins, d'autres contraintes du streaming constituent des défis pour une meilleure détection d'anomalies dans les flux de données. Nous avons donc décidé d'approfondir l'étude de IForest pour une amélioration de son efficacité.

1.2.2 Etude approfondie de Isolation Forest

Nous avons dans cette deuxième contribution, analysé et discuté des limites de IForest. Les limites de IForest exposées dans la littérature ont été recensées et nous avons fait un état de l'art des évolutions de IForest qui ont été proposées dans la littérature tout en mettant en exergue les limites traitées par ces évolutions. L'objectif de cette deuxième contribution est d'explorer les faiblesses de IForest afin d'améliorer ses performances pour une meilleure détection d'anomalies dans les flux de données. À part les limites traitées dans la littérature, nous avons également découvert d'autres faiblesses qui constituent des axes d'amélioration de IForest.

Nous avons particulièrement mis l'accent sur la réduction du temps d'exécution de IForest. L'objectif est de répondre encore plus efficacement à la contrainte majeure de temps imposée par les flux de données. Ceci nous a conduit à proposer une amélioration de IForest appelée Majority Voting IForest (MVIForest).

1.2.3 Majority Voting IForest

Pour gagner en rapidité dans la détection d'anomalies avec IForest, nous avons dans cette troisième contribution changé le mode de prise de décisions de IForest. En effet, IForest est basée sur une prise de décision collective de tous les arbres de sa forêt. Ceci engendre une phase de test très coûteuse notamment pour les jeux de données larges. Majority Voting IForest se base sur le vote de la majorité. MVIForest calcule le score de la donnée x pour chaque arbre et s'arrête quand la majorité est atteinte soit $t/2+1$ pour l'une des deux classes (t étant le nombre d'arbres). Pour valider notre approche, nous avons réalisé des expérimentations sur des jeux de données publics réels et sur des jeux de données synthétiques. MVIForest est plus rapide que IForest et sa version étendue Extended IForest avec des performances de détection similaires, sinon meilleures.

1.2.4 IForest pour les flux de données

Dans cette quatrième contribution, nous nous sommes concentrés sur la détection d'anomalies dans les flux de données. Peu d'adaptations ont été recensées pour IForest au contexte du streaming. IForest for Anomaly detection in Streaming Data (IForest ASD) (Ding et Fei, 2013) en est une. À notre meilleure connaissance, aucune implémentation de IForest ASD n'existait dans la littérature. Nous avons proposé une implémentation de IForest ASD dans le framework de traitement de flux de données bien connu scikit-multiflow (River) (Montiel et al., 2018). Jusqu'alors, une seule méthode de détection d'anomalies était disponible dans ce framework. L'objectif de cette contribution est de rendre accessible à toute la communauté une autre méthode de détection d'anomalies avec de bonnes performances. Nos expérimentations ont prouvé l'efficacité de notre implémentation.

Toute méthode de détection d'anomalies dans les flux de données doit donner une attention particulière à la gestion du concept drift. La gestion du concept drift dans IForest ASD est largement améliorable. Nous avons proposé dans une autre contribution trois améliorations de IForest ASD.

1.2.5 Score-based ADWIN IFA, Prediction-based ADWIN IFA et NDKSWIN-based IFA

Le concept drift est un défi majeur dans le traitement des flux de données et plus particulièrement pour la détection d'anomalies dans les flux de données. Dans cette cinquième contribution, nous avons proposé et implémenté trois méthodes de détection d'anomalies dans les flux de données avec un accent particulier sur la détection du drift en utilisant ADWIN et KSWIN. ADaptive WINdowing (ADWIN) (Albert Bifet et Gavalda, 2007) et Kolmogorov-Smirnov WINdowing (KSWIN) (Raab et al., 2020) sont deux méthodes de détection du drift bien connues et implémentées dans le framework scikit-multiflow (River). KSWIN étant unidimensionnelle, nous l'avons d'abord adaptée au contexte du multidimensionnel. Notre adaptation est nommée N-Dimensional KSWIN (NDKSWIN).

IForest ASD détecte le drift en prenant en compte un paramètre d'entrée u représentant le pourcentage d'anomalies dans chaque fenêtre. IForest ASD étant une méthode non supervisée, il s'agit là d'une limite importante. Nos trois méthodes appelées SADWIN IFA (Score-based ADWIN IForest ASD), PADWIN IFA (Prediction-based ADWIN IForest ASD) et NDKSWIN IFA (N-Dimensionnal KSWIN-based IForest ASD) corrigent donc cette limite de IForest ASD. SADWIN IFA et PADWIN IFA détectent le drift du modèle construit par IForest soit en se basant sur les scores (le cas de SADWIN IFA), soit en se basant sur les prédictions (le cas de PADWIN IFA). NDKSWIN IFA quant à elle est en amont à la phase de test, c'est-à-dire que le drift est détecté directement sur les données de la fenêtre considérée.

Cette contribution nous a permis d'avoir une méthode centralisée de détection d'anomalies dans les flux de données, efficace et robuste par rapport au concept drift.

1.2.6 IForest distribuée

En prenant en compte la nature, par défaut distribuée, de certaines données comme celles collectées par les réseaux de capteurs, l'approche distribuée de la détection d'anomalies serait une très bonne piste pour une meilleure efficacité et surtout pour une rapidité dans le traitement des données. Dans cette contribution, nous avons considéré l'approche distribuée des méthodes de détection d'anomalies. Nous avons mené une étude comparative des deux versions distribuées de IForest existantes dans la littérature.

1.3 Organisation du manuscrit

Cette thèse est organisée comme suit :

Dans le **chapitre 2**, nous présentons un état de l'art des méthodes de détection d'anomalies. Après avoir listé les principales revues existantes, nous les avons classées en fonction des différents types de données (flux de données, séries temporelles, graphes, données spatiales, etc.), des techniques utilisées (statistiques, clustering, classification, plus proches voisins, etc.) et des domaines

d'application (détection d'intrusion, fraude, santé, etc.). Nous avons fait le choix de détailler certaines méthodes de détection d'anomalies phares comme Local Outlier Factory (LOF), One Class Support Vector Machine (OC-SVM) et Isolation Forest (IForest). Nous y avons exposé les principes et le fonctionnement de la méthode IForest avant de la comparer aux méthodes LOF et OC-SVM. Nous avons également présenté tous les jeux de données et les métriques utilisées dans le manuscrit.

Dans le **chapitre 3**, nous avons approfondi l'étude de IForest. En effet, nous avons évalué l'impact des différents paramètres de IForest sur ses performances. Pour explorer les limites de IForest, nous avons considéré des jeux de données synthétiques, multidimensionnelles contenant des anomalies non triviales. Plusieurs expérimentations ont été menées pour évaluer la méthode Majority Voting IForest (MVIForest) que nous avons proposée dans ce chapitre.

Dans le **chapitre 4**, nous avons considéré particulièrement la détection d'anomalies dans les flux de données. Nous avons commencé par un état de l'art des méthodes de détection d'anomalies dans les flux de données. Puis, nous avons présenté brièvement la notion de fenêtrage qui est essentielle pour le traitement des flux de données. Ensuite, nous avons expliqué notre implémentation de Isolation Forest for Anomalies detection in Streaming Data (IForest ASD), une variante de IForest pour les flux de données, dans le framework scikit-multiflow. Enfin, nous avons effectué plusieurs expérimentations et comparaisons pour valider cette implémentation.

Dans le **chapitre 5**, nous avons approfondi nos travaux du chapitre 4 en nous concentrant sur la gestion du concept drift. Nous avons dans un premier temps discuté du concept drift avant de présenter nos propositions d'améliorations de IForest ASD pour une meilleure gestion du concept drift. Nos propositions utilisent les méthodes ADaptive WINdowing (ADWIN) et Kolmogorov-Smirnov WINdowing (KSWIN) bien connues pour la détection du concept drift. Des expérimentations ont été menées pour valider nos propositions.

Dans le **chapitre 6**, nous avons présenté un état de l'art des méthodes distribuées de détection d'anomalies et avons effectué une étude comparative des deux implémentations distribuées disponibles dans la littérature pour IForest.

Le **chapitre 7** conclut le rapport. Nous y avons repris l'essentiel des résultats

obtenus et avons exposé quelques pistes de travaux futurs dans la continuité des travaux effectués dans cette thèse.

1.4 Liste des publications

Toutes nos propositions ont été validées avec des expérimentations sur différents jeux de données réels et synthétiques à travers différentes métriques utilisées en machine learning et qui sont adaptées aux jeux de données non équilibrés. Nos résultats ont été publiés dans des conférences et journaux internationaux.

Journaux internationaux

— Yousra Chabchoub, **Maurras Ulbricht TOGBE**, Aliou Boly, Raja Chiky. An in-depth study and improvement of Isolation Forest. IEEE Access, 2022, vol. 10, p. 10219-10237. https://doi.org/10.1109/ACCESS.2022.3144425

— **Maurras Ulbricht TOGBE**, Yousra Chabchoub, Aliou Boly, Mariam Barry, Raja Chiky, Maroua Bahri. Anomalies Detection Using Isolation in Concept-Drifting Data Streams. Computers 2021, 10, 13. https://doi.org/10.3390/computers10010013

Conférences internationales

— **Maurras Ulbricht TOGBE**, Yousra CHABCHOUB, Aliou BOLY et Raja CHIKY. Distributed anomalies detection using Isolation Forest and Spark. 14th International Conference on Computational Collective Intelligence, ICCCI 2022.

— **Maurras Ulbricht TOGBE**, Mariam BARRY, Aliou BOLY, Yousra CHABCHOUB, Raja CHIKY, Jacob MONTIEL et Vinh-Thuy TRAN. Isolation based Anomaly Detection for DataStreams using scikit-multiflow. The 20th International Conference on Computational Science and its Applications, ICCSA 2020, Springer.

— **Maurras Ulbricht TOGBE**, Yousra CHABCHOUB, Aliou BOLY et Raja CHIKY. Etude comparative des méthodes de détection d'anomalies. Revue des Nouvelles Technologies de l'Information, Extraction et Gestion des Connaissances, RNTI-E-36 :109–120, 2020

Séminaires et Posters

— **Maurras Ulbricht TOGBE**, Yousra CHABCHOUB, Aliou BOLY et Raja CHIKY. Classification of anomaly detection methods with a focus on Isolation Forest. Workshop Advances in Data Science for Big and Complex Data, Université Paris Dauphine, 2020.

— **Maurras Ulbricht TOGBE**, Yousra CHABCHOUB, Aliou BOLY. Outliers detection methods : A survey. Poster dans Apprentissage à partir de flux de données et séries temporelles : convergences, spécificités et défis partagés, FDST, Télécom ParisTech, 2019.

Chapitre 2

Méthodes de détection d'anomalies

Sommaire

Les résultats obtenus dans ce chapitre ont été publiés dans la conférence internationale EGC 2020 :

Maurras Ulbricht TOGBE, Yousra CHABCHOUB, Aliou BOLY et Raja CHIKY. Etude comparative des méthodes de détection d'anomalies. Revue des Nouvelles Technologies de l'Information, Extraction et Gestion des Connaissances, RNTI-E-36 :109–120, 2020

2.1 Introduction

La détection d'anomalies est un volet du datamining qui intéresse de plus en plus de chercheurs actuellement. On trouve dans la littérature plusieurs définitions de l'anomalie souvent appelée outlier. Dans Hawkins (1980), un outlier est défini par Hawkins comme une observation qui dévie considérablement du reste des autres observations comme si elle était générée par un processus différent. Dans une étude plus récente Dunning et Friedman (2014), Dunning et Friedman affirment que la détection d'anomalie consiste à modéliser ce qui est normal dans le but de découvrir ce qui ne l'est pas.

La détection d'anomalies peut servir à l'amélioration de la qualité des données par suppression ou remplacement des données anormales. L'exploitation de données sans anomalies permet d'aboutir à des résultats et des décisions plus sûres pour plusieurs applications. Dans d'autres contextes, les anomalies traduisent un événement et apportent de nouvelles connaissances utiles. Détecter une anomalie assez à l'avance permet de limiter ou au mieux d'éviter des pertes financières, naturelles ou humaines gigantesques. Par exemple, la détection d'anomalies peut prévenir un dommage matériel et donc inciter à la maintenance prédictive dans le domaine de l'industrie. Elle trouve son application dans plusieurs autres domaines comme la santé, la cybersécurité, la finance, la prédiction des catastrophes naturelles, et bien d'autres domaines.

Les données sont de différentes natures et existent sous plusieurs formes : les données statiques, les flux de données, les données structurées et non structurées, les séries temporelles, les données spatiales, les données univariées et multivariées, etc. La multitude des types de données et leurs caractéristiques différentes impliquent l'existence de méthodes différentes pour la détection d'anomalies, chacune trouvant son efficacité dans un domaine particulier, avec un objectif donné. Ces méthodes utilisent en général un seuil de décision permettant d'isoler les anomalies en se basant sur les différentes techniques comme la classification, le clustering, la régression, les plus proches voisins et les outils statistiques.

Plusieurs critères peuvent être considérés pour comparer ces méthodes et permettent de choisir la méthode la plus adaptée au contexte : l'implication de l'humain (supervisée, non supervisée, semi-supervisée), la nécessité de faire

des hypothèses sur la loi de distribution des données (paramétriques, non-paramétriques, semi-paramétriques), la capacité de traiter des données multivariées et bien d'autres critères. La mesure de la performance de telles méthodes peut s'appuyer sur différents critères comme la précision de la détection, le temps de réponse et le passage à l'échelle (par rapport au volume des données ou au débit du flux).

Comme pour toutes les applications de l'apprentissage automatique, la détection d'anomalies peut être réalisée suivant trois approches : supervisée, semi-supervisée et non supervisée.

Supervisée : La détection d'anomalies est généralement un exercice de classification binaire. Il n'y a que deux classes possibles : normale et anormale, sauf si selon le cas, on décide d'avoir plusieurs classes normales et/ou plusieurs classes anormales. Dès la phase d'entraînement, la méthode reçoit en entrée les différentes classes possibles en fonction des données de l'entraînement. Ainsi, le comportement normal est modélisé et le comportement anormal est également modélisé. Il est donc plus facile de classer les futures données en fonction du modèle correspondant. L'approche supervisée nécessite une connaissance à priori des classes des données d'apprentissage. Il est difficile et coûteux de labéliser les données, particulièrement en ce qui concerne les anomalies. Ces dernières dépendent en effet du domaine applicatif et nécessitent un niveau d'expertise humaine très élevé.

Semi-supervisée : L'approche semi-supervisée implique que le modèle soit créé sur la base de données partiellement labellisées.

Non supervisée : Une méthode non supervisée est celle qui construit son modèle sans aucune connaissance à priori sur les labels des données d'entraînement qui lui ont été fournies. Cette approche est la moins coûteuse car la méthode détecte par elle même les informations dont elle a besoin par analyse des données d'entraînement. Par contre, ces méthodes sont généralement plus lentes vu le travail analytique à faire. En pratique, il n'est pas toujours simple d'avoir les labels ou des informations sur les données, et donc les méthodes non supervi-

sées sont les plus recherchées. Néanmoins, il convient de préciser que le choix d'une approche dépend des données à disposition et des critères que l'utilisateur voudrait respecter.

Dans ce chapitre, nous proposons une classification multicritère des méthodes de détection d'anomalies existantes dans la littérature. Puis nous nous focalisons sur trois méthodes : LOF, OC-SVM et Isolation Forest que nous testons sur deux jeux de données différents.

2.2 Etat de l'art des méthodes

La détection d'anomalies est un sujet qui intéresse beaucoup de chercheurs et qui a fait l'objet de nombreux travaux. Plusieurs méthodes ont été proposées pour la détection d'anomalies et chaque méthode a ses forces et ses faiblesses. Patcha et Park (2007) ont fait une revue des méthodes utilisées pour la détection d'intrusion. Une revue plus générale des techniques existantes couvrant plusieurs approches est proposée dans Aggarwal (2017) et Chandola et al. (2009). M Gupta, Gao, Aggarwal et al. (2014) fait l'état de l'art des méthodes en fonction du type de données considérées : les données temporelles telles que les séries temporelles, les données spatio-temporelles et les flux de données. Salehi et Rashidi (2018), Souiden et al. (2016), Thakkar et al. (2016), Tellis et D'Souza (2018) présentent également des méthodes applicables aux flux de données. Dans le Tableau 2.1, nous présentons une synthèse qui s'appuie sur 10 revues majeures dans la littérature. Nous identifions respectivement les techniques de détection d'anomalies, les types de jeux de données et les domaines d'application abordés dans chacune de ces 10 revues.

Le but de cette section est de fournir un état de l'art complet en agrégeant plusieurs informations sur les différentes méthodes de détection d'anomalies, les jeux de données et les domaines d'applications. Une classification est proposée afin de recommander des méthodes de détection d'anomalies à utiliser selon le type de données dont on dispose (flux de données, série temporelle, graphes...) avec des références bibliographiques pertinentes (Tableau 2.2) et selon l'approche qu'on voudrait utiliser (Figure 4.4). Une exposition des forces et faiblesses de différentes techniques est disponible dans Aggarwal (2017), Chandola et al.

(2009) et Chalapathy et S Chawla (2019).

2.2.1 Types des jeux de données

L'émergence des nouvelles technologies a conduit à la génération de différents types de données. On trouve plusieurs jeux de données ayant des caractéristiques différentes et apportant de nouveaux challenges dans la détection d'anomalies. Nous présentons dans le Tableau 2.1 les types de jeux de données abordés dans les 10 revues étudiées. Différents algorithmes de détection d'anomalies sont appliqués en fonction du type de jeux de données considéré. Nous listons dans le tableau 2.2 les algorithmes les plus utilisés pour chacun des types de jeux de données.

A la différence des séries temporelles, les flux de données sont générés de façon continue et infinie à une vitesse variable. Compte tenu de la taille des données produites, les flux de données ne peuvent pas être exhaustivement stockés pour une exploitation future. Les méthodes de détection d'anomalies dans les flux de données doivent donc être appliquées en temps réel souvent sans aucune connaissance a priori sur la distribution des données (voir M Gupta, Gao, Aggarwal et al. (2014), Salehi et Rashidi (2018) et Tellis et D'Souza (2018)). Le traitement en ligne exige un algorithme de détection de faible complexité pour une exécution plus rapide que la vitesse d'arrivée des données et souvent aussi une faible consommation mémoire si la solution est implémentée sur un équipement à ressources limitées.

2.2.2 Domaines d'application

La détection d'anomalies est transversale à tout domaine qui exploite les données. Ainsi, elle a de nombreuses applications possibles. Les domaines d'application ayant leur spécificité en fonction des données générées ou exploitées, toutes les méthodes de la détection d'anomalies ne sont pas adaptées à tous les domaines d'application. Chandola et al. (2009) et Aggarwal (2017) présentent un état de l'art couvrant plusieurs domaines d'application. Dans M Gupta, Gao, Aggarwal et al. (2014), les auteurs ont fait la revue des méthodes de détection

		1	2	3	4	5	6	7	8	9	10
Techniques	Statistique	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Clustering	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Plus proches voisins	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Classification	✓	✓	✓	✓		✓				
	Régression				✓					✓	
	Approche spectrale			✓	✓						
	Motifs fréquents				✓		✓				
	Deep learning				✓						✓
Type de jeux de données	Flux de données				✓	✓	✓	✓	✓	✓	
	Séries temporelles				✓	✓					✓
	Graphes					✓			✓		
	Grande dimension				✓	✓				✓	
	Séquentielles				✓						
	Spatio-temporelles				✓	✓					✓
	Spatiales				✓						
Domaines d'application	Détection d'intrusion		✓	✓	✓	✓					✓
	Détection de fraude			✓	✓						✓
	Santé			✓	✓	✓					✓
	Maintenance prédictive			✓	✓	✓					✓
	Réseaux de capteurs			✓	✓	✓				✓	✓
	Traitement d'images			✓	✓						✓
	Traitement de texte			✓	✓						
	Données biologiques					✓					
	Astronomie					✓					
	Économie					✓					

Tableau 2.1 – Synthèse de 10 revues existantes : 1- Hodge et Austin (2004) 2- Patcha et Park (2007) 3- Chandola et al. (2009) 4- Aggarwal (2017) 5-M Gupta, Gao, Aggarwal et al. (2014) 6-Souiden et al. (2016) 7-Tellis et D'Souza (2018) 8-Salehi et Rashidi (2018) 9- J Zhang (2013) 10-Chalapathy et S Chawla (2019).

Jeux de données	Méthodes applicables	Références
Flux de données	SmartSifter, AnyOut, CluStream, LEAP, MiLOF, DCLUST, IncLOF, DenStream, Abstract-C, AMCOD,HPStream, WaveCluster, MDEF	Yamanishi et al. (2004), Cao et al. (2006), Y Chen et Tu (2007), Ren et Ma (2009), J Zhang (2013), M Gupta, Gao, Aggarwal et al. (2014), Aggarwal (2017) et Mishra et M Chawla (2019)
Séries temporelles	ARMA, ARIMA, VARMA, CUSUM, EWMA, LSA, MLP, ART NN, AE,GAN	M Gupta, Gao, Aggarwal et al. (2014) et Chalapathy et S Chawla (2019)
Graphes	DBMM, ECOutlier, NetSpot, ParCube, Com2, NetSmile, DeltaCon	M Gupta, Gao, Sun et al. (2012), M Gupta, Gao, Aggarwal et al. (2014), Salehi, CA Leckie et al. (2014) et Salehi et Rashidi (2018)
Grande dimension	GLOF, HighDoD, SOF, CLIQUE, HPStream, ABOD, SOD, SPOT	Domingues et al. (2018) et Mishra et M Chawla (2019)
Séquentielles	CLUSEQ, TARZAN	Aggarwal (2017)
Spatio-temporelles	Outstretch, TRAOD, LSTM, CNN	E Wu et al. (2008) et Chalapathy et S Chawla (2019)
Données spatiales	Moran scatterplot, DBSCAN	Ester et al. (1996) et El Sibai et al. (2018)

Tableau 2.2 – Méthodes applicables pour chaque type de jeu de données.

d'anomalies temporelles applicables dans différents domaines. La détection d'intrusion, décrite dans Chandola et al. (2009), se base sur l'analyse d'une cible, généralement un réseau ou un hôte, pour détecter les comportements anormaux. Il s'agit en effet de tentatives frauduleuses d'accès à une ressource par violation de la sécurité mise en place pour la cible en question Aggarwal (2017). La détection de fraude permet de repérer les activités suspectes menées par un individu généralement sous une fausse identité (identité usurpée). Le Tableau 2.1 liste plusieurs revues ayant abordé différents domaines d'application tels que la détection d'intrusion, la détection de fraude, la santé, la maintenance

prédictive dans l'industrie, les réseaux de capteurs, le traitement d'images, le traitement de texte, les données biologiques, l'astronomie et l'économie. La détection d'anomalies dans les données textuelles (comme les logs) intéresse de nombreux chercheurs surtout avec l'utilisation accrue des réseaux sociaux. Différentes méthodes existent pour la détection d'anomalies dans les données textuelles. Après avoir fait un état de l'art détaillé des méthodes existantes, Mohotti et Nayak proposent dans Mohotti et Nayak (2020) une méthode qui se base sur la fréquence et le rang des mots pour détecter les anomalies.

2.2.3 Techniques de détection d'anomalies

Les techniques de détection d'anomalies existantes se basent sur deux propriétés importantes des anomalies : elles sont rares et elles présentent un comportement très différent de la majorité des données. Parmi ces techniques, nous distinguons : les méthodes statistiques, les méthodes du machine learning classique, celles du deep learning et d'autres méthodes. Dans la Figure 4.4, nous présentons une synthèse de cette classification avec des exemples d'algorithmes appartenant à chacune de ces catégories.

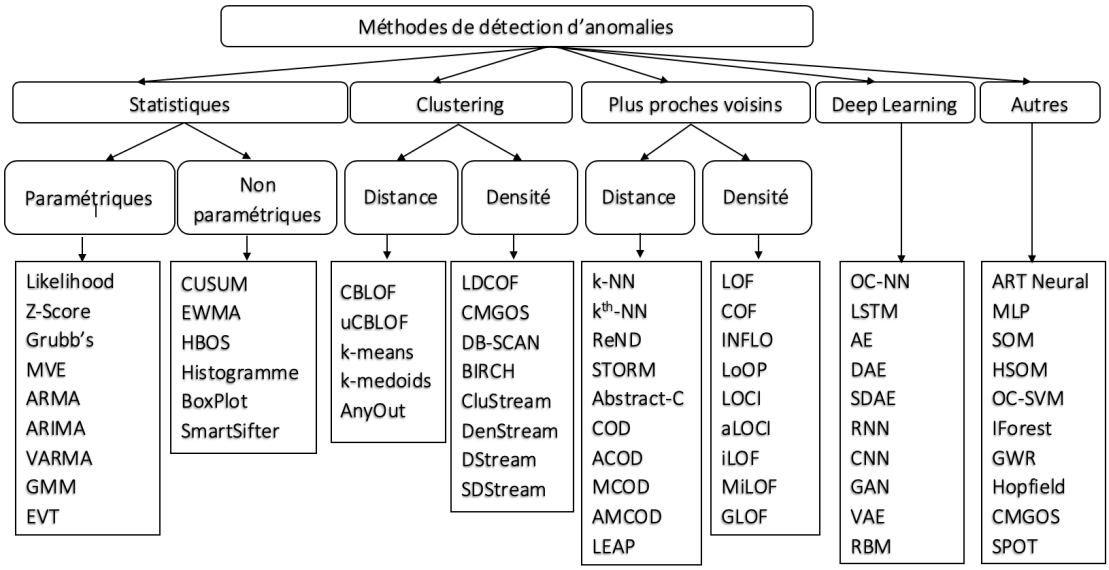


Figure 2.1 – Classification des différentes techniques de détection d'anomalies.

2.2.4 Méthodes statistiques

Les méthodes statistiques construisent un modèle avec un intervalle de confiance à partir des données existantes. Les nouvelles données qui ne correspondent pas à ce modèle seront considérées anormales (voir Desforges et al. (1998), Unkel et al. (2012) et Aggarwal (2017)). Comme le montre la figure 4.4, elles peuvent être paramétriques ou non paramétriques. A la différence de l'approche non paramétrique, l'approche paramétrique suppose une connaissance a priori de la distribution des données. Les cartes de contrôle tels que CUmulative SUM (CUSUM) Lucas et Crosier (2000) et Exponentially Weighted Moving Average (EWMA) Chabchoub, Chiky et al. (2014) ainsi que les méthodes SmartSifter Yamanishi et al. (2004), AutoRegressive–Moving-Average (ARMA) Torres et al. (2005), etc. font partie des méthodes les plus utilisées. Aggarwal (2017) décrit plusieurs méthodes statistiques de détection d'anomalies. Dans Unkel et al. (2012), les auteurs ont classé plusieurs méthodes statistiques suivant leur mode de fonctionnement. Bien que ces méthodes soient adaptées à la détection d'anomalies dans les données évolutives comme flux de données et séries temporelles, elles nécessitent des connaissances à priori sur les données ce qui constitue une limite considérable. De plus, ces méthodes sont performantes dans un environnement de données à faible dimension.
La figure 4.4 présente d'autres groupes de méthodes qui sont principalement des méthodes de machine learning dont le fonctionnement et les besoins diffèrent des méthodes statistiques.

2.2.5 Deep learning

Elles représentent une classe d'algorithmes d'apprentissage automatique supervisé, semi-supervisé ou non supervisé basés sur l'utilisation de plusieurs couches d'unité de traitement non linéaire. Parmi ces méthodes, on peut citer celles basées sur les auto-encoders (AE), Recurrent Neural Network (RNN) et Long short-term memory (LSTM). Plusieurs revues font l'état de l'art des méthodes de détection d'anomalies basées sur le deep learning (Chalapathy et S Chawla (2019), Kwon et al. (2019), R Wang et al. (2020) et Pang, Shen et al. (2021)). Les méthodes de deep learning sont aussi performantes pour la détec-

tion d'anomalies bien qu'elles nécessitent une grande disponibilité de ressources (mémoire, puissance de calcul, temps pour traiter les données). Plusieurs frameworks et outils sont conçus pour faciliter l'utilisation de ces méthodes. Les méthodes de deep learning seront donc privilégiées pour des jeux de données de très grande dimension avec à disposition les ressources nécessaires.
Les méthodes du machine learning classique sont quant à elles beaucoup plus faciles d'accès et fournissent également de très bons résultats dans divers contextes. Dans la littérature, on rencontre de nombreuses méthodes du machine learning classique. Quelques unes sont classées dans la figure 4.4 selon leur mode de fonctionnement (Clustering, plus proche voisins, etc.).

2.2.6 Machine learning

2.2.6.1 Les techniques basées sur la proximité

Elles regroupent celles basées sur les plus proches voisins et celles basées sur le clustering. Les techniques basées sur les plus proches voisins déterminent pour une observation o ses k plus proches voisins à travers le calcul de la distance entre toutes les observations du jeu de données. Ces méthodes nécessitent un calcul préalable, et de ce fait, elles sont coûteuses en temps d'exécution. Il existe deux approches de méthodes basées sur les plus proches voisins : l'approche basée sur la distance Angiulli et Pizzuti (2002) et Yamanishi et al. (2004) et l'approche basée sur la densité Breunig et al. (2000). Les techniques de clustering ont pour objectif principal de diviser le jeu de données en clusters contenant les données qui ont des comportements similaires. On distingue deux approches dans ces techniques : l'approche basée sur la distance selon laquelle le cluster le plus éloigné représente une anomalie et l'approche basée sur la densité qui définit l'anomalie par le cluster qui contient le moins de données.

Local Outlier Factor (LOF) est une méthode phare de la détection d'anomalies locales basée sur la densité de l'observation en question par rapport à la densité de ses plus proches voisins. Proposée par Breunig et al. (2000), LOF est une méthode non supervisée qui donne un score représentant le degré d'aberrance de l'observation. Les observations dont le degré d'aberrance est largement supérieur

à 1 sont considérées comme anomalies. La méthode prend en paramètre le nombre de plus proches voisins à considérer k, et calcule le degré d'aberrance d'un point p par la formule suivante : $LOF_k(p) = \frac{\sum_{o \in N(p,k)} \frac{lrd_k(o)}{lrd_k(p)}}{k}$ avec $N(p,k)$ l'ensemble des k plus proches voisins de p. $lrd(p)$ est la densité d'accessibilité locale de p et correspond à l'inverse de la distance d'accessibilité moyenne entre p et ses k plus proches voisins : $lrd_k(p) = \frac{k}{\sum_{o \in N(p,k)} reach\_dist_k(p,o)}$ La distance d'accessibilité de p par rapport à o considérant les k plus proches voisins ($reach\_dist_k(p,o)$) entre deux points (p et o), correspond au maximum entre la distance entre le $k^{ième}$ plus proche voisin de o ($k - distance(o)$) et la distance entre p et o ($d(p,o)$), soit : $reach\_dist_k(p,o) = \max(k - distance(o), d(p,o))$. La figure 2.2 montre le calcul de $reach\_dist_k(p_1,O)$ et $reach\_dist_k(p_2,O)$ pour $k = 5$.

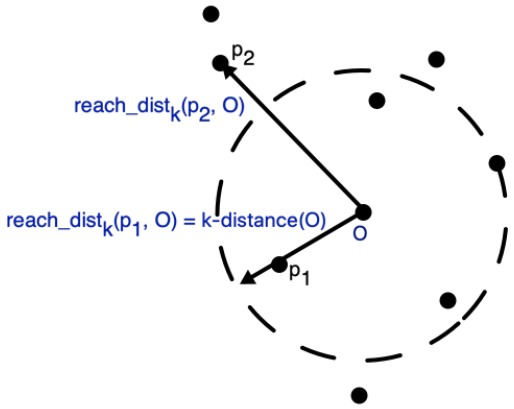


FIGURE 2.2 – $reach\_dist_k(p_1,O)$ et $reach\_dist_k(p_2,O)$ pour $k = 5$.

On trouve dans la littérature différentes améliorations de LOF comme Incremental LOF «iLOF» (proposé dans POKRAJAC et al. (2007)) qui est adaptée aux flux de données, mais qui consomme beaucoup de mémoire pour le calcul de la densité des nouvelles données entrantes. Memory Efficient ILOF «MILOF» (SALEHI, C LECKIE et al. (2016)) est une évolution de iLOF qui réduit la consommation mémoire tout en ayant une précision similaire à iLOF. Pour la détection

d'anomalies dans les jeux de données de grandes dimensions, Lee et Cho (2016) ont proposé Grid-LOF «GLOF» qui divise le jeu de données en petites régions (Grid) avant de calculer la densité.

2.2.6.2 One-Class Support Vector Machine

One-Class Support Vector Machine (OC-SVM) est une méthode de détection d'anomalies qui applique des algorithmes de SVM au problème de One class classification (OCC) proposée par Schölkopf, Williamson et al. (2000) et Schölkopf, Platt et al. (2001). Le séparateur à vaste marge (SVM) appelé aussi machine à vecteurs de support est très utilisé pour l'apprentissage automatique du fait de sa puissance et de sa polyvalence (classification linéaire, non-linéaire, régression). OCC est une approche de classification semi-supervisée qui consiste à repérer toutes les observations appartenant à une classe précise connue pendant l'apprentissage, dans tout le jeu de données. L'idée clé de cette méthode est de trouver un hyperplan dans un espace de grande dimension qui sépare les anomalies des données normales. La figure 2.3, présente un exemple du fonctionnement de la méthode OC-SVM pour la classification.

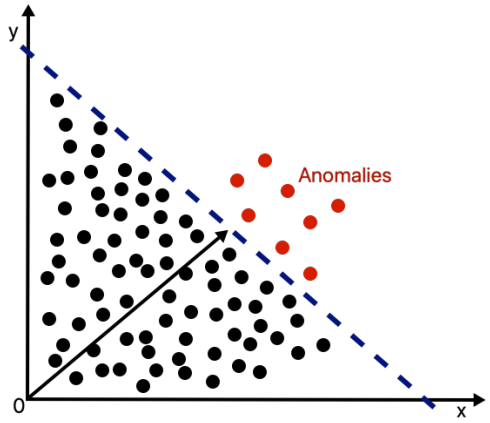


Figure 2.3 – One-Class Support Vector Machine.

2.2.6.3 Isolation Forest ou IForest

FT Liu et al. (2008) et FT Liu et al. (2012) est une méthode basée sur les arbres de décision et les forêts aléatoires. Elle utilise l'isolation d'observations à partir de la construction de plusieurs arbres aléatoires. Quand une forêt d'arbres aléatoires et indépendants produit collectivement un chemin d'accès court pour atteindre une observation depuis la racine, celle-ci a une forte probabilité d'être une anomalie. Isolation Forest fait partie des méthodes de détection d'anomalies les plus récentes et les plus utilisées. Elle donne également de bons résultats pour les jeux de données de grandes dimensions. La section suivante (2.2.7) présente plus en détails la méthode IForest.

2.2.7 Isolation Forest

2.2.7.1 Principes et avantages de Isolation Forest

L'isolation consiste à mettre à part une ou plusieurs données spécifiques dans tout le jeu de données. Dans le cadre de la détection d'anomalies, les jeux de données sont généralement déséquilibrés. En effet, les données normales sont beaucoup plus nombreuses que les données anormales, l'anomalie étant l'exception. La technique d'isolation se base sur deux caractéristiques principales des données anormales.

- **Les données anormales sont peu nombreuses.** Les données anormales sont par définition rares.
- **Les données anormales sont distinctes.** Les données anormales ont par essence un comportement assez différent de celui des données normales. De plus, elles sont plus faciles à identifier quand elles sont distinctes entre elles, ce qui évite l'effet de recouvrement appelé "masking".

Une conséquence directe de ces deux caractéristiques est que les données anormales sont plus faciles à isoler que les données normales.

La majorité des approches de détection d'anomalies existantes créent un modèle à partir des données existantes, soit en se basant sur des connaissances acquises sur les données non labélisées pour les méthodes non supervisées, ou à l'aide de la labélisation faite par l'être humain (identification préalable d'une

classe) pour les méthodes semi-supervisées et supervisées. La modélisation du comportement des données normales (majoritaires) permet par la suite d'identifier les données anormales comme étant les données qui ne respectent pas ce comportement. Certaines approches calculent la distance ou la densité des différentes données. Bien que donnant de bons résultats, ces méthodes souffrent généralement d'un problème de passage à l'échelle par rapport à la dimension ou la taille du jeu de données. En effet, face à un jeu de données massif, le temps d'exécution et la consommation mémoire d'une telle méthode deviennent rapidement non satisfaisants. Généralement, les méthodes de détection d'anomalies basées sur les plus proches voisins comme Local Outlier Factory (LOF Breunig et al. (2000)), k Nearest Neighbors (k-NN Keller et al. (1985)), etc. ont une complexité quadratique $O(n^2)$ Goldstein et Uchida (2016), comme elles sont basées sur le calcul de distance entre chaque paire de données. Le temps de réponse de ces méthodes est aussi non adapté au traitement en temps réel des flux de données produits aujourd'hui par différents systèmes et capteurs avec des débits de plus en plus élevés. La modélisation basée sur le calcul de distance entre toutes les données, par exemple, voit rapidement ses limites dans de telles conditions.

Isolation Forest (IForest) est une méthode de détection d'anomalies basée sur une approche différente des autres (statistique, clustering, plus proches voisins, etc.). Elle ne calcule ni la distance, ni la densité et de ce fait, elle réduit significativement le temps d'exécution et la consommation mémoire. IForest a une faible et constante consommation mémoire et un temps d'exécution linéaire, proportionnel au nombre de données considérées. Son passage à l'échelle est remarquable et rend cette méthode adaptée aux jeux de données de grande dimension et de grande taille, ainsi qu'au traitement en temps réel.

Les anomalies sont par définition des données peu nombreuses et ayant un comportement très différent de celui des données normales. La figure 2.4 montre un exemple de donnée anormale (X_0) et un exemple de donnée normale (X_i). L'isolation se fait par scissions successives du jeu de données. On remarque que les données anormales sont plus faciles à isoler que les données normales. X_0 a été isolée en 3 étapes alors qu'il en a fallu 11 pour isoler la donnée normale (X_i).

Isolation Forest (IForest) (FT Liu et al. (2008) et FT Liu et al. (2012)) est la

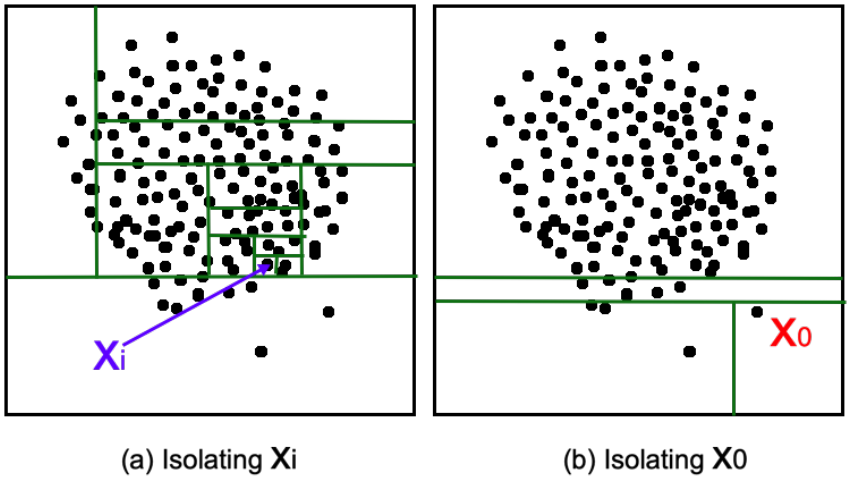


FIGURE 2.4 – Les données anormales (X_0) sont plus rapidement isolées que les données normales (X_i).

première méthode qui a été proposée dans la catégorie des méthodes de détection d'anomalies basées sur l'isolation. Cette méthode utilise un ensemble d'arbres (iTree) aléatoires et indépendants appelés forêt aléatoire. IForest calcule un score pour chaque donnée en faisant intervenir tous les arbres de la forêt. IForest utilise deux paramètres d'entrée pour calculer le score des données. Il s'agit de ψ qui est la taille de l'échantillon aléatoirement choisi dans tout le jeu de données et de t qui est le nombre d'arbres dans la forêt. Chaque arbre étant construit indépendamment des autres et sur la base de son propre échantillon, le nombre d'arbres correspond au nombre d'échantillons.

IForest comporte deux étapes : la phase d'entraînement qui correspond essentiellement à la construction de la forêt et la phase dite de scoring où on calcule le score de chaque donnée du jeu de données.

Soit $X(n,m) \subset \mathbb{R}^m$ un jeu de données contenant des anomalies, n désigne le nombre de données dans X et m la dimension des données.

2.2.7.2 Phase d'entraînement

L'objectif principal de la phase d'entraînement est de construire une forêt d'arbres aléatoires et indépendants (iTree). ψ et t sont des paramètres clés de

IForest. IForest sélectionne aléatoirement un sous-ensemble de ψ données qui constituent l'échantillon dédié à la construction d'un arbre. Cette étape est très importante, car elle est la base de toute la performance et l'efficacité de IForest. IForest choisit aléatoirement ψ données de X sans remise. Il est important de noter que chaque échantillon est utilisé uniquement pour un seul arbre. Vu que les données sont choisies aléatoirement et que les données anormales sont supposées être très peu nombreuses relativement aux données normales, l'échantillon pourrait ne contenir que des données normales ou un mélange majoritairement constitué de données normales.

L'arbre d'isolation (iTree) est un arbre binaire. La construction de l'arbre se fait comme suit : Au départ, le nœud racine contient toutes les données de l'échantillon. Lors de la construction de l'arbre, tous les nœuds internes sont scindés en deux (left et right) jusqu'à ce que toutes les données de l'échantillon soient isolées ou que l'arbre atteigne sa profondeur maximale : $max\_depth = log_2(\psi)$. On dit qu'une donnée est isolée quand elle est seule dans son noeud, comme on peut le voir sur la figure 2.4 où X_0 et X_i ont respectivement été isolées après trois et onze scissions. Pour scinder un nœud i, IForest choisit de manière aléatoire une dimension d_i (splitDimension). Puis, la valeur de scission v_i (splitValue) est également choisie de manière aléatoire entre la valeur minimale ($\min(d_i)$) et la valeur maximale ($\max(d_i)$) des données du nœud en question pour la dimension d_i. Les éléments du noeud i sont alors divisés en deux groupes (left et right) en comparant leurs valeurs à v_i. Le pseudo-code de la phase d'entraînement est donné dans l'algorithme 1. La figure 2.5 montre un exemple de $iTree$.

Pour construire les t arbres de la forêt, ces deux étapes (échantillonnage et construction d'un arbre) sont répétées t fois. Ainsi, chaque arbre a son échantillon dédié. Pour chaque iTree, l'échantillon est choisi dans tout le jeu de données X. La complexité de la phase d'entraînement est donnée par $O(t\psi \log \psi)$ car les ψ données (échantillon) de chacun des t arbres doivent être isolées ou presque dans l'arbre concerné. Le nombre d'arbres t est un paramètre déterminant pour la stabilité des résultats de IForest.

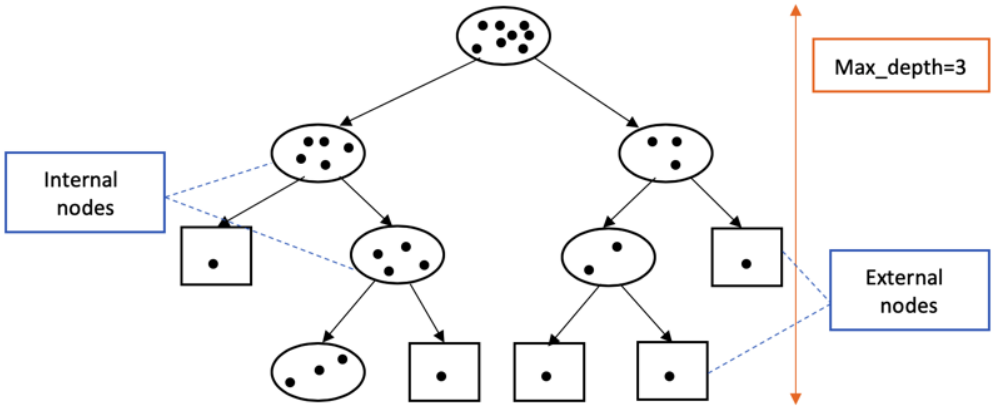


FIGURE 2.5 – Exemple d'arbre ($iTree$) avec $\psi = 8$.

2.2.7.3 Phase de scoring

Pendant la phase de scoring, le score de chaque instance du jeu de données X est calculé. Ce score représente le degré d'aberrance de la donnée. Pour calculer ce score, la donnée soit x, devra passer dans chaque arbre de la forêt. La donnée échouera certainement dans un nœud externe de chaque arbre, dépendant des critères de scission. Le nombre de nœuds parcourus par la donnée depuis le nœud racine pour atteindre son noeud externe est appelé longueur de chemin de x et noté $h(x)$. Le pseudo-code du calcul de la longueur de chemin d'une donnée dans un arbre est donné dans l'algorithme 2. Une fois la donnée passée par tous les arbres de la forêt, IForest calcule la longueur moyenne des t chemins de x notée $E(h(x))$. En utilisant un résultat connu sur la recherche dans un arbre binaire (Binary Search Tree - BST), les auteurs proposent de calculer le score $s(x,n)$ de x avec la formule suivante :

$$s(x,n) = 2^{-\frac{E(h(x))}{C(n)}};$$

$C(n) = 2H(n-1) - (2(n-1)/n)$ est la longueur moyenne des chemins d'une recherche infructueuse dans un arbre binaire (Binary Search Tree). $H(i) = ln(i) + 0,5772156649$ (constante d'Euler) est une valeur harmonique. Notons que $C(n)$ est utilisé pour une simple normalisation de $h(x)$.

Avec cette formule du score, les auteurs classent les données comme suit :

— Quand $E(h(x)) \to C(n)$, $s \to 0.5$. Si toutes les instances ont un score $s \approx 0.5$ alors le jeu de données ne contient pas d'anomalies réellement identifiables ;
— Quand $E(h(x)) \to 0$, $s \to 1$. Si une donnée a un score s très proche de 1 alors elle est une anomalie ;
— Quand $E(h(x)) \to n-1$, $s \to 0$. Si une donnée a un score s très inférieur à 0.5 alors elle est une donnée normale.

Une longueur moyenne de chemin relativement courte $E(h(x))$ implique que la forêt de t arbres classe globalement x comme une anomalie. La complexité de la phase de scoring est donnée par $O(nt \log \psi)$ car chaque donnée du jeu de données de taille n sera traitée par les t arbres de la forêt pour évaluer la longueur de son chemin ou encore sa profondeur et générer ainsi sa profondeur moyenne puis son score. Au final, IForest a une complexité linéaire, proportionnelle à la taille du jeu de données (n), car t et ψ sont des constantes.

Algorithme 1 : iTree(X, e, l) - Construction d'un arbre (iTree)

Input : X - l'ensemble des données dans le noeud, e - la longueur actuelle de l'arbre, max_depth - la longueur maximale de l'arbre
Output : iTree

```
if |X| <= 1 || e >= max_depth then
    return NoeudExterne(size = |X|)
else
    Q ← liste des attributs dans X
    q ← choix aléatoire d'un attribut dans Q (q ∈ Q)
    p ← choix aléatoire d'une valeur entre le min et le max des valeurs de
      x pour q
    l ← filter(X, q < p)
    r ← filter(X, q >= p)
    return NoeudInterne(left ← iTree(l, e+1, max_depth), right ← iTree(r,
      e+1, max_depth), splitDimension ← q, splitValue ← p )
```

2.2.7.4 Le temps d'exécution

Le temps d'exécution d'une méthode correspond au temps consommé par chacune de ces deux phases. La complexité temporelle d'IForest est linéaire,

Algorithme 2 : LongueurChemin(x, T_i, e)

Input : x - une donnée, T_i - un arbre, e - la longueur actuelle du chemin.
/* e est initialisé à 0 au premier appel */
Output : La longueur du chemin de x dans l'arbre T_i

```
1 if T_i est un noeud externe then
2     return e + C(T_i.size) /* C(n) défini dans 2.2.7.3 */
3 q ⟵ T_i.splitDimension
4 if x_q < T_i.splitValue then
5     return LongueurChemin(x, T_i.left, e + 1)
6 else
      /* x_q >= T_i.splitValue */
7     return LongueurChemin(x, T_i.right, e + 1)
```

au pire, égale à $O(t\psi^2)$ pour la phase d'apprentissage et $O(nt\psi)$ pour la phase de test (voir FT Liu et al. (2012)). n est la taille du jeu de données, t correspond au nombre d'arbres dans la forêt et ψ est la taille de l'échantillon utilisé pour construire un arbre. Notons que la durée de la phase d'apprentissage est constante, indépendamment de la taille totale des données.

2.2.7.5 Consommation mémoire

IForest a un faible besoin en mémoire égal à $O(t\psi)$ (voir FT Liu et al. (2012)). La mémoire consommée est donc constante, indépendamment de la taille du jeu de données, ce qui représente un avantage considérable de la méthode pour traiter de grandes données.

Le swamping et le masking représentent un vrai défi pour IForest qui les gère jusqu'à une certaine limite. Le swamping consiste à identifier à tort des données normales comme des anomalies à cause d'une faible différence entre ces deux types de données. Lorsque les données normales sont très proches des anomalies, le nombre de partitions nécessaires pour séparer les anomalies augmente, ce qui rend plus difficile la distinction entre les anomalies et les données normales. Le masking est l'existence d'un grand nombre d'anomalies dissimulant leur propre présence. Lorsqu'un groupe d'anomalies est grand et dense, le nombre de partitions nécessaires pour isoler chaque anomalie augmente également FT Liu

et al. (2008). Ces données anormales risquent ainsi d'être considérées comme normales. Le fait que IForest se base sur des échantillons aléatoires et non sur tout le jeu de données pour construire la forêt d'arbres aléatoires permet d'atténuer le masking et le swamping. En effet, chaque arbre de la forêt est construit avec son propre échantillon. Les arbres n'isolent pas forcément les mêmes anomalies et certains échantillons pourraient même ne contenir aucune anomalie. D'où la robustesse de IForest face aux effets swamping et masking.

2.3 Etude comparative de quelques méthodes

Dans cette partie, nous avons fait le choix de comparer expérimentalement trois méthodes de détection d'anomalies non supervisées très utilisées dans la littérature : LOF, OC-SVM et IForest. En effet, il s'agit de méthodes performantes appartenant à des catégories différentes. Ces trois méthodes donnent de très bons résultats comparées aux autres méthodes de leurs approches respectives. De plus, elles sont souvent utilisées comme références de comparaison pour évaluer les performances des nouvelles méthodes de détection d'anomalies.

Cette section est organisée comme suit : nous présentons d'abord tous les jeux de données utilisés dans le manuscrit, même les jeux de données qui ne sont pas concernés par les expérimentations de ce chapitre. En effet certains jeux de données sont communs à plusieurs chapitres, et pour faciliter la lecture de ce manuscrit, tous les jeux de données sont décrits dans une même section. Dans ce même ordre d'idées toutes les métriques de détection d'anomalies sont présentées dans une même section. Enfin nous détaillons les résultats comparatifs des trois méthodes choisies.

2.3.1 Jeux de données

Les jeux de données réels considérés sont publics et souvent utilisés dans le domaine de la détection des anomalies. Il s'agit de données labélisées, contenant des anomalies identifiées. Ces jeux de données proviennent de l'UCI Machine Learning Archive Dua et Graff (2017), possédée par l'Université de Californie. Cette archive est une ressource très utile et représente une base commune pour

tester, valider et comparer différents algorithmes d'apprentissage automatique.

Nous avons sélectionné différents jeux de données présentant des caractéristiques différentes en termes de taille, de nombre de dimensions et de taux d'anomalie, comme le montre le tableau 6.1.

2.3.1.1 HTTP et SMTP

Il s'agit de jeux de données extraits du grand jeu de données de la compétition KDD-CUP99. Ils décrivent des intrusions dans les réseaux.

Les jeux de données HTTP et SMTP ont été prétraités pour ne conserver que quatre attributs pertinents (service, durée, src-bytes, dst-bytes) parmi les 41 attributs originaux. Pour ces deux ensembles de données, les anomalies font référence à des attaques de réseaux. La méthode de prétraitement est décrite dans Ding et Fei (2013). Le jeu de données original KDD-CUP99 contient 4 898 431 données, dont 3 925 651 attaques (80,1%). Ce qui représente une très grande proportion. Pour mieux adapter le jeu de données au contexte de détection d'anomalies, les auteurs se sont limités à un petit sous-ensemble des anomalies. Le jeu de données ainsi obtenu est décrit par Ding et al dans Ding et Fei (2013). HTTP et SMTP ont été utilisés par plusieurs chercheurs dans plusieurs études comme : FT Liu et al. (2008), Ding et Fei (2013)... etc. KDD-Cup99 HTTP est la version publiée par Goldstein et Uchida (2016) après quelques simplifications du jeu de données original KDD-Cup99 (Lazarevic et al. (2003)). Cette variante de Goldstein et Uchida est disponible à `http://dx.doi.org/10.7910/DVN/OPQMVF`

2.3.1.2 Shuttle

Shuttle est un jeu de données de 9 dimensions collectés par la NASA, contenant des informations sur les radars. C'est le jeu de données qui contient la plus grande proportion d'anomalies, parmi tous les jeux de données utilisés dans ce manuscrit. Statlog Shuttle est une variante obtenu par Goldstein et Uchida (2016) à partir du jeu de données original (Martin et al. (2007)) après réduction du nombre d'anomalies. Cette variante de Goldstein et Uchida est disponible à `http://dx.doi.org/10.7910/DVN/OPQMVF`

2.3.1.3 Forest Cover

Forest Cover est un jeu de données décrivant les arbres dans quatre zones différentes de la Roosevelt National Forest au Colorado. Il contient 10 dimensions donnant des informations sur le type d'arbre, la couverture d'ombre, la distance par rapport aux points de repère proches (routes, etc.), le type de sol ainsi que la topographie locale.

2.3.1.4 Mammography

Le jeu de données Mammography présente des données pour la classification des mammographies bénignes ou malignes. Il contient 6 attributs avec 2,32% d'anomalies.

D'autres détails sur la taille des jeux de données et le taux d'anomalies sont résumés dans le tableau 6.1.

Tableau 2.3 – Jeux de données publiques

	Size (n)	Dimensions	Anomalies
HTTP	567 498	3	0.39%
SMTP	95 156	3	0.03%
Shuttle	49 097	9	7.15%
Forest Cover	286 048	10	0.96%
Mammography	11 183	6	2.32%
Statlog Shuttle	46 464	10	1,89%
KDD-Cup99 HTTP	103 351	30	0,17%

Les trois méthodes de détection d'anomalies choisies dans ce chapitre étant non supervisées (IForest, LOF et OC-SVM), les labels présents dans les jeux de données ne sont pas exploités dans la phase d'apprentissage. Ils sont utilisés après la détection pour évaluer l'efficacité de la méthode via différentes métriques comme la spécificité, le recall, ROC AUC...

2.3.2 Métriques des méthodes de détection d'anomalies

Dans le domaine de l'apprentissage automatique et plus particulièrement de la détection d'anomalies, plusieurs métriques peuvent être considérées pour

évaluer et comparer les différentes approches. En général, le temps d'exécution et la consommation mémoire sont des critères importants pour aborder la scalabilité de la méthode, en particulier dans le contexte des flux de données avec un débit toujours croissant. Plus spécifiquement pour le Machine Learning, d'autres critères tels que le rappel, la spécificité, la précision, le score F1, l'aire sous la courbe ROC sont à prendre en compte pour évaluer la performance de la méthode.

Dans le cas de la détection d'anomalies, qui est un exercice de classification binaire dans des ensembles de données déséquilibrés, il n'y a que deux classes possibles : normale ou anormale. Dans nos travaux, nous avons utilisé le framework scikit-learn Pedregosa et al. (2011). Cependant, contrairement aux définitions implémentées dans scikit-learn, nous considérons ici la convention suivante, bien connue et adoptée par la communauté de détection des anomalies : la classe normale est désignée par Négatif et la classe anormale par Positif. Nous avons utilisé l'API de scikit-learn pour calculer le rappel, l'AUC du ROC et le score F1 à partir de la matrice de confusion.

2.3.2.1 La matrice de confusion

La matrice de confusion contient le nombre de :

— vrais positifs : TP (True Positives)
— vrais négatifs : TN (True Negatives)
— faux positifs : FP (False Positives)
— faux négatifs : FN (False Negatives)

A partir de ces 4 grandeurs nous avons générés les métriques suivantes :

2.3.2.2 Rappel

Dans certains domaines d'application, on préfère se concentrer sur les anomalies, plus précisément sur le taux de données anormales bien classées (parmi toutes les anomalies réelles). Dans ce cas, la métrique la plus adaptée est le rappel (recall). C'est la métrique à considérer quand la non-détection d'une anomalie est importante. L'impact de rater une anomalie doit être pris en compte dans le choix de la meilleure métrique.

$$Recall = \frac{TP}{TP+FN}.$$

2.3.2.3 Précision

La précision dépend des vrais positifs (les vrais alarmes) et des faux positifs (les fausses alarmes). Bien que la précision soit utile pour évaluer la capacité de classification d'une méthode, dans certains cas, ce n'est pas la meilleure métrique à considérer. En particulier dans le cas d'un ensemble de données déséquilibré où les données normales sont manifestement plus importantes que les données anormales. Dans ce cas, le nombre de faux positifs (fausses alarmes) est souvent plus important que le nombre de vrais positifs. La précision est la métrique à considérer si on veut surtout éviter les fausses alarmes.

$$Precision = \frac{TP}{TP+FP}$$

2.3.2.4 Spécificité

La spécificité est utile pour la détection des anomalies, car elle permet d'évaluer les performances de la méthode dans la détection des données normales. La précision et la spécificité sont symétriques et se focalisent respectivement sur la classe des données anormales et normales.

$$Specificity = \frac{TN}{TN+FN}$$

2.3.2.5 Taux de fausses alarmes

Le taux de fausses alarmes (FAR) est le rapport entre le nombre de données normales classées comme anomalies et le nombre total de données normales réelles. L'objectif de toute méthode de détection d'anomalie est de trouver toutes les anomalies tout en minimisant le taux de fausses alarmes. Cette métrique est couramment utilisée pour évaluer l'efficacité de différentes méthodes de détection des anomalies. Il y a toujours un compromis entre la détection de vraies anomalies et les fausses alarmes qui sont tolérées jusqu'à une certaine limite en fonction du contexte.

$$FAR = \frac{FP}{FP+TN} = 1 - specificity.$$

2.3.2.6 F1 score

Pour comparer deux méthodes différentes, nous pouvons également utiliser le score F1. Il s'agit de la moyenne harmonique de la précision et du rappel. C'est une métrique intéressante lorsque les classes sont déséquilibrées. C'est le cas du contexte de détection des anomalies où les anomalies sont beaucoup moins nombreuses que les données normales.

$$F1_{score} = 2 \times \frac{precision \times recall}{precision + recall}$$

2.3.2.7 L'Aire sous la courbe ROC - ROC AUC

L'aire sous la courbe ROC (ROC AUC) est souvent utilisée pour quantifier l'efficacité de la méthode de détection des anomalies. La ROC AUC augmente avec l'efficacité de la détection et reste bornée par 1 (voir figure 2.6). La diagonale représente le tracé d'une méthode purement aléatoire.

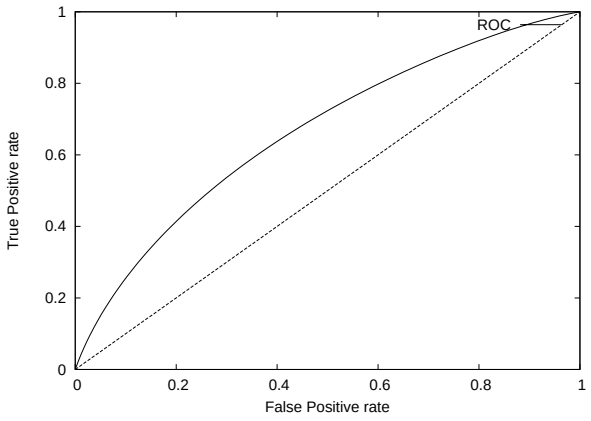


Figure 2.6 – Courbe ROC.

2.3.2.8 Le temps d'exécution

Dans le contexte spécifique de l'apprentissage automatique, les algorithmes sont composés d'une phase d'apprentissage suivie d'une phase de test. Le temps

d'exécution d'une méthode correspond au temps total d'exécution de ces deux phases. Le temps d'exécution est très dépendant de la complexité de l'algorithme.

2.3.2.9 Consommation mémoire

Les contraintes sur la mémoire disponible est spécifique à certains contextes comme l'IoT ou le Edge computing ou les algorithmes sont exécutés sur des capteurs et des équipements ayant des ressources limités. La consommation mémoire est une mesure importante dans un tel contexte.

2.3.3 Résultats

Afin d'évaluer les performances des méthodes choisies en fonction de la dimension du jeu de données, nous avons utilisé deux jeux de données (KDD-Cup99 HTTP et Shuttle) précédemment décrits (Tableau 6.1). Nous nous intéressons aux quatre métriques suivantes :

— L'aire sous la courbe ROC (ROC AUC) qui est un standard dans la comparaison des performances des méthodes de détection d'anomalies,
— La spécificité
— Le rappel
— Le temps d'exécution

Le tableau 2.4 résume les résultats des trois méthodes sur les deux jeux de données considérés.

Jeu de données	Rappel	Spécificité	ROC AUC	CPU Time(s)
IForest(SSh)	**0.98**	**0.92**	**0.95**	**3.86**
IForest(KDD)	**0.90**	**1**	**0.95**	**28.92**
LOF(SSh)	0.38	0.91	0.65	32.18
LOF(KDD)	**0.90**	0,67	0.79	35.98
OC-SVM(SSh)	0.95	0.60	0.78	253.67
OC-SVM(KDD)	0.58	0.51	0.54	4615.74

Tableau 2.4 – Tableau comparatif de la performance de IForest, LOF et OC-SVM sur les jeux de données Shuttle(SS) et KDD-Cup99 HTTP(KDD).

IForest a une durée d'exécution plus petite que LOF et OC-SVM pour les deux jeux de données de tailles et de dimensions différentes. Cela s'explique par le fait que IForest ne calcule ni la distance entre les points ni la densité des points. IForest détecte un grand nombre d'anomalies avec beaucoup moins de fausses alertes que OC-SVM qui est dépassé par LOF. LOF laisse passer beaucoup d'anomalies (plus de 60% pour shuttle) mais fait beaucoup moins de fausses alertes que OC-SVM qui par contre détecte 95% des anomalies de Shuttle. Il faut noter également qu'en terme de temps d'exécution, LOF est beaucoup plus rapide que OC-SVM. En général, les performances de toutes ces méthodes régressent (mais pas de la même manière) au fur et à mesure que la taille du jeu de données et ses dimensions augmentent. OC-SVM est adaptée aux jeux de données complexes de grandes dimensions, mais n'est pas adaptée aux jeux de données de grande taille. LOF est adaptée à la détection d'anomalies locales, car l'observation n'est comparée qu'à ses k plus proches voisins sans avoir de vision globale sur la totalité des données. Le mode de fonctionnement de IForest (pas de calcul de distance ni de densité) fait que le traitement est rapide même pour un jeu de données de grande taille ou de grande dimension. La contribution de différents arbres indépendants dans la prise de décision fournit également une très bonne précision dans la détection.

2.4 Discussion

La détection d'anomalies étant transversale à tout domaine de traitement de données, différentes méthodes sont proposées suivant les contraintes de chaque domaine d'application et type de données. Dans ce chapitre, nous avons fait une revue des méthodes existantes et celles adaptées à chaque domaine d'application et principaux types de jeux de données. Nous avons fourni une classification de ces méthodes suivant différentes approches.

L'étude comparative que nous avons menée a montré que IForest est plus performant que LOF et OC-SVM en termes de détection et de durée d'exécution sur deux jeux de données de différentes tailles et dimensions.

La détection d'anomalies dans les flux de données est confrontée aux contraintes des flux de données qui en constituent des défis. Les performances de IForest

en détection d'anomalies, sa complexité linéaire et sa rapidité en font une méthode bien adaptée au contexte des flux de données.Dans le prochain chapitre, nous menons une étude plus approfondie de IForest pour déterminer ses limites, l'impact de ses différents paramètres et pour évaluer ses extensions. D'autres problématiques comme l'adaptation de IForest aux flux de données ou aux contextes distribués sont abordés dans d'autres chapitres.

Chapitre 3

Etude approfondie de Isolation Forest

Sommaire

Nos travaux présentés dans ce chapitre ont été publiés dans le journal international IEEE Access :

Yousra Chabchoub, **Maurras Ulbricht TOGBE**, Aliou Boly, Raja Chiky. An in-depth study and improvement of Isolation Forest. IEEE Access, 2022, vol. 10, p. 10219-10237. https://doi.org/10.1109/ACCESS.2022.3144425

3.1 Introduction

Isolation Forest (IForest) est une méthode de détection d'anomalies basée sur un concept différent des approches présentées ci-dessus. Elle ne calcule aucune distance ou densité et réduit donc considérablement le temps d'exécution et les besoins en mémoire. IForest a été conçu par FT Liu et al. en 2008 (FT Liu et al., 2008 ; FT Liu et al., 2012). IForest est une méthode non supervisée qui donne d'excellents résultats en terme de précision avec une complexité temporelle linéaire et une faible consommation mémoire.

L'objectif de ce chapitre est d'étudier, en profondeur, IForest et d'évaluer l'impact de ses différents paramètres sur ses performances et l'efficacité de la détection. Nous nous concentrons particulièrement sur les anomalies non triviales, dans des jeux de données multidimensionnels, afin d'explorer les limites d'IForest. Nous avons comparé IForest à sa version étendue soit Extended Isolation Forest (EIF) (Hariri, Kind et Brunner, 2018a) qui améliore les faux positifs mais avec un temps d'exécution plus long. Dans ce chapitre, nous avons également proposé une nouvelle version que nous avons appelée Majority Voting Isolation Forest (MVIForest) offrant la même précision que IForest avec un temps d'exécution plus court. Pour comparer ces trois algorithmes, des expériences ont été réalisées sur des jeux de données synthétiques et réels, fréquemment utilisés dans le domaine de la détection d'anomalies, et présentant des caractéristiques différentes.

Ce chapitre est organisé comme suit : Dans une première section (3.2.1), nous présentons une étude de l'art sur la détection d'anomalies basée sur l'isolation. Dans une deuxième section (3.2.2), nous nous concentrons sur la méthode Extended Isolation Forest (EIF). La section 3.3 met en évidence une étude des performances de la méthode IForest. Dans la section 3.4, nous proposons et évaluons une amélioration de IForest appelée MVIForest. Une discussion des évaluations expérimentales des méthodes présentées est proposée dans la section

3.5.

3.2 Méthodes basées sur l'isolation

3.2.1 Etat de l'art des évolutions de IForest

La technique de détection d'anomalies par isolation a fait l'objet de plusieurs recherches. Depuis la première méthode IForest (FT Liu et al., 2008 ; FT Liu et al., 2012) conçue en 2008, de nombreuses adaptations et améliorations ont été proposées. Nous avons classé les méthodes proposées en deux catégories en fonction du type de données considérées : les données statiques et les flux de données.

3.2.1.1 Données statiques

Dans FT Liu et al. (2010), les auteurs ont proposé la méthode SCiForest qui, comme les méthodes basées sur la distance et la densité, permet d'identifier des clusters dans les données. SCiForest est une évolution de IForest pour le clustering afin de détecter les anomalies locales. SCiForest choisit aléatoirement un hyperplan afin de diviser les données en un nœud et ainsi prendre en compte les anomalies portées par plusieurs attributs en même temps. Pour séparer les différents clusters, SCiForest établit un critère de division pour chaque nœud en tenant compte de l'écart-type [1] des données pour ce nœud. Même si le traitement de SCiForest est adapté aux données complexes, SCiForest a une complexité élevée, ce qui représente un inconvénient majeur pour cette méthode.

Dans le même contexte, sur la manière de diviser les données dans le nœud, une version étendue de Isolation Forest (EIF) a été proposée dans (Hariri, Kind et Brunner, 2018a). EIF corrige le biais introduit dans IForest en raison de la division verticale ou horizontale (en dimension 2) des nœuds qui crée une incohérence dans les scores fournis par IForest. Cette méthode est discutée plus en détail dans la section 3.2.2.

1. Ecart-type : La dispersion des données d'un échantillon. Il s'agit d'une évaluation de l'étalement des valeurs autour de la moyenne.

Dans une étude plus récente, Staerman et al. ont proposé la méthode FIF (Functional IForest) (Staerman et al., 2019) pour détecter les anomalies dans les ensembles de données fonctionnelles (Ramsay, 2004). Partant de l'observation que la méthode IForest n'est pas efficace pour toute distribution de jeux de données, Marteau et al. ont proposé la méthode Hybrid IForest (HIF) (Marteau et al., 2017). Ils ont ajouté un nouveau critère de décision prenant en compte la similarité entre les données d'un même nœud feuille. Cette amélioration permet de minimiser les faux négatifs en repérant les données anormales situées dans un nœud feuille ayant un chemin relativement long. En effet, dans la version originale d'IForest, certaines anomalies peuvent passer inaperçues en raison de leur similarité : c'est l'effet de masquage (voir 1.1). En combinant des techniques supervisées et non supervisées, la méthode HIF est capable de détecter des anomalies dans divers ensembles de données présentant des distributions différentes. Cependant, il n'est pas toujours facile d'avoir des données labélisées pour une détection supervisée des anomalies.

Dans Cortes (2019), les auteurs ont utilisé IForest pour calculer la longueur du chemin de chaque donnée dans les arbres et ont défini cette métrique comme une nouvelle distance entre deux points. Comparée à d'autres méthodes de distance comme la distance euclidienne ou la distance de Manhattan, cette métrique est plus robuste. Elle offre également une adaptation d'IForest pour les données catégoriques et manquantes.

Simulated Annealing IForest (SA-IForest) (D Xu et al., 2017) est une amélioration de IForest qui consiste à créer une forêt plus stable et plus fiable. Pour ce faire, la méthode SA-IForest construit une première forêt à partir d'IForest. Ensuite, elle calcule la similarité entre les arbres de la forêt. La précision des arbres est également calculée par validation croisée. A la fin du processus, seuls les arbres à haute précision sont retenus. La validation nécessite l'étiquetage des données d'entraînement, ce qui n'est pas toujours évident et représente la limite de cette méthode semi-supervisée.

Pour pallier le problème d'inefficacité de l'IForest lorsque le nombre de dimensions des données augmente, la méthode Entropy IForest (E-IForest) a été proposée dans Liao et B Luo (2018). Cette méthode gère également le manque de robustesse de la méthode IForest en cas de données bruitées. Selon les auteurs, le

problème réside dans le choix de l'attribut de découpage des données au niveau de chaque nœud de l'arbre. Ils ont proposé une évolution d'IForest basée sur l'entropie. Lors du fractionnement des nœuds, E-IForest choisit un attribut en fonction de son entropie dans l'échantillon aléatoire. Les auteurs ont exploré différentes stratégies pour le choix du meilleur attribut.

Dans X Zhang et al. (2017), les auteurs ont remarqué qu'en procédant à des divisions successives selon une dimension aléatoire, IForest utilise implicitement la distance de Manhattan. IForest, dans sa version originale, est donc limité à cette mesure de distance. Les auteurs ont ensuite proposé un cadre générique appelé LSHIForest (Locality-Sensitive Hashing IForest) qui généralise IForest à d'autres mesures de distance et espaces de données.

D'autres travaux ayant présenté des améliorations de IForest peuvent être cités notamment concernant l'échantillonnage. En effet, l'échantillonnage à partir d'un ensemble de données déséquilibré où les données normales sont majoritaires n'est pas évident. Il existe plusieurs façons d'équilibrer l'ensemble de données avant l'échantillonnage. Ces méthodes peuvent être classées en deux catégories : sur-échantillonnage (ajout de données synthétiques dans classe minoritaire) ou sous-échantillonnage (suppression de données de la classe majoritaire pour un meilleur équilibre). Le sur-échantillonnage peut être réalisé à l'aide de la méthode appelée Synthetic Minority Over-Sampling TEchnique (SMOTE) (NV Chawla et al., 2002). Dans Zheng et al. (2019), les auteurs ont utilisé IForest pour identifier les catégories de données et les classer. Ensuite, à l'aide de la technique RWS (Roulette Wheel Selection), ils déterminent les plus proches voisins de ces données pour les classer. Puis, avec SMOTE, ils créent de nouvelles données de rembourrage basées sur les informations de voisinage. Ainsi, IForest-SMOTE (Zheng et al., 2019) permet d'équilibrer un jeu de données en créant de nouvelles données basées sur une présélection d'IForest et la méthode SMOTE. Dans la deuxième catégorie (sous-échantillonnage), la méthode IOS (Isolation forest Outlier detection and Subset selection) (WR Chen et al., 2016) est utilisée pour créer un échantillon représentatif. IOS s'appuie sur IForest pour détecter les anomalies. Certaines données normales sont retirées et l'échantillon est ainsi construit sur la base d'un ensemble de données équilibré.

Ni IForest ni ses différentes améliorations présentées ci-dessus ne sont adap-

tés au contexte des flux de données, où les données sont reçues en continu et sont volatiles.

3.2.1.2 Flux de données

Half-Space Trees (HSTrees) (Tan et al., 2011), Isolation Forest Algorithm for Streaming Data (IForest ASD) (Ding et Fei, 2013), et Performance Counter-Based iForest (PCB-iforest) (Heigl et al., 2021) sont des adaptations d'IForest au contexte des flux de données. IForest ASD utilise une technique de fenêtre glissante. IForest est exécutée pour détecter les anomalies en se basant sur un modèle préalablement créé avec les données des fenêtres précédentes. En cas de dérive (concept drift) [2], ce modèle est réinitialisé. IForest ASD est ainsi étroitement lié au jeu de données. Plus précisément, pour gérer le concept drift, IForest ASD maintient une valeur d'entrée (μ) qui est le taux d'anomalies de référence. Lorsque, dans une fenêtre donnée, le taux d'anomalies dépasse sensiblement μ, IForest ASD suppose qu'un changement se produit dans le comportement normal des données (le concept drift) et réinitialise donc le modèle avec les données de la fenêtre courante. Cette approche n'est pas très efficace car tout l'historique du comportement normal est perdu à chaque drift.

HSTrees est une évolution de IForest, conçue pour le streaming. Elle divise les nœuds en utilisant la moyenne des éléments des nœuds pour l'attribut sélectionné de manière aléatoire. Par conséquent, contrairement à IForest ASD, HSTrees gère automatiquement le concept drift (Joao Gama et al., 2004; João Gama, Žliobaitė et al., 2014) sans mettre à jour son modèle par une réinitialisation. En effet, HSTrees est plus rapide que IForest ASD et construit son modèle indépendamment du jeu de données considéré.

Randomized space trees (RS-Forest) (K Wu et al., 2014) est également une méthode de détection d'anomalies dans les flux de données basée sur le concept d'isolation. Elle est basée sur un estimateur de densité utilisé pour décider s'il est nécessaire de mettre à jour le modèle pour gérer la dérive. RS-Forest est basée sur l'hypothèse que les données anormales ont une faible densité, ce qui rejoint l'hypothèse précédemment discutée sur la rareté des données anormales, leur

2. Changement du comportement normal (voir 5.2.1)

différence entre elles et aussi par rapport aux données normales.

Les méthodes de détection des anomalies basées sur l'isolation ont été implémentées dans plusieurs frameworks. Le plus connu est scikit-learn [3] qui se concentre sur les données statiques. Une autre implémentation d'IForest est fournie par le framework H20 [4] qui est également très connu dans le domaine de l'apprentissage automatique. Certaines versions adaptées aux flux de données comme HSTrees (Tan et al., 2011) et IForest ASD Ding et Fei (2013) ont été implémentées dans le framework open source scikit-learn multiflow (Montiel et al., 2018) [5] qui est la version de scikit-learn consacrée aux flux de données. Un framework de détection d'anomalies basé sur le cloud est proposé dans Hariri et Kind (2018). Il utilise le cluster Spark [6] et certaines interfaces utilisateur, le tout géré par Kubernetes [7].

3.2.2 Extended Isolation Forest

3.2.2.1 Limite de IForest considérée

IForest est une méthode puissante, mais qui présente certains inconvénients. Une des limites d'IForest qui a été abordée dans la littérature est son incohérence dans la classification des données. En effet, IForest donne un score à chaque donnée. Selon la définition de l'anomalie : plus la donnée est différente des autres, plus son score est élevé. Ce qui n'est pas toujours le cas avec les scores attribués par IForest. Ainsi, il y a une certaine incohérence dans la conception des scores de IForest. Afin d'illustrer cette faiblesse, nous avons considéré dans la figure 3.1 un jeu de données bi-dimensionnel uniformément réparti sur un anneau. Les anomalies sont placées au centre de l'anneau. Cette distribution est invariante par rotation. Après avoir exécuté IForest, nous avons représenté la carte (heat map) qui correspond aux scores attribués par IForest. Les zones avec une forte densité de points ont un score faible et sont donc considérées comme des zones de données normales. La couleur claire homogène au centre de la

3. https://scikit-learn.org/stable/
4. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/if.html
5. https://scikit-multiflow.readthedocs.io/en/stable/api/api.html
6. https://spark.apache.org/
7. https://kubernetes.io/fr/docs/reference/

heat map donne l'impression que les données sont distribuées sur un disque et ignore la forme d'anneau avec un centre presque vide. Loin des zones denses, les scores deviennent plus élevés. Nous pouvons clairement voir l'effet de bord sur les zones frontalières, ainsi que les zones plus sombres qui correspondent aux zones vides. Cependant, la heat map n'a pas la même symétrie que les données. La heat map n'est pas invariante par rotation. Plus précisément, l'anneau s'est transformé en un carré, et nous pouvons voir l'apparition artificielle de quatre coins sombres avec des scores très élevés. Ces zones sont considérées comme des anomalies.

Cette situation est due à la façon dont IForest divise les nœuds pendant la phase d'entraînement. En effet, IForest divise chaque nœud verticalement ou horizontalement. Cela crée des zones de concentration artificielles comme si d'autres données s'y trouvaient. La figure 3.1 montre un exemple de division des nœuds lors de la construction d'un arbre de la forêt. Notons que les différentes divisions de l'ensemble de données forment des zones rectangulaires artificielles de haute densité en dehors de l'anneau de l'ensemble de données.

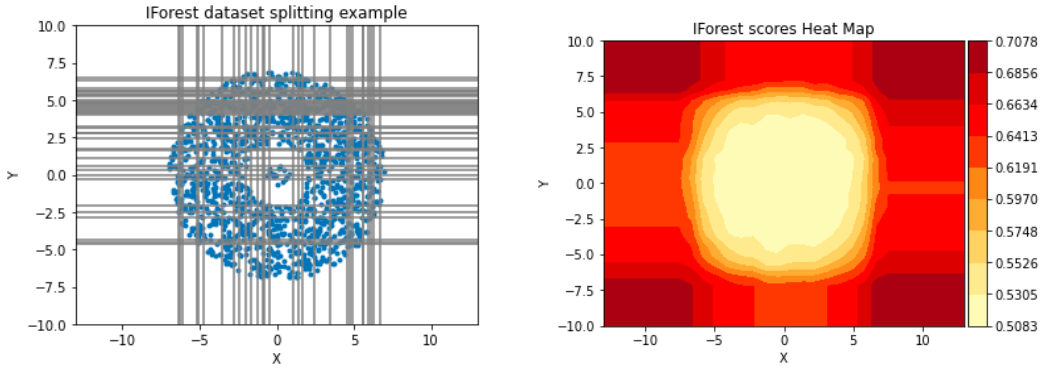


FIGURE 3.1 – Exemple de scissions des noeuds par IForest avec sa heat map basée sur les scores associés : présence de zones sombres artificielles. X et Y sont les coordonnées des données.

Une évolution d'IForest qui surmonte ce problème est EIF. L'idée principale d'EIF est de diviser les données du nœud selon une direction choisie au hasard (pas nécessairement horizontale ou verticale comme IForest). Cela élimine les zones fictives créées par IForest et améliore par conséquent la cohérence des

scores. La méthode appelée Extended Isolation Forest qui vise à corriger cette incohérence a été proposée dans Hariri, Kind et Brunner (2018a). La principale différence entre EIF et IForest est la manière de diviser les données dans un nœud.

3.2.2.2 La méthode EIF

Comme expliqué dans la section 2.2.7, dans le cas de données bi-dimensionnelles, IForest fractionne chaque nœud horizontalement ou verticalement (voir figure 2.4). Ce découpage crée un biais clairement visible sur la heat map des scores. EIF contient deux étapes, tout comme IForest : la phase d'entraînement et la phase de test où le score est calculé (phase de scoring). Si la phase de scoring reste la même pour les deux méthodes, la phase d'entraînement a été modifiée de manière significative. En effet, EIF divise les nœuds selon un point et une direction choisis aléatoirement par combinaison de toutes les dimensions contrairement à IForest.

Un exemple de scission des données à l'aide d'EIF est donné à la figure 3.2 où l'on peut remarquer la scission oblique et aléatoire des nœuds. Une heat map des scores obtenue pour le même jeu de données est présentée sur cette même figure. Tout comme pour IForest, dans cette expérience, nous avons utilisé toutes les données (normales et anormales) pour construire les échantillons d'arbres, pendant la phase d'entraînement. Nous pouvons remarquer la correspondance parfaite entre la distribution des données et les couleurs de la heat map. La forme circulaire est conservée par la heat map des scores. Les zones de concentration fictives, précédemment visibles avec IForest, ont disparu. De plus, la couleur du centre de l'anneau (relativement sombre) reflète clairement la forme de l'anneau et les données au centre, et apporte plus de précision qu'IForest. Ainsi, cette heat map est conforme à la distribution réelle du jeu de données initial.

Avec des données de plus grande dimension, EIF conserve le même principe de fonctionnement. Les nœuds sont répartis selon un hyperplan aléatoire défini par un vecteur normal $\vec{n}$ et un point $\vec{p}$ appelé point d'interception. Les données des nœuds sont distribuées comme suit : pour toute donnée $\vec{x}$, si $(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0$ alors $\vec{x}$ est classée dans le noeud fils de gauche sinon il est affectée au noeud fils

de droite.

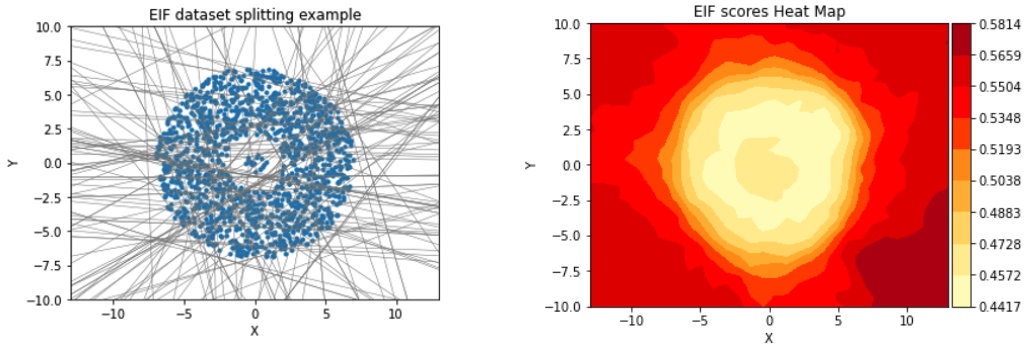


FIGURE 3.2 – Exemple de scissions des noeuds par EIF avec sa heat map basée sur les scores : disparition des zones sombres artificielles. X et Y sont les coordonnées des données.

Bien que EIF remédie à l'incohérence d'IForest pour le score, cette méthode présente certaines limites qu'il convient de prendre en compte. La méthode EIF réduit considérablement le taux de fausses alertes d'IForest. Cependant, en termes de détection d'anomalies, elle n'est pas toujours plus performante qu'IForest. Une comparaison des deux méthodes est effectuée dans la section 3.5.

3.3 Etude de IForest

IForest est l'une des méthodes les plus récentes et les plus efficaces pour détecter des anomalies selon plusieurs critères. Cependant, comme toutes les méthodes existantes, IForest présente des limites qui en font une méthode perfectible. Dans cette section, nous présenterons les limites d'IForest que nous avons identifiées et qui n'ont pas été abordées dans la littérature 3.3.2. Ensuite, nous évaluerons l'impact des paramètres d'IForest sur ses performances 3.3.4, et nous nous focaliserons sur l'effet de la présence ou non de données anormales dans les échantillons 3.3.5. Nous présenterons également l'impact de l'augmentation des dimensions des données sur IForest 3.3.6 puis nous terminerons cette section par une analyse du choix du meilleur seuil de décision 3.3.7. Sauf indication

contraire, nous avons utilisé dans toutes les expériences les valeurs par défaut recommandées par les auteurs d'IForest, à savoir $t = 100$ et $\psi = 256$. Le seuil de décision utilisé est $threshold = 0.6$.

Les métriques considérées dans cette étude ont été décrites à la section 2.3.2. Nous avons choisi d'évaluer les performances d'IForest au cours de notre étude en testant cette méthode sur différents jeux de données publics réels décrits à la section 2.3.1 et des jeux de données synthétiques. Le code source de toutes nos expériences, celui de la génération de jeux de données synthétiques et celui de MVIForest sont disponibles dans le dépôt git public : `https://github.com/Elmecio/IForestStudyAndMVIForest`.

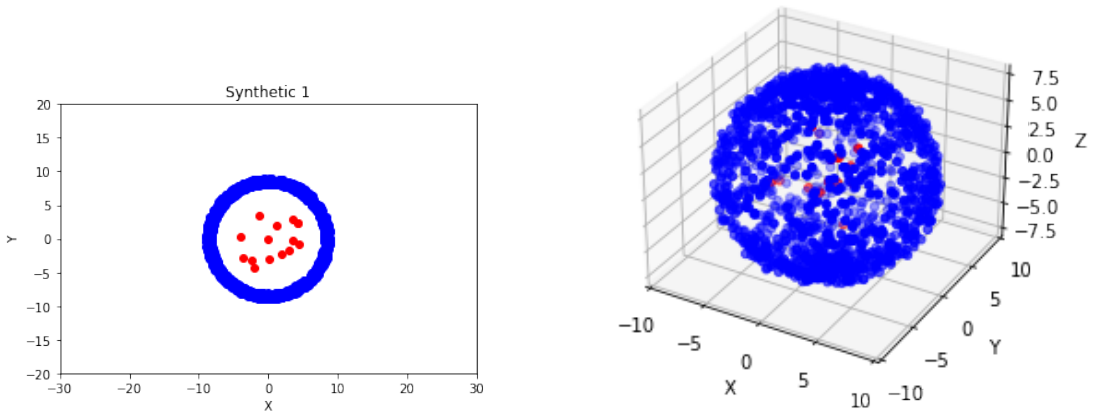
3.3.1 Jeux de données synthétiques

Afin de tester le comportement et la sensibilité d'IForest à différentes variations des caractéristiques des données, nous avons utilisé des jeux de données synthétiques. En effet, les jeux de données réels contiennent souvent des anomalies portées par un seul attribut et ne permettent pas une analyse détaillée de l'impact de certains paramètres (dimensions des données par exemple). Nous avons donc fait varier, sur les jeux de données synthétiques, les dimensions des données, la densité des données normales et anormales ainsi que la distance entre les anomalies et les données normales. Ces configurations permettent d'évaluer les limites d'IForest et d'ajuster le choix de ses paramètres d'entrée. Dans le cas de données multivariées, nous avons conçu des anomalies portées par toutes les dimensions à la fois, bien enveloppées dans les données normales.

Les jeux de données synthétiques conçus sont présentés, en deux dimensions, sous la forme d'un anneau où les données normales sont uniformément distribuées. Les données anormales sont situées au centre de l'anneau, uniformément réparties sur un disque de plus petit rayon. En trois dimensions, les données normales sont contenues dans l'enveloppe épaisse d'une sphère et les anomalies sont regroupées au centre de la sphère, uniformément réparties dans une sphère plus petite. Les données anormales sont donc à chaque fois entourées de données normales. Les anomalies ne sont pas triviales, ce qui permet de tester les limites des algorithmes de détection. Les différents jeux de données utilisés sont

présentés dans les figures 3.3 et 3.4.

Les données anormales sont en rouge et les données normales en bleu. Tous les jeux de données sont composés de 1500 données normales et de 15 (1%) données anormales. Les coordonnées de chaque donnée sont choisies aléatoirement dans la distribution décrite ci-dessus. Nous avons fait varier l'épaisseur de l'anneau et le rayon de la sphère centrale pour tester différentes densités et distances entre les données normales et les anomalies, comme indiqué dans le tableau 3.1.

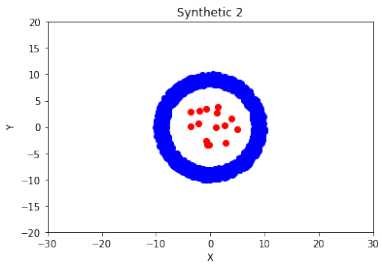


(a) 2D (b) 3D

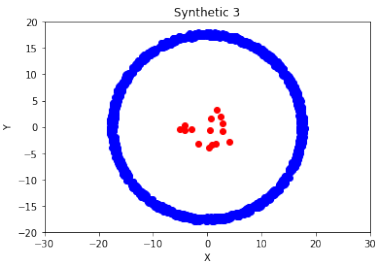
FIGURE 3.3 – Dataset Synthetic_1 : Forte densité des données normales (ND++), faible densité des données anormales (AD+) et petite distance entre les données normales et anormales (dAN+).

Le jeu de données appelé Synthetic_1 est représenté sur la figure 3.3. Il est caractérisé par une forte densité de données normales et une faible distance entre les données normales et les anomalies. Le jeu de données appelé Synthetic_2 est illustré sur la figure 3.4a où nous avons simplement augmenté l'épaisseur de l'anneau pour évaluer l'impact de la diminution de la densité des données sur les performances d'IForest. La distance entre les données normales et anormales reste assez faible. Le jeu de données Synthetic_3 (figure 3.4b) est caractérisé par une forte densité de données normales et une grande distance entre les données normales et anormales. Dans l'ensemble de données Synthétique_4 (figure

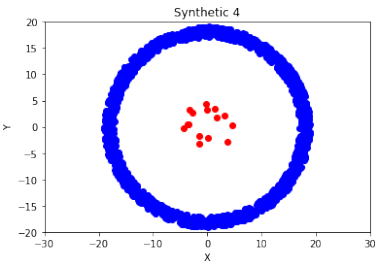
3.4c), nous avons conservé la grande distance entre les données normales et anormales, cette fois-ci, avec une faible densité de données normales. L'objectif du jeu de données Synthetic_5 (figure 3.4d), est de tester l'effet du masquage discuté précédemment. Nous avons donc conçu des anomalies avec une forte similarité, en réduisant significativement le rayon de leur disque. Les anomalies sont plus concentrées au centre de l'anneau. Tous les jeux de données synthétiques ont 1500 données normales et 15 données anormales. Nous résumons les caractéristiques de ces jeux de données synthétiques dans le tableau 3.1. Pour chacun de ces jeux de données, nous avons également créé l'ensemble de données tridimensionnel correspondant, comme expliqué pour le jeu de données Synthétique_1 dans la figure 3.3.



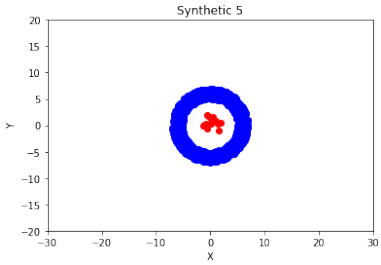
(a) ND+, AD+, dAN+



(b) ND++, AD+, dAN++



(c) ND+, AD+, dAN+++



(d) ND+, AD++, dAN+

FIGURE 3.4 – Jeux de données synthétiques à 2 dimensions.

3.3.2 Limites de IForest

Certaines limites d'IForest ont été identifiées et traitées dans la littérature, en proposant des évolutions et extensions d'IForest (EIF (HARIRI, KIND et BRUNNER,

Tableau 3.1 – Caractéristiques des jeux de données synthétiques. + : Faible, ++ : Forte, ND : Densité des données normales, AD : Densité des données anormales, dAN : distance entre données anormales et normales.

	ND	AD	dAN
Synthetic_1	++	+	+
Synthetic_2	+	+	+
Synthetic_3	++	+	++
Synthetic_4	+	+	++
Synthetic_5	+	++	+

2018a), IForest ASD (Ding et Fei, 2013), ...). Dans cette section, nous explorons d'autres limites d'IForest dont la résolution permettrait d'améliorer la robustesse de la méthode et d'améliorer sa position dans le classement des méthodes de détection d'anomalies.

3.3.2.1 Le seuil de décision

IForest est basé sur un score, noté $s(x)$, qui est calculé pour chaque donnée x sur la base de la longueur moyenne de sa trajectoire $E(h(x))$ dans les t arbres. Dans l'article original d'IForest (FT Liu et al., 2008), les auteurs recommandent 0.5 comme seuil de décision appliqué au score s. En effet,

— si $E(h(x)) \to 0$ alors $s \to 1$. Quand s est très proche de 1, x peut être considérée comme une anomalie ;

— si $E(h(x)) \to n-1$ alors $s \to 0$. Quand $s << 0.5$, x peut être considérée comme une donnée normale.

On note clairement un manque de précision du seuil à partir duquel une donnée doit être considérée comme anormale. Cette faiblesse constitue une première limite d'IForest.

3.3.2.2 Choix des paramètres de IForest

IForest possède deux paramètres d'entrée : la taille de l'échantillon (ψ) et le nombre d'échantillons (t) qui correspond également au nombre d'arbres. Dans FT Liu et al. (2008) et FT Liu et al. (2012), la recommandation des auteurs est de fixer ψ à 256. Selon leur expérience, $\psi = 256$ donne de bons résultats avec un faible

temps d'exécution et un faible besoin en mémoire. Les auteurs recommandent de fixer t à 100 arbres pour obtenir des résultats stables. Ils démontrent qu'avec 100 arbres, le résultat est optimal et qu'au-delà de 100 arbres, il n'y a pas de gain de performance mais le temps d'exécution et la consommation mémoire augmentent logiquement.

Les expériences montrent que la performance d'IForest au niveau de la détection (spécificité, rappel, AUC) dépend de ces paramètres. Le choix optimal de ces paramètres dépend des caractéristiques du jeu de données considéré. En effet, la taille de l'échantillon a un effet considérable sur la performance d'IForest. D'une part, une grande taille d'échantillon (ou un taux d'échantillonnage élevé) peut engendrer un sur-entraînement. En effet, on obtient dans ce cas une grande similarité entre les arbres, ce qui est en contradiction avec la forêt d'arbres aléatoires et indépendants. A l'inverse, un taux d'échantillonnage très faible engendre un risque de sous-apprentissage (un échantillon pas assez représentatif de l'ensemble du jeu de données et des arbres pas assez profonds pour isoler les anomalies). Il est donc important d'ajuster la taille de l'échantillon, d'autant plus qu'elle représente un paramètre d'entrée clé pour la méthode IForest. Nous traitons cette limite dans la sous-section 3.3.4.

3.3.2.3 Les dimensions des données

Dans FT Liu et al. (2008), les auteurs affirment que IForest fournit de bons résultats même sur des données à grandes dimensions car plusieurs de ces dimensions sont non significatives. Ils proposent également d'appliquer des tests statistiques comme le Kurtosis pour sélectionner les dimensions pertinentes et ainsi réduire les dimensions. Le nombre de dimensions m n'est pas pris en compte dans le choix des paramètres d'entrée dans IForest. Les anomalies qui sont portées par plusieurs dimensions à la fois sont plus difficiles à détecter. La figure 3.3 montre deux types de données où les anomalies sont respectivement portées par 2 dimensions (a) et 3 dimensions (b).

Dans les deux cas, avec les paramètres par défaut d'IForest, nous obtenons un taux de faux positifs élevé (de l'ordre de 15%). De plus, les anomalies ne sont détectées que si elles sont suffisamment éloignées des données normales

pour 2 dimensions. En 3 dimensions, aucune anomalie n'est détectée. Les performances d'IForest diminuent lorsque le nombre de dimensions pertinentes augmente. En effet, lors de la construction des arbres, à chaque étape, nous choisissons aléatoirement une dimension parmi les m dimensions alors que la profondeur maximale de l'arbre, max_depth donnée par $\lceil log_2(\psi) \rceil$ est indépendante de m. Pour une grande valeur de m, certaines dimensions peuvent ne pas être sélectionnées. Nous traitons de cette limite dans la sous-section 3.3.6.

3.3.2.4 La décision de classification

Une donnée x est déclarée comme une anomalie selon une décision collective produite par des arbres. Les t itrees [8] sont utilisés pour calculer la longueur moyenne du chemin de x, Dans la phase de test (scoring), le temps d'exécution d'IForest dépend de la taille des données, mais aussi du nombre d'arbres de la forêt. Ceci est dû au fait que chaque donnée doit être traitée par chaque arbre et construire son chemin avant de prendre une décision. Ce temps d'exécution peut être considérablement réduit en choisissant une autre approche. Nous avons proposé d'appliquer la méthode de vote majoritaire pour déclarer une donnée comme une anomalie. Cela nous a permis de réduire le taux de fausses alarmes et le temps d'exécution. Cette méthode est présentée dans 3.4.

3.3.3 L'effet de l'aléatoire dans IForest

IForest est une méthode qui se base sur l'aléa à plusieurs étapes. En effet, le choix de l'échantillon est aléatoire pour chaque arbre de la forêt. Le choix de la dimension adressée ainsi que de la valeur de scission sont aléatoires pour chaque nœud. Il est intéressant de vérifier l'impact de ce caractère aléatoire sur les résultats d'IForest en inspectant la variance des résultats sur plusieurs exécutions d'IForest. Nous avons réalisé cette expérience en exécutant IForest dix (10) fois avec le même seuil de décision c'est-à-dire $threshold = 0.5$ sur le jeu de données shuttle décrit dans la sous-section 2.3.1.

8. itree : arbre aléatoire, binaire et indépendant construit selon les principes de Isolation Forest.

Tableau 3.2 – Statistiques de 10 exécutions de IForest sur le jeu de données Shuttle avec les mêmes paramètres : $\psi = 100$, $t = 256$, $threshold = 0.5$. μ est la moyenne et σ est l'écart type.

Metrics	CPU Time	ROC AUC	Specificity	Recall
Values	[2.25, 2.62]	[0.997, 0.998]	[0.93, 0.94]	[0.98, 0.99]
μ	2.448	0.998	0.939	0.986
σ	0,114	0,001	0,006	0,003

Le tableau 3.2 montre que les dix (10) exécutions successives d'IForest donnent des résultats assez constants en termes des valeurs des métriques ROC AUC, spécificité et rappel. Les écarts types de ces trois métriques sont très proches de zéro. Ces résultats illustrent la stabilité d'IForest malgré son caractère aléatoire. Nous pouvons donc nous fier à une seule exécution d'IForest.

3.3.4 Impact des paramètres de IForest

Les paramètres d'entrée d'IForest : la taille de l'échantillon (ψ) et le nombre d'arbres (t) sont d'une grande importance pour les performances d'IForest. Nous avons réalisé quelques expériences pour évaluer l'impact de ces paramètres sur les résultats. Ces paramètres doivent être bien choisis par l'utilisateur pour optimiser les résultats. En particulier, lors de l'application d'IForest à un flux de données avec un concept drift, les paramètres d'entrée devraient idéalement s'adapter automatiquement aux caractéristiques variables du flux. Nous étudions ici l'impact de ces paramètres sur l'efficacité d'IForest.

3.3.4.1 Impact de la taille de l'échantillon

Le temps d'exécution d'IForest est lié au nombre d'échantillons ou d'arbres t et à la taille de chaque échantillon ψ. Dans cette partie, nous nous intéressons à l'impact de ce paramètre ψ. Nous avons fixé t à sa valeur par défaut (100 arbres) dans les expériences.

La figure 3.5 montre les résultats de l'exécution d'IForest sur le jeu de données Shuttle avec différentes valeurs de taille d'échantillon. Notons que lorsque

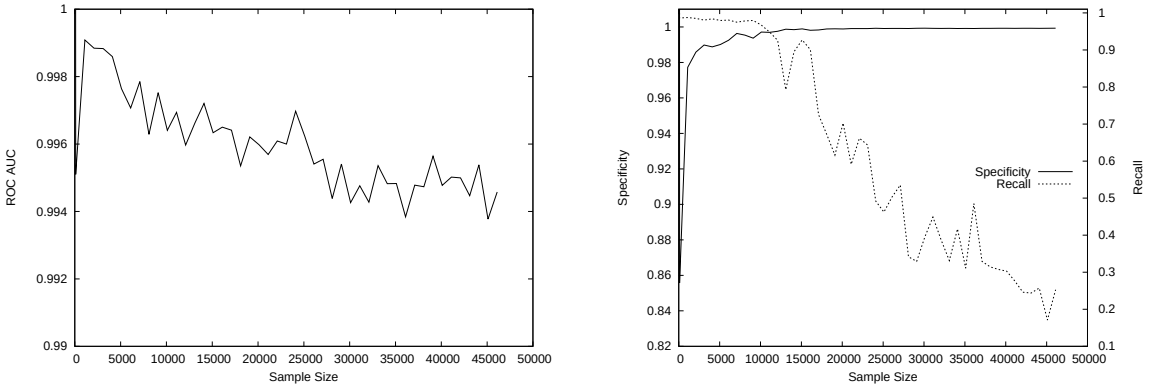


FIGURE 3.5 – Performances de IForest avec différentes tailles d'échantillon (ψ).

la taille de l'échantillon augmente, le temps d'exécution est plus long. Cela peut s'expliquer par le fait que la longueur maximale des arbres dépend de la taille de l'échantillon. En outre, chaque arbre est construit à partir d'un échantillon, de sorte que la phase d'apprentissage de la méthode IForest dépend étroitement de la taille de l'arbre. Une grande taille d'arbre ne signifie pas nécessairement un gain de performance. Ceci est mis en évidence dans les autres métriques (specificity et recall) de la figure 3.5. A partir d'un seuil donné, plus la taille de l'échantillon augmente, moins IForest est capable de détecter les anomalies (rappel décroissant). En revanche, les données normales sont bien classées (la spécificité atteint rapidement la valeur 1). Pour un très grand échantillon, IForest a tendance à classer toutes les données comme normales. Une raison possible serait la mauvaise qualité des arbres construits : plus la taille de l'échantillon est grande, plus les arbres de la forêt se ressemblent et deviennent redondants, ce qui affecte leur décision collective. En résumé, avec IForest, la taille de l'échantillon est un paramètre important qui a une grande influence sur les performances de la détection. Une bonne taille d'échantillon doit donc être suffisamment petite pour une meilleure performance d'IForest, tout en veillant à garder l'échantillon le plus représentatif possible des données initiales. La taille optimale de l'échantillon dépend donc de la nature et de la variance du jeu de données adressé. Dans la figure 3.6, l'étude a été réalisée sur les quatre (4) jeux de données décrits

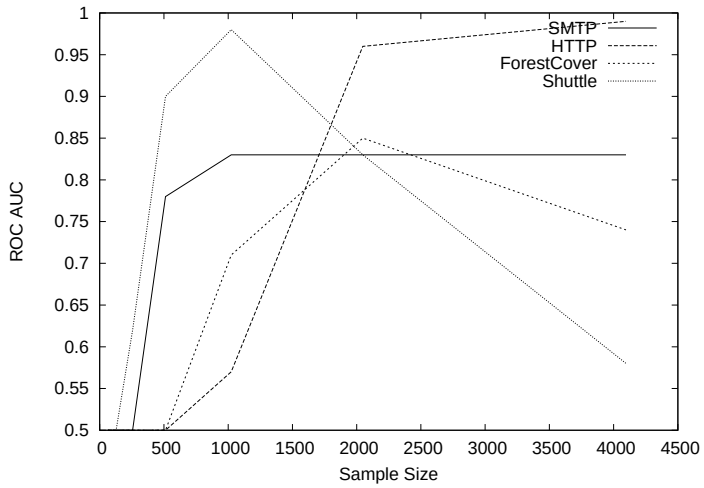


FIGURE 3.6 – Impacts de la taille de l'échantillon sur les résultats de IForest. Évaluation sur quatre (4) jeux de données réels.

dans 2.3.1, dans le but de trouver la taille idéale de l'échantillon dans chaque cas. Notons que pour ces différents jeux de données, la taille de l'échantillon qui donne les meilleurs résultats diffère. En effet, pour Shuttle et SMTP, la meilleure valeur de ROC AUC a été obtenue avec $\psi = 1024$. Alors que pour Forest Cover, $\psi = 2048$ était la taille d'échantillon optimale, et $\psi = 4096$ pour HTTP. Contrairement à SMTP et HTTP où la valeur de ROC AUC est restée constante après la taille d'échantillon optimale, IForest est moins performant pour Forest Cover et Shuttle au-delà de la taille d'échantillon correcte. Ces résultats confirment que la taille optimale de l'échantillon pour une bonne classification des données (normales et anormales) dépend des caractéristiques de l'ensemble de données. La capacité d'IForest à détecter les anomalies dépend de la longueur maximale des arbres construits dans la forêt. Or, la longueur de l'arbre dépend de la taille de l'échantillon. Il est donc important de fournir la taille d'échantillon adaptée en entrée pour une bonne performance d'IForest.

3.3.4.2 Impact du nombre d'arbres

IForest est basé sur une forêt aléatoire composée de plusieurs arbres binaires aléatoires et indépendants. Leur indépendance vient du fait que chaque arbre est construit sur la base d'un seul échantillon aléatoire de même taille. Tous les arbres participent de manière égale à la prise de décision concernant la classification d'une donnée : normale ou anormale. Le nombre d'arbres à construire dans la forêt est un paramètre d'entrée de la méthode. Les besoins en mémoire sont étroitement liés au nombre d'arbres de la forêt. Dans cette sous-section, nous avons mené des expériences pour évaluer l'impact du nombre d'arbres sur les performances d'IForest. Pour cela, nous avons utilisé différentes valeurs pour ce paramètre t. Nous avons fixé la taille de l'échantillon à sa valeur par défaut : $\psi = 256$.

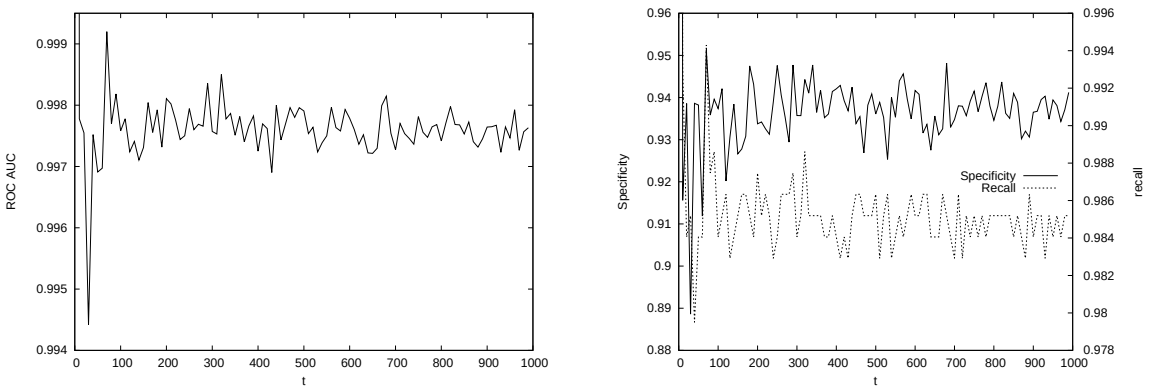


FIGURE 3.7 – Performances de IForest pour différents nombres d'arbres (t).

Comme prévu, lorsque le nombre d'arbres augmente, le temps d'exécution devient plus long, ce qui correspond au fait que tous les arbres sont créés pendant la phase d'apprentissage et aussi que pendant la phase de test, la donnée doit passer par chaque arbre de la forêt pour la prise de décision. Comme on peut le voir sur la figure 3.7, en considérant les métriques ROC AUC, spécificité et rappel, on remarque que le nombre d'arbres n'a pas un grand impact sur ces résultats. En effet, à partir d'un certain nombre d'arbres, les valeurs

convergent alors que le temps d'exécution continue d'augmenter linéairement. Nous pouvons donc conclure que, au-delà d'un seuil donné, le nombre d'arbres n'a pas un grand impact sur les performances d'IForest en termes de détection d'anomalies. La recommandation des auteurs de générer une décision collective basée sur la collaboration de $t = 100$ arbres semble être un bon compromis entre le temps d'exécution et la qualité de la décision. Nous voyons également sur cette même figure 3.7 que la métrique ROC AUC converge rapidement vers sa valeur maximale (presque égale à 1), ce qui confirme l'efficacité de la méthode IForest.

3.3.5 Impact de la présence d'anomalies dans les échantillons

Lors de l'apprentissage, comme le choix des données dans l'échantillon est aléatoire, les échantillons peuvent ne pas contenir de données anormales. Nous évaluons dans cette section l'impact de la présence de données anormales dans les échantillons pendant la phase d'apprentissage. Deux expériences ont été réalisées à cet effet. À partir du jeu de données Synthetic_3 (3.4b), nous avons d'abord effectué un choix aléatoire dans l'ensemble du jeu de données (données normales et anormales) noté "AllDataset" puis appliqué IForest. Puis dans la deuxième expérience, l'échantillon aléatoire est sélectionné uniquement sur les données normales notées "NormalOnly". Le résultat est enregistré dans le tableau 3.3. IForest est plus efficace lorsque la construction de l'arbre est effectuée sur la base d'un mélange de données normales et anormales. Lorsque nous construisons la forêt avec uniquement des données normales, nous obtenons une valeur de ROC AUC de $0,95$ avec 9% de fausses alarmes qui reflètent le fait que certaines données normales ont été très tôt isolées dans l'arbre car elles sont légèrement différentes des autres. Ce nombre est réduit en cas de présence de données anormales dans l'échantillon. Le pourcentage de fausses alarmes est réduit à 8% et la valeur du ROC AUC augmente à 0.96. La présence de données anormales dans l'échantillon augmente donc les performances d'IForest et réduit le taux de fausses alarmes.

Plus précisément, cela peut s'expliquer par le fait qu'en présence de données anormales dans l'échantillon, l'intervalle de variation d'une dimension donnée

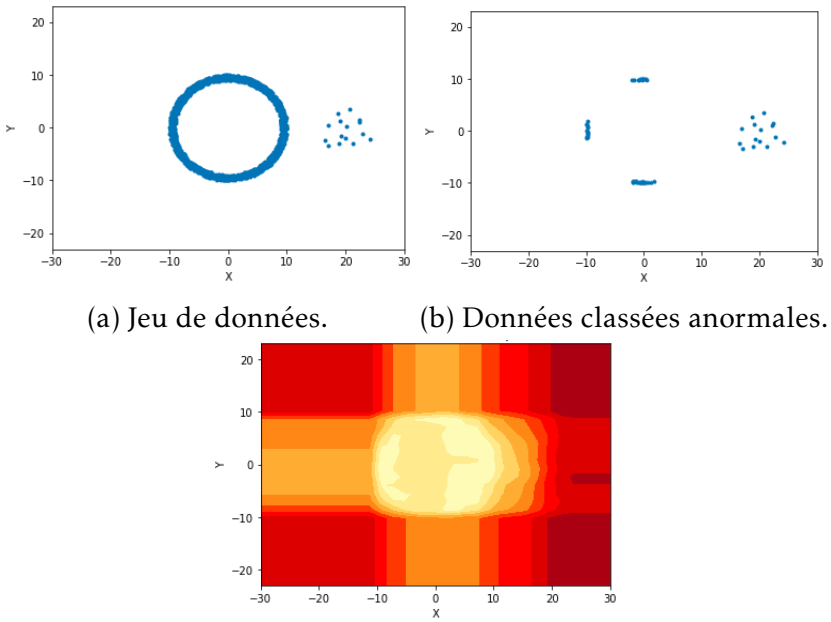
au niveau du nœud considéré est généralement plus large. Dans ce cas, le choix aléatoire de la valeur de scission a plus de chance de se rapprocher des valeurs des données anormales permettant ainsi de les isoler plus rapidement. Nous avons ensuite choisi de considérer une autre distribution de données (présentée dans la figure 3.8) et d'évaluer l'impact de la présence de données anormales dans l'échantillon dans une telle configuration. Dans ce cas, les anomalies sont bien séparées des données normales, car elles sont à l'extérieur de l'anneau. Nous avons gardé la même proportion pour les anomalies (1% du nombre total de données). En construisant l'échantillon à partir de toutes les données (normales et anormales), les données anormales sont parfois présentes dans l'échantillon et ont un impact significatif sur la heat map des scores, montrée dans cette même figure. On note une déformation à droite de la heat map avec une dégradation plus importante des couleurs. Il s'agit d'une zone d'incertitude qui peut créer à la fois des faux positifs et des faux négatifs pour les données. Ainsi, la présence de données anormales dans l'échantillon ne semble pas utile dans une telle configuration.

Tableau 3.3 – Comparaison des performances de IForest sur un jeu de données avec et sans anomalies dans la phase d'apprentissage. Jeu de données utilisé : Synthetic_3 (3.4b).

	AllDataset	*NormalOnly*
ROC AUC	**0.96**	0.95
Recall	1	1
Specificity	**0.92**	0.91
False Alarm Rate (%)	**8**	9
F1 score	**0.20**	0.18

3.3.6 Impact du nombre de dimensions

Les données sont souvent multivariées, décrivant la variation d'au moins deux observables au cours du temps. Afin d'évaluer les performances d'IForest en fonction du nombre de dimensions du jeu de données, nous avons utilisé les jeux de données synthétiques à deux et trois dimensions présentés dans la section 3.3.1. La particularité de ces jeux de données est que les anomalies sont



(a) Jeu de données.

(b) Données classées anormales.

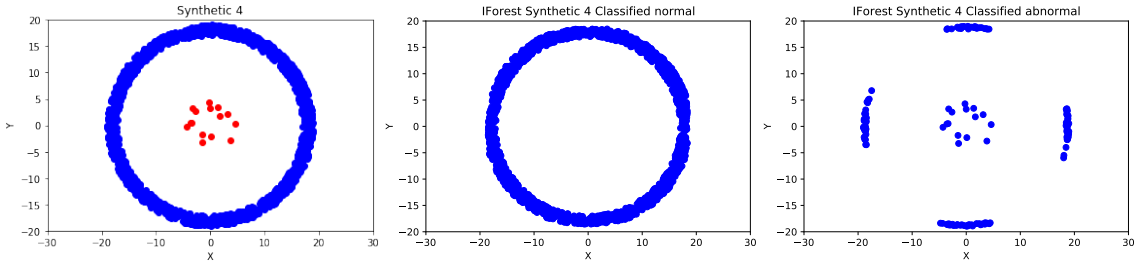
(c) Heat Map basée sur les scores.

FIGURE 3.8 – Impacts de la distribution des données sur IForest.

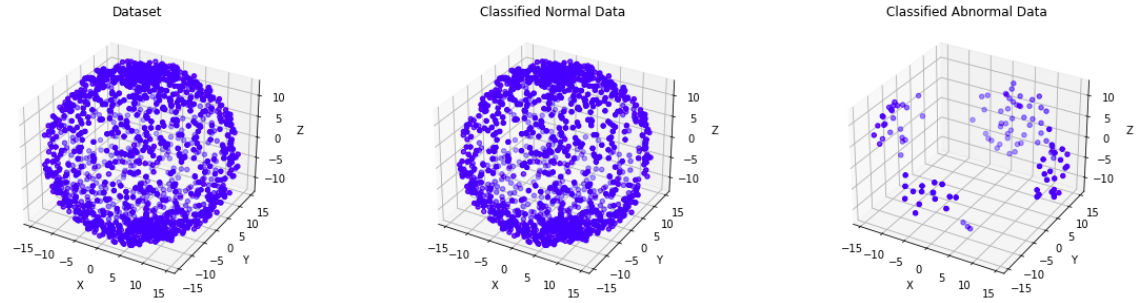
portées par plusieurs dimensions à la fois. Il sera donc nécessaire de considérer plusieurs dimensions pour détecter les anomalies. Lorsque les anomalies sont portées par plusieurs dimensions à la fois (Figure 3.9), IForest est moins efficace. Les anomalies ne sont détectées dans ces derniers cas que lorsque la différence de densité et de distance entre les données normales et les anomalies est très nette. IForest fait toujours beaucoup de fausses alertes, surtout aux frontières des données normales. Plusieurs anomalies réelles n'ont pas été détectées en raison du processus de division des nœuds dans la phase d'apprentissage d'IForest. En effet, IForest effectue des scissions en choisissant un attribut de manière aléatoire. Le fractionnement se fait donc avec un seul attribut à la fois. Cependant, en ne considérant qu'une seule dimension, les données anormales semblent être normales car elles sont dans la même plage de valeurs que les données normales. Nous remarquons également qu'avec un nombre élevé de dimensions, le taux de faux positifs augmente.

Le nombre de dimensions est donc un paramètre très important, à prendre

en compte pour améliorer l'évolutivité d'IForest et l'adapter aux données de grande dimension. Une évolution possible d'IForest serait de choisir la taille de l'échantillon ψ en fonction de la taille des données et aussi de leur dimension. En effet, afin de détecter une anomalie portée par m dimensions, il est évident qu'un arbre de profondeur minimale m doit être utilisé. Cependant, la profondeur de l'arbre est bornée par $depth\_max = \lceil log_2(\psi) \rceil$. Cette formule doit être modifiée pour inclure m soit directement, soit en liant m à ψ.



(a) 2 dimensions



(b) 3 dimensions

FIGURE 3.9 – Résultats de l'exécution de IForest sur des jeux de données de 2 et 3 dimensions.

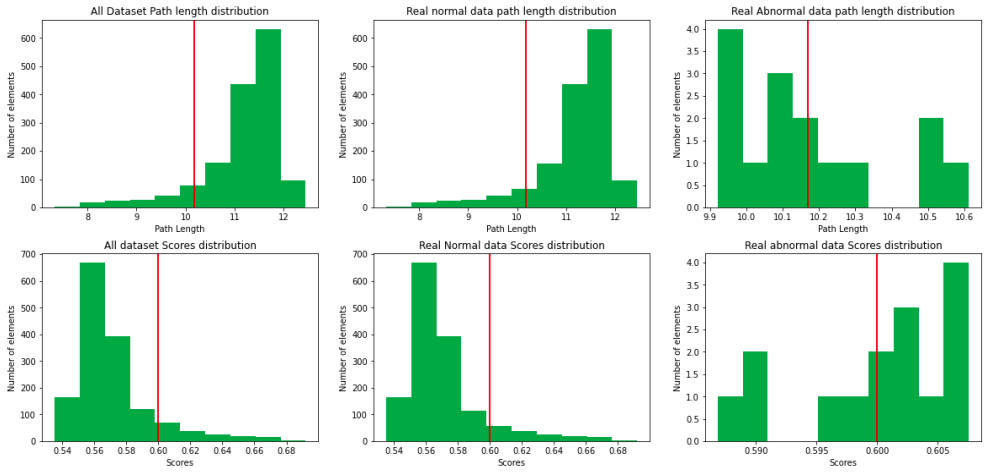
3.3.7 Choix de la décision de classification

Les méthodes de détection des anomalies sont généralement basées sur un score généré pour chaque élément de données et comparé à un seuil de détection. En utilisant ce score, ces méthodes classent les différents éléments en séparant les données anormales et normales.

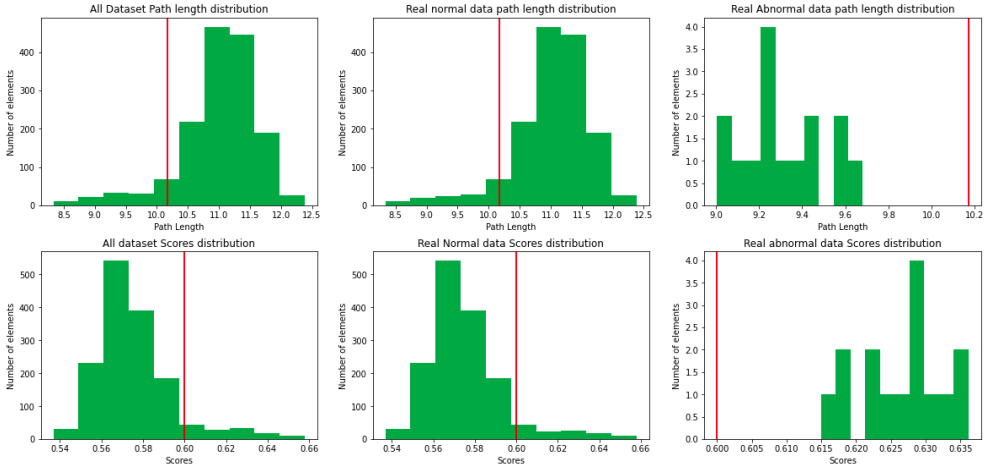
Dans l'article original d'IForest, les auteurs ont suggéré d'utiliser 0.5 comme seuil pour la prise de décision en calculant le score $s(x,n) = 2^{-\frac{E(h(x))}{C(n)}}$, comme suit :

— lorsque s (x, n) est très proche de 1 alors x est une anomalie ;

— lorsque s (x, n) est inférieur à 0,5 alors x est une donnée normale.



(a) Synthetic_2 dataset



(b) Synthetic_3 dataset

FIGURE 3.10 – Distribution des scores et des longueurs de chemins de IForest sur les jeux de données Synthetic_2 et Synthetic_3.

En pratique, la décision d'anomalie est prise lorsque le score est supérieur à 0.5. Mais, cette règle peut engendrer des fausses alarmes ou des faux négatifs, car le seuil de décision optimal n'est pas toujours égal à 0.5. Afin d'illustrer

cette observation, nous avons considéré les deux jeux de données synthétiques Synthetic_2 et Synthetic_3 et nous avons représenté, dans la figure 3.10, la distribution des scores d'abord pour toutes les données, puis pour les données normales uniquement et enfin pour les données anormales uniquement. Pour compléter l'étude, nous avons déduit, à partir des scores, les longueurs des chemins et nous avons représenté leur distribution. Le but de cette expérience est de tester si nous pouvons facilement trouver un seuil de score ou un seuil de profondeur permettant de séparer les données normales des données anormales. Le seuil de score ou de profondeur est représenté par la ligne verticale rouge. Nous remarquons que beaucoup de données normales ont un score entre 0.5 et 0.6. Il est clair que le seuil de 0.6 est beaucoup plus approprié que celui de 0.5 pour les deux jeux de données.

Avec un seuil de score de 0.6, pour le jeu de données Synthetic_2, où la densité de données normales est faible, seules quelques données anormales ont été détectées. Les distributions montrent que les vraies données anormales ont un chemin plus long que le seuil et un score inférieur au seuil de score (0.6). En revanche, avec le jeu de données Synthetic_3, toutes les données anormales ont un score supérieur au seuil de décision. En outre, en comparant Synthetic_2 et Synthetic_3, nous pouvons remarquer qu'une plus grande distance entre les données normales et anormales engendre une petite longueur de chemin pour les anomalies. Cela signifie que les données anormales sont rapidement isolées lorsqu'elles sont très différentes des données normales. La différence de densité entre les données normales et anormales a donc un impact considérable sur l'efficacité de la détection pour IForest et pourrait être prise en considération lors du choix du seuil de détection.

3.4 Majority Voting IForest (MVIForest)

3.4.1 Majority Voting IForest (MVIForest)

IForest identifie les anomalies en se basant sur une décision collective produite par l'ensemble des arbres construits lors de la phase d'apprentissage. En effet, chaque arbre i participe à la prise de décision par le chemin $h_i(x)$ pour

chaque donnée x. Le chemin moyen d'une donnée est utilisé dans le calcul de son score comme suit : $s(x,n) = 2^{-\frac{\sum_{i=1}^{t}(h_i(x))/t}{C(n)}}$. Cette formule implique qu'il faut calculer pour tous les arbres de la forêt, le parcours de x pour en déduire son score. Cette façon de procéder peut être améliorée en appliquant le vote majoritaire. Le temps d'exécution ainsi que le besoin en mémoire peuvent être considérablement réduits avec le vote majoritaire sans perdre en efficacité.

Nous proposons une nouvelle version d'IForest basée sur le vote majoritaire que nous appelons MVIForest. Le principe de MVIForest consiste à considérer la décision de la majorité des arbres et à arrêter l'exécution lorsque cette majorité est atteinte, sans générer le score moyen. De la même manière que IForest, MVIForest est composé de deux phases. La phase de construction de la forêt se fait exactement comme pour IForest (2.2.7.2). La différence se situe au niveau de la phase de test. En effet, au lieu de calculer la longueur du chemin dans tous les arbres avant de calculer le score final de x, MVIForest calcule le score de x pour chaque arbre i. Il est noté par $s_i(x,n) = 2^{-\frac{h_i(x)}{C(n)}}$. Ce score est comparé au score seuil (commun à tous les arbres) et une décision locale sera prise par l'arbre considéré. Ce processus est répété pour x successivement dans chaque arbre de la forêt jusqu'à ce qu'une décision majoritaire soit prise. La majorité correspond ici à $t/2+1$ arbres. Le processus de notation est donc terminé pour x et la décision finale est prise.

3.4.2 Evaluation de MVIForest sur des jeux de données réels

Pour évaluer la méthode MVIForest, nous avons réalisé différentes expériences sur les quatre jeux de données publics réels décrits précédemment (2.3.1). Nous avons choisi de considérer les jeux de données utilisés dans l'article original d'IForest. L'objectif de ces expériences est de comparer les performances de MVIForest et d'IForest. Les métriques utilisées pour ces évaluations sont décrites dans 2.3.2. Le tableau 3.4 résume les valeurs sélectionnées des paramètres utilisés pour réaliser les expériences. Les expériences de cette section ont été réalisées sur un ordinateur 64 bits avec un processeur Intel Core i5-4570 @ 3.20GHZ * 4. Les différents résultats sont présentés dans le tableau 4.3.

On peut remarquer que MVIForest donne des résultats similaires à IForest en

Tableau 3.4 – Valeurs des paramètres utilisés pour chaque jeu de données.

Jeux de données	t	ψ	threshold
Shuttle	100	1024	0.6
HTTP	100	1024	0.6
SMTP	100	2048	0.6
Forest Cover	100	4096	0.6

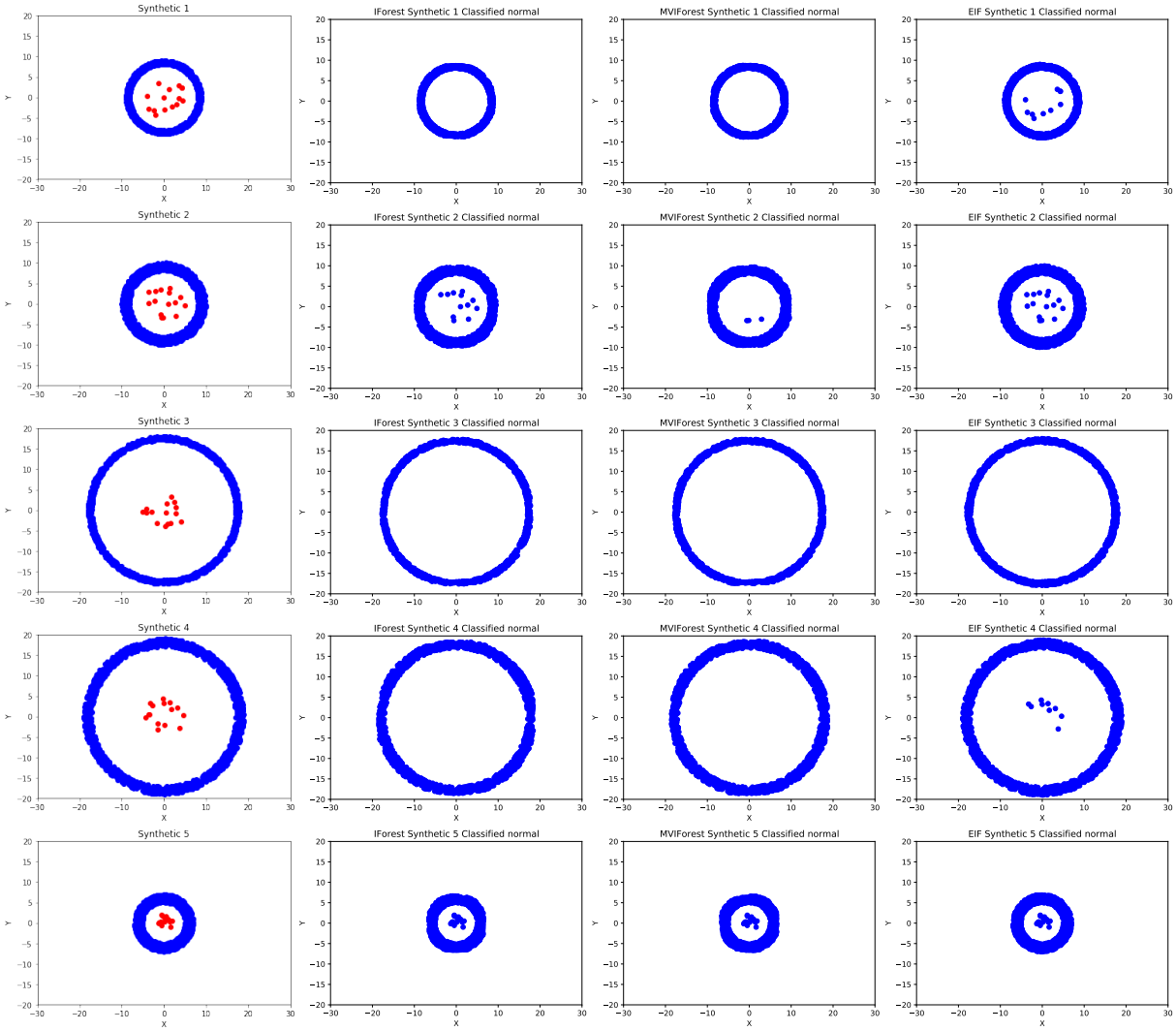
Tableau 3.5 – Comparaison entre MVIForest (MVIF) et IForest (IF) sur des jeux de données réels.

	Forest Cover		**Shuttle**		**SMTP**		**HTTP**	
Metrics	*MVIF*	*IF*	*MVIF*	*IF*	*MVIF*	*IF*	*MVIF*	*IF*
ROC AUC	0.8	**0.86**	0.98	0.98	**0.84**	0.83	0.99	0.99
Recall	0.7	**0.87**	0.98	**0.99**	**0.75**	0.70	1	1
Specificity	**0.9**	0.84	**0.98**	0.97	0.92	**0.93**	0.98	0.98
FAR (%)	**9.82**	15.54	**2.36**	3.01	7.81	**7.24**	**2.17**	2.43
F1 Score	**0.12**	0.1	**0.86**	0.83	0.01	0.01	**0.27**	0.25
CPU Time (s)	**677.69**	930.88	**88.25**	137.75	**142.6**	227.8	**836.92**	1393.49

termes de ROC AUC. Cependant, MVIForest est toujours plus rapide qu'IForest avec un temps d'exécution réduit de 35% en moyenne. Le meilleur résultat est obtenu avec le jeu de données HTTP, le plus grand jeu de données considéré, où le temps d'exécution de MVIForest ne représente que 60% du temps d'exécution d'IForest. Lorsque les anomalies sont évidentes et faciles à détecter, avec une distance élevée par rapport aux données normales et une densité très différente, MVIForest peut économiser jusqu'à 50% du temps d'exécution de la phase de test. Cependant, lorsque la plupart des données se trouvent dans la zone d'incertitude entre les anomalies et les données normales, le temps d'exécution sera presque le même.

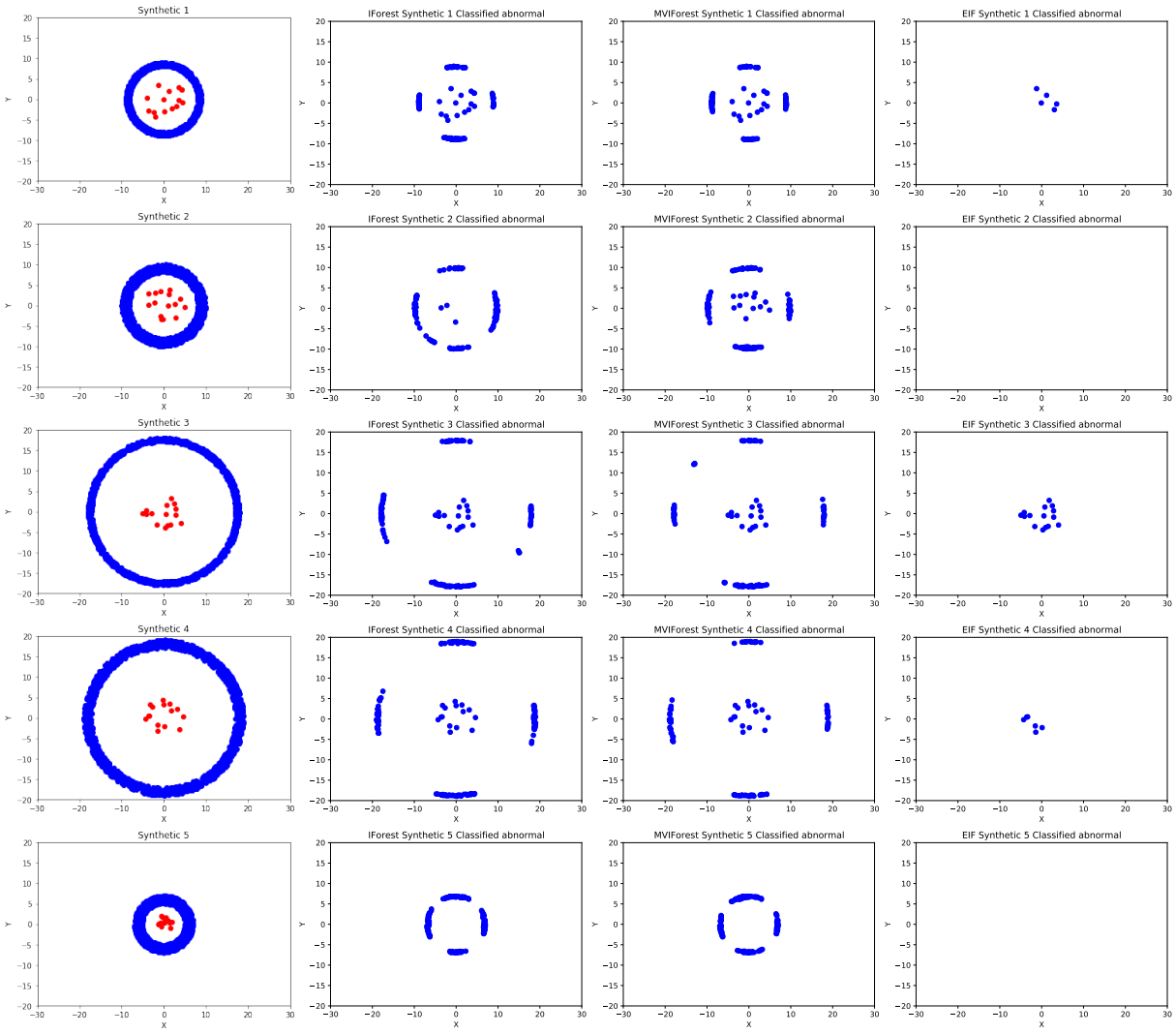
3.5 IForest VS EIF VS MVIForest

Le tableau 3.6 et les figures 3.11 et 3.12 montrent les résultats de l'exécution de IForest, EIF et MVIForest sur les ensembles de données synthétiques décrits



(a) Dataset (b) IForest (c) MVIForest (d) EIF

FIGURE 3.11 – Données classées normales par IForest, EIF et MVIForest respectivement dans les jeux de données Synthetic_1, Synthetic_2, Synthetic_3, Synthetic_4 and Synthetic_5 .



(a) Dataset (b) IForest (c) MVIForest (d) EIF

FIGURE 3.12 – Données classées anormales par IForest, EIF et MVIForest respectivement dans les jeux de données Synthetic_1, Synthetic_2, Synthetic_3, Synthetic_4 and Synthetic_5 .

Tableau 3.6 – Comparaison entre MVIForest (MVIF), IForest (IF) et ExtendedIForest (EIF) sur des jeux de données synthétiques à 2 dimensions.

	Synthetic_1			**Synthetic_2**			**Synthetic_3**		
Metrics	*MVIF*	*IF*	*EIF*	*MVIF*	*IF*	*EIF*	*MVIF*	*IF*	*EIF*
ROC AUC	**0.97**	**0.97**	0.67	**0.86**	0.56	0.5	0.97	0.96	**1**
Recall	**1**	**1**	0.33	**0.8**	0.2	0	1	1	1
Specificity	0.94	0.93	**1**	0.93	0.92	**1**	0.94	0.92	**1**
FAR (%)	6	6.67	**0**	6.86	7.6	**0**	5.73	7.73	**0**
F1 Score	0.25	0.23	**0.5**	**0.18**	0.04	-	0.26	0.20	**1**
CPU Time (s)	**5.68**	5.75	16.56	**5.28**	5.82	19.54	**5.3**	6.43	19.17
Memory (Mo)	**9.29**	9.56	11.92	**8.92**	9.04	11.66	**9.43**	9.53	12.01

	Synthetic_4			**Synthetic_5**		
Metrics	*MVIF*	*IF*	*EIF*	*MVIF*	*IF*	*EIF*
ROC AUC	**0.97**	0.96	0.7	0.45	0.45	**0.5**
Recall	**1**	**1**	0.4	0	0	0
Specificity	0.94	0.92	**1**	0.9	0.89	**1**
FAR (%)	5.4	7.93	**0**	9.93	10.06	**0**
F1 Score	0.27	0.20	**0.57**	-	-	-
CPU Time (s)	**7.07**	7.95	15.65	**5.18**	6.6	19.01
Memory (Mo)	9.33	**9.17**	12.11	**8.73**	9.06	11.49

dans la section 3.3.1. Comme on peut le voir dans ce tableau, contrairement aux jeux de données réels, MVIForest donne des résultats meilleurs qu'IForest ou similaires en termes de ROC AUC. Sur ce critère, MVIForest dépasse également les performances de la méthode EIF dans presque tous les cas.

Le seul jeu de données pour lequel les trois méthodes ont été efficaces est Synthétique_3 où il y a une forte densité de données normales, une faible densité de données anormales et une grande distance entre les données normales et les données anormales. La différence de densité entre les données normales et anormales est un critère important pour la détection d'anomalies. Cette densité doit également être considérée après l'échantillonnage, c'est-à-dire la densité des données utilisées pour chaque arbre. En effet, si la densité des données de l'échantillon est proche de la densité des anomalies dans le jeu de données de test (l'échantillon), les anomalies peuvent être ratées.

Selon le tableau 3.6, pour le jeu de données Synthetic_5, toutes les anomalies

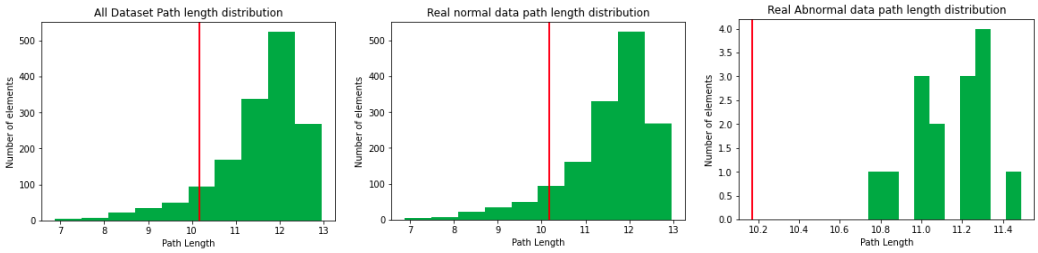
ont été ratées par les trois méthodes. Cela s'explique par la forte densité d'anomalies (bien qu'elles soient relativement peu nombreuses) dans ce jeu de données. En raison de cette forte densité, ces anomalies sont globalement perçues comme des données normales. C'est l'effet de masquage.

La figure 3.12 montre les anomalies détectées par chaque méthode pour chacun des 5 jeux de données. On peut remarquer que pour tous ces jeux de données, IForest et MVIForest donnent des résultats assez similaires en termes de détection. De plus, ils génèrent à chaque fois de fausses alarmes en classant les données normales à la frontière comme anormales. Comme expliqué ci-dessus, cela est dû à la façon dont les nœuds sont divisés : IForest sélectionne aléatoirement un attribut à chaque scission d'un noeud. Ce problème a été corrigé par la méthode EIF en introduisant des hyperplans. Comme on peut le remarquer, sur les mêmes ensembles de données, EIF n'a pas fait de fausses alarmes et a réalisé une spécificité maximale (toujours égale à 1 dans le tableau 3.6). EIF classifie correctement toutes les données normales, mais il manque les données anormales lorsque la distance qui les sépare des données normales est faible. Dans une telle configuration, la méthode EIF produit un rappel très faible (nul pour Synthetic_2).

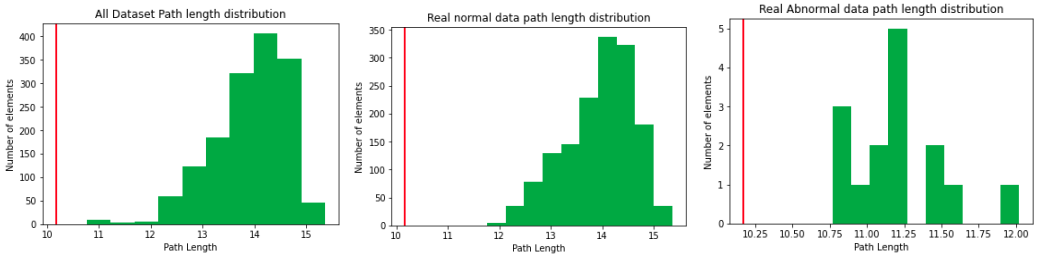
En outre, la prise en compte simultanée de toutes les dimensions de données lors du fractionnement des nœuds rend EIF plus lent qu'IForest. Comme on peut le voir dans le tableau 3.6, l'exécution de EIF prend environ 3 fois plus de temps que IForest. Ce temps est principalement consommé pendant la phase d'apprentissage correspondant à la construction de la forêt. MVIForest reste la plus rapide des trois méthodes, avec une détection plus efficace que IForest (une valeur de ROC AUC supérieure ou similaire).

Les besoins en mémoire des trois méthodes sont également indiqués dans le tableau 3.6. On peut remarquer que EIF a toujours le besoin de mémoire le plus élevé. En effet, pour chaque nœud des t arbres, EIF stocke les paramètres de l'hyperplan utilisé pour scinder ce nœud, ce qui représente une information supplémentaire par rapport à IForest. MVIForest et IForest ont presque les mêmes besoins en mémoire. MVIForest utilise légèrement moins de mémoire que IForest car l'exécution est interrompue dès qu'un vote majoritaire est déclenché. Les scores de tous les arbres ne sont pas stockés pour calculer une moyenne, et

la donnée n'est pas forcément testée sur tous les arbres. Puisque la détection d'anomalies exige la rapidité et l'efficacité, MVIForest serait un choix judicieux. Cependant, en fonction de l'objectif, on peut choisir EIF. EIF classifie mieux les données normales, mais peut manquer certaines anomalies, tandis que la méthode MVIForest est meilleure pour détecter rapidement les anomalies, mais peut générer de fausses alarmes. Les contraintes de l'application ainsi que le contexte doivent guider le choix de la méthode la plus adaptée.



(a) IForest



(b) EIF

FIGURE 3.13 – Distribution de la longueur des chemins obtenus avec une exécution de IForest et EIF sur le jeu de données Synthetic_5.

La figure 3.13 présente une exploration de la distribution de la profondeur des chemins de toutes les données, avec IForest et EIF, pour l'ensemble de données Synthetic_5. Nous nous sommes concentrés sur cet ensemble de données car aucune anomalie n'a été détectée. MVIForest n'a pas été considérée dans cette expérience car avec MVIForest, chaque donnée n'a pas un score moyen unique, mais plusieurs scores donnés par les différents arbres utilisés. Nous rappelons qu'une anomalie est caractérisée par un score élevé, résultant d'un parcours moyen relativement peu profond sur les arbres de la forêt. A partir du score seuil (0.5), nous avons calculé la profondeur seuil que nous avons

représentée par une ligne verticale rouge. La distribution des profondeurs nous permet de situer cette profondeur seuil d'abord par rapport à l'ensemble des données, puis en utilisant uniquement les données normales et enfin en utilisant uniquement les données anormales. Nous remarquons, pour les deux méthodes, un large chevauchement des profondeurs des chemins des données normales et anormales. Ce chevauchement met en évidence l'impossibilité de séparer les deux classes de données sur la base de leurs profondeurs dans les arbres IForest, quelle que soit la profondeur seuil. Avec IForest, certaines données normales ont une profondeur inférieure à la profondeur seuil et sont donc classées comme anomalies, ce qui n'est pas le cas avec la méthode EIF.

L'ensemble de données Synthetic_5 décrit dans la figure 3.4d met en évidence les limites d'IForest en termes de traitement des effets de swamping et de masquage. En considérant une seule dimension séparément, les données anormales ressemblent à des données normales et ces deux classes ne peuvent être séparées. Contrairement à EIF qui, grâce à sa façon de construire la forêt, a pu éviter le swamping, IForest et MVIForest ont mal classé plusieurs données normales car elles ne considèrent qu'une seule dimension à la fois. De plus, en raison de la forte densité de données anormales, elles n'ont pas pu être isolées assez tôt dans la forêt. Elles ont obtenu des profondeurs supérieures au seuil (voir figure 3.13) pour toutes les méthodes.

Nous avons réalisé les mêmes expériences sur les ensembles de données tridimensionnels décrits dans 3.3.1. Aucune des méthodes ne détecte d'anomalies à aucun moment. IForest et MVIForest ne font que des fausses alarmes. Ce résultat confirme l'intuition selon laquelle la performance de détection de IForest et même de MVIForest diminue considérablement avec l'augmentation du nombre de dimensions significatives dans le jeu de données. Une solution possible serait d'explorer des arbres plus profonds afin de pouvoir traiter toutes les dimensions successivement.

3.6 Discussion

La forêt d'isolation est l'une des meilleures méthodes pour détecter les anomalies. Elle est rapide, précise et ne nécessite pas d'énormes ressources par rapport

à d'autres techniques telles que le clustering ou le nearest neighbor. Dans ce chapitre, nous avons réalisé un état de l'art des méthodes de détection des anomalies basées sur l'isolation. La plupart d'entre elles sont des améliorations d'IForest. Chaque nouvelle méthode répond à une limite ou à une adaptation d'IForest à un nouveau contexte. Nous avons également mis en évidence certaines faiblesses d'IForest qui ne sont pas abordées dans ces améliorations, notamment le choix des paramètres d'entrée et l'impact des caractéristiques des jeux de données (nombre de dimensions significatives, densité de données normales et anormales, etc.). Nous avons testé IForest et sa version étendue (EIF) sur plusieurs jeux de données réels et synthétiques afin d'illustrer les faiblesses que nous avons identifiées. Nous avons ensuite proposé une amélioration d'IForest modifiant la manière dont elle prend ses décisions. Cette nouvelle version, que nous avons appelée MVIForest (Majority Voting IForest), est plus rapide que IForest, puisque son exécution est interrompue dès qu'une décision majoritaire est possible, sans solliciter tous les arbres.

Malgré la performance des méthodes basées sur l'isolation, abordées dans ce chapitre, elles ne sont pas adaptées au contexte du streaming. La détection d'anomalies dans les flux de données basée sur l'isolation n'a pas été suffisamment explorée dans la littérature. Dans le prochain chapitre, nous nous concentrons sur la version d'IForest adaptée au contexte des flux de données : IForest ASD.

Chapitre 4

Détection d'anomalies dans les flux de données

Sommaire

Les résultats obtenus dans ce chapitre ont été publiés dans la conférence internationale ICCSA 2020 :
Maurras Ulbricht TOGBE, Mariam BARRY, Aliou BOLY, Yousra CHABCHOUB, Raja CHIKY, Jacob MONTIEL et Vinh-Thuy TRAN. Isolation based Anomaly Detection for DataStreams using scikit-multiflow. The 20th International Conference on Computational Science and its Applications (ICCSA 2020), Springer.

4.1 Introduction

Un flux de données est défini comme une séquence infinie d'éléments générés de façon continue. L'exploration de flux de données est un sous-domaine de l'exploration de données qui traite continuellement les données entrantes à la volée. L'exploration de flux de données impose plusieurs défis, tels que la nature évolutive des données et leur taille énorme, qui est potentiellement infinie. Ces défis nécessitent des méthodes d'exploration efficaces et optimisées, en termes de temps de traitement et d'utilisation de la mémoire, qui sont adaptées au contexte du flux. Le traitement en temps réel des flux de données exige souvent un seul passage sur chaque donnée. Les flux de données sont utilisés dans plusieurs domaines comme la finance, la surveillance des réseaux, les télécommunications et les réseaux de capteurs. L'arrivée continue de flux de données de manière rapide, variable dans le temps et éventuellement illimitée peut soulever de nouveaux problèmes de recherche fondamentale. Alors que la recherche sur le contexte des flux de données ne cesse de progresser, le problème de l'alerte efficace en cas de comportement de données ou de modèles anormaux (valeurs aberrantes) reste un défi important à relever. En effet, les valeurs aberrantes doivent être détectées aussi rapidement que possible pour obtenir des informations à des fins d'exploitation. Les algorithmes de détection d'anomalies dans les flux font référence aux méthodes capables d'extraire suffisamment de connaissances des données afin de calculer les scores d'anomalies tout en traitant des flux de données en constante évolution.

De nombreuses solutions ont été récemment développées pour le traitement

des flux de données uni-source ou multi-source. L'idée principale de ces approches proposées consiste à traiter les flux de données à la volée avant de les stocker. Dans ce contexte, un framework récent a été développé pour les approches de flux de données multi-sorties, multi-labels appelé scikit-multiflow (Montiel et al., 2018) (voir la section 4.2.3.4). Ce framework comprend également l'algorithme populaire Half-space Trees (Tan et al., 2011) qui assure une détection rapide d'anomalies dans les flux de données. La principale motivation des développeurs de scikit-multiflow est d'encourager les chercheurs et les industriels à intégrer et à partager facilement leurs approches au framework et à rendre leurs travaux facilement accessibles par la communauté Python en pleine expansion. Par conséquent, nous avons décidé d'étendre le cadre en proposant d'autres approches de détection d'anomalies qui peuvent être appliquées dans le contexte des flux de données.

Dans ce chapitre, nous énumérons d'abord les frameworks existants d'analyse de flux de données évolutifs en soulignant leurs caractéristiques. Ensuite, nous présentons les algorithmes de détection d'anomalies qui ont été adaptés au contexte des flux en discutant des avantages et des inconvénients de chacun d'entre eux. De plus, nous présentons une implémentation de certaines méthodes qui se sont avérées efficaces dans la littérature dans le framework scikit-multiflow. En particulier, nous implémentons l'algorithme de détection d'anomalies basé sur l'isolation appelé IForest ASD proposé par Ding et Fei dans (Ding et Fei, 2013) et mettons en avant la simplicité du framework pour faciliter et accélérer son évolution. Ensuite, nous réalisons différentes expériences pour évaluer la performance prédictive et la consommation de ressources (en termes de mémoire et de temps) de la méthode IForest ASD et la comparons avec l'algorithme de détection d'anomalies bien connu pour les flux de données, Half-Space Trees.

A notre connaissance, il n'existe pas d'implémentation open source d'IForest ASD, ni sur Github, ni dans un framework de streaming.

D'après la revue des frameworks de gestion de flux de données, scikit-multiflow semble être le plus prometteur car il implémente la majorité des méthodes d'apprentissage de flux connues en utilisant le langage de programmation très connu Python impliquant une communauté croissante. Du fait qu'il ne contient qu'une seule méthode de détection d'anomalie, qui est Half-Spaces-Trees

(Tan et al., 2011), nous avons décidé d'étendre le framework en implémentant une approche de détection d'anomalie basée sur l'algorithme Isolation Forest proposé par Ding et Fei dans (Ding et Fei, 2013).

Par conséquent, les motivations derrière notre implémentation fournissant un algorithme basé sur l'isolation dans Scikit-multiflow sont les suivantes :

— **Isolation Forest** est un algorithme performant pour la détection d'anomalies et est la seule méthode d'ensemble dans Scikit-learn et qui est largement utilisée par la communauté de la détection d'anomalies. En outre, il s'agit d'un modèle basé sur les arbres, ce qui convient parfaitement à l'apprentissage en ligne et incrémental.

— **Scikit-multiflow** est le principal framework de traitement de flux de données en Python qui comprend une variété d'algorithmes d'apprentissage et de méthodes adaptées aux flux de données.

Ce chapitre est organisé comme suit : la section 4.2 présente des généralités sur les flux de données. Dans la section 4.3, nous fournissons un aperçu et une classification des méthodes de détection d'anomalies adaptées aux flux de données. Dans la section 4.4, nous nous concentrons sur la description des algorithmes : Isolation Forest et sa variante implémentée pour le streaming (IForest ASD). Puis, dans la section 4.5, nous présentons des évaluations expérimentales, en comparant IForest ASD à Half-Space Trees avant de discuter des résultats obtenus. Enfin, la section 4.6 conclut le travail en proposant de futures directions de recherche.

4.2 Gestion des flux de données

L'utilisation des capteurs engendrant des flux de données hétérogènes à des débits variables est devenue omniprésente dans beaucoup de domaines. Il est en général impossible de stocker un flux de données dans sa totalité. Pour pallier les limites des Systèmes de Gestion de Bases de Données (SGBD) traditionnels, des chercheurs ont proposés des Systèmes de Gestion de Flux de Données (SGFD) (Abdessalem et al., 2007) qui permettent d'exécuter des requêtes sans interruption et renvoient de nouveaux résultats au fur et à mesure que de nouvelles données arrivent. Le stockage des flux de données est trop coûteux, même dans

les SGFD, à cause du volume important des données. En vue de réduire le volume des données à stocker et de ne garder que la partie pertinente des données, certains chercheurs ont proposé des solutions d'échantillonnage de flux de données (Chiky et al., 2008 ; Haas, 2016), d'autres ont conçu des techniques de résumé de flux de données (Dia et al., 2018 ; Ahmed, 2019).

Cette section présente les généralités sur la gestion des flux de données. Dans un premier temps, nous présentons le système de fenêtrage (section 4.2.1. Nous discutons des contraintes et défis de l'analyse des flux de données (section 4.2.2) dans un second temps. Pour finir cette section, quelques frameworks et librairies d'analyse de flux de données sont exposés dans la section 4.2.3. Toutes ces techniques fonctionnent généralement sur des fenêtres du flux.

4.2.1 Fenêtrage

Par définition, un flux de données est illimité, continu et généré à une vitesse variable. Le traitement de ces gros volumes de données continues et illimitées en les considérant dans leur globalité est très difficile, voire impossible. Une solution est de traiter une portion de ces données selon leur ordre d'arrivée d'où le système de fenêtrage (Ng et Dash, 2010). Une fenêtre serait donc un intervalle de temps ou une quantité fixe de données au bout duquel un ou des traitements seront effectués. Il existe de ce fait deux catégories de fenêtre : la fenêtre physique et la fenêtre logique.

4.2.1.1 Fenêtre physique

La fenêtre physique est celle basée sur le temps. Il s'agit donc de définir un intervalle de temps, soit une date de début et une date de fin. Ceci implique trois types de fenêtres physiques différents. Il est important de noter qu'avec les fenêtres physiques, la taille de données récoltées (en termes de nombre d'éléments) peut varier d'une fenêtre à une autre.

— Fenêtre fixe : La fenêtre physique fixe est celle dont la date de début et de fin sont bien fixées. C'est la fenêtre la plus simple. Un exemple (figure 4.1) dans le cadre des enquêtes policières serait de récupérer les

enregistrements vidéos ou les données collectées par des capteurs entre le moment exact de début d'un interrogatoire soit t_d et le moment exacte de la fin de celui-ci soit t_f pour analyser les expressions faciales et fréquences cardiaques du détenu.

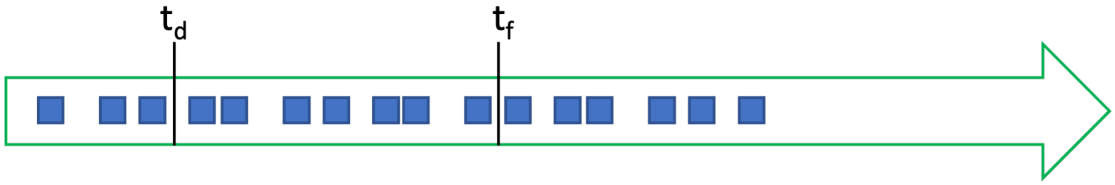


FIGURE 4.1 – Fenêtre physique fixe.

— Point de repère : La fenêtre physique de type point de repère (figure 4.2) est celle dont la date de début est connue et fixe mais que la date de fin est la date courante (l'instant présent). Évidemment, la taille de la fenêtre (le volume de données) augmente avec le temps. Un exemple serait de collecter les données liées à votre sentiment entre le moment exact où vous avez commencé la lecture de ce chapitre et l'instant où vous lisez ce mot soit l'instant présent.

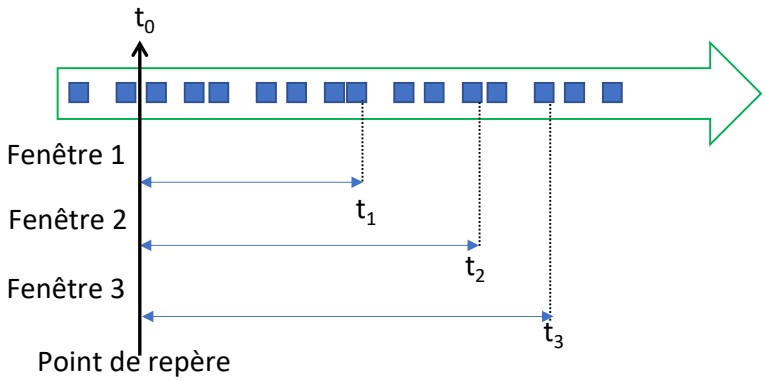


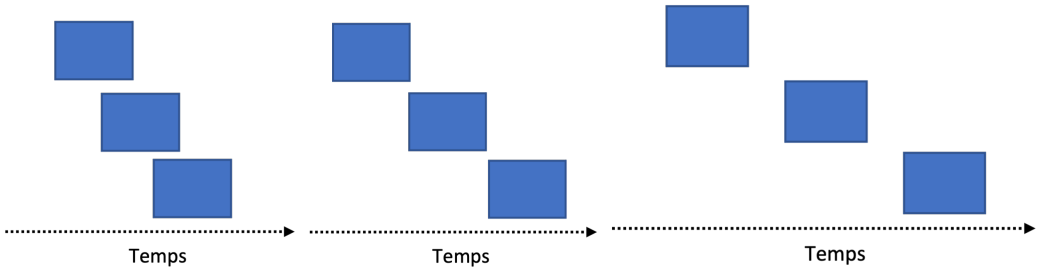
FIGURE 4.2 – Fenêtre physique point de repère.

— Fenêtre glissante : La fenêtre physique glissante est la plus utilisée. Comparativement aux deux autres fenêtres vues précédemment, ses dates de début et de fin peuvent évoluer avec le temps et selon les besoins. On dit que ses bornes sont relatives. Ainsi défini, on peut distinguer trois

variantes de fenêtre glissante en fonction de leur taille et de leur pas de décalage.

1. La fenêtre purement glissante (sliding window) dont le pas de décalage est inférieur à la taille de la fenêtre (figure 4.3a) ;
2. La fenêtre sautante (jumping window) dont le pas de décalage est égal à la taille de la fenêtre (figure 4.3b) ;
3. La fenêtre bondissante (hopping window) dont le pas de décalage est supérieur à la taille de la fenêtre (figure 4.3c).

La figure 4.3 montre les différentes variantes de fenêtres glissantes.



(a) Purement glissante (b) Sautante (c) Bondissante

FIGURE 4.3 – Variantes de fenêtre glissante.

4.2.1.2 Fenêtre logique

Contrairement aux fenêtres physiques, les fenêtres logiques sont basées sur le nombre de données. Ainsi, selon les contraintes, la taille de la fenêtre est fixe et connue d'avance. Un exemple serait de récupérer et de traiter chaque 1000 derniers battements d'un coeur. Une fenêtre logique est la fenêtre appelée session. Une fenêtre session reste ouverte tant que, pendant un temps prédéfini (délai d'attente), une donnée est produite. La session s'arrête et la fenêtre est fermée quand à la fin du délai d'attente aucune donnée n'est produite.

Une grande différence entre une fenêtre physique et une fenêtre logique est que la taille d'une fenêtre logique est connue à priori tandis-que celle d'une fenêtre physique est connue à posteriori.

En contrepartie, le temps est mieux contrôlé au niveau des fenêtres physiques. On sait à quelle fréquence un traitement sera lancé, ce qui n'est pas le cas pour les fenêtres logiques car la vitesse de génération des données est variable. Vu les caractéristiques des flux de données (rapide, continu, illimité, vitesse variable), le choix du type de fenêtre dépendra du domaine d'application et des contraintes ainsi que de l'algorithme à utiliser.

4.2.2 Contraintes et défis

La nature évolutive des flux de données soulève de multiples exigences qui doivent être prises en compte par les algorithmes de type batch. Dans cette section, nous présentons certaines contraintes principales rencontrées dans l'environnement des flux (Aggarwal et Philip, 2007 ; João Gama et Gaber, 2007).

— Données évolutives : la nature infinie des flux de données évolutives nécessite un traitement en temps réel - ou quasi réel - afin de prendre en compte l'évolution et la vitesse des données.
— Contrainte de mémoire : les flux de données étant infinis, les algorithmes de flux doivent utiliser une mémoire limitée en stockant le moins possible d'observations ainsi que des informations statistiques concernant les données traitées vues jusqu'à présent.
— Contrainte de temps : Pour ne pas cumuler de retard sur le traitement de flux de données, une contrainte forte pour les algorithmes de stream mining est d'avoir un traitement plus rapide que la vitesse d'arrivée de données.
— Concept drift : le concept drift est un phénomène qui se produit lorsque les caractéristiques du flux de données (moyenne, variance...) changent dans le temps. Le défi que pose le concept drift a fait l'objet de multiples recherches. Une étude complète sur le concept drift est présentée dans (João Gama, Žliobaitė et al., 2014).

Afin de gérer les contraintes présentées ci-dessus, les algorithmes d'analyse de flux de données doivent utiliser des stratégies de traitement incrémental pour pouvoir travailler en flux. Il existe plusieurs techniques (Aggarwal et Philip, 2007 ; João Gama et Gaber, 2007 ; Ikonomovska et al., 2007) qui peuvent être

utilisées afin de répondre à ces contraintes. Nous en résumons quelques-unes ci-dessous :

— Passage unique : puisque le flux de données évolue dans le temps et n'est pas stocké exhaustivement, chaque donnée ne peut être traitée plus d'une fois.
— Fenêtrage : des modèles de fenêtres ont été proposés pour maintenir une partie du contenu du flux en mémoire. Il existe différents modèles de fenêtres (Section 4.2.1) (Ng et Dash, 2010).
— Construction de synopsis : une variété de techniques de résumé peut être utilisée pour la construction de synopsis dans les flux de données évolutives (Aggarwal, 2007), comme les méthodes d'échantillonnage, les histogrammes, les croquis ou les méthodes de réduction de dimension.

4.2.3 Framework et librairies d'analyse de flux de données

L'analyse des flux de données devant répondre à certaines contraintes, des frameworks et librairies dédiés aux flux de données ont été proposés dans la littérature. Quelques frameworks et librairies comme MOA, SAMOA, StreamDm et Scikit-Multiflow sont présentés dans cette section.

4.2.3.1 MOA : Massive Online Analysis

MOA (Albert Bifet, Holmes et al., 2010) est le framework open source le plus populaire pour l'exploration de flux de données en Java. Il comprend une collection d'algorithmes d'apprentissage automatique (classification, régression, regroupement, détection d'aberrations, détection de concepts drift et systèmes de recommandation), y compris des générateurs de données, des outils d'évaluation et une interface graphique permettant de visualiser les résultats des expériences. MOA est lié au Waikato Environment for Knowledge Analysis (WEKA) Witten et al. (2017). Dans (Albert Bifet, Gavaldà et al., 2018), les auteurs ont exploré le domaine de l'apprentissage en ligne à partir d'esquisses et d'approches de détection de drift avec des algorithmes supervisés et non supervisés pour les flux de données. Quelques exemples pratiques avec MOA sont également fournis. De

plus amples informations sont disponibles dans le manuel MOA [1]. Behera et al. ont fourni dans (Behera et al., 2017) une étude comparative des outils distribués pour l'analyse des données en continu, avec une comparaison qualitative entre Apache Spark, Storm, Samza et Apache S4.

De nombreuses bibliothèques logicielles open-source utilisent MOA pour effectuer des analyses de flux de données dans leurs systèmes (Albert Bifet, Gavaldà et al., 2018), notamment ADAMS, MEKA et OpenML.

4.2.3.2 Apache SAMOA

Apache SAMOA (A. Bifet et GDF Morales, 2014) est une plateforme open source pour l'exploitation des flux de données volumineux. Il s'agit d'un framework qui contient des algorithmes distribués d'apprentissage automatique des flux de données.

4.2.3.3 StreamDM

StreamDM (Albert Bifet, Maniu et al., 2015) est une bibliothèque open source d'exploration de données et d'apprentissage automatique, conçue au-dessus de Spark Streaming, une extension de l'API Spark de base qui permet le traitement évolutif des flux de données.

4.2.3.4 Scikit-Multiflow

Scikit-multiflow (Montiel et al., 2018) est un framework open-source écrit en Python pour l'apprentissage multi-label/multi-sortie et single-label de flux de données évolutifs. Scikit-multiflow s'inspire des frameworks populaires *scikit-learn* [2] et MOA. Étant donné la popularité croissante du langage de programmation Python, l'avantage de scikit-multiflow est qu'il complète scikit-learn qui est largement utilisé par la communauté de la science des données, et qui se concentre principalement sur du batch. Scikit-multiflow fusionne actuellement avec le framework creme (Halford et al., 2020), donnant naissance à un nouveau

1. https://moa.cms.waikato.ac.nz/documentation/
2. https://scikit-learn.org

framework python pour l'apprentissage automatique en ligne en python, nommé RIVER [3].

Scikit-multiflow comprend une collection de divers algorithmes et méthodes d'apprentissage : des méthodes basées sur les règles et les arbres telles que Hoeffding Anytime Tree ou Extremely Fast Decision Tree (Manapragada et al., 2018), des méthodes d'ensemble telles que le classificateur Adaptive Random Forest (Gomes, Albert Bifet et al., 2017). La liste des algorithmes existants est disponible sur la page web officielle [4]. Comme MOA, scikit-multiflow comprend également une collection de divers algorithmes de flux pour différentes tâches, fournit des outils d'évaluation et offre la possibilité d'étendre facilement les algorithmes.

Half-Spaces Trees est la seule méthode de détection d'anomalies dans la version actuelle de scikit-multiflow.

4.3 Détection d'anomalies dans les flux de données : état de l'art

La détection d'anomalies est une application transversale à tout domaine exploitant n'importe quel type de données. La détection d'anomalies dans les flux de données présente néanmoins quelques particularités émanant des caractéristiques des flux de données. Dans cette section, nous proposons un état de l'art des méthodes de détection d'anomalies dans les flux de données que nous classifions selon différents critères.

La détection anomalies dans les flux de données (ADiDS : Anomaly Detection in Data Stream) présente de nombreux défis en raison des caractéristiques de ce type de données. Un défi important est que le traitement du flux de données doit être effectué en un seul passage pour faire face aux limites de mémoire et pour un traitement en temps réel. Ainsi, les différentes approches de détection d'anomalies hors ligne existantes, telles que l'approche statistique, l'approche de clustering, etc, (Hodge et Austin, 2004 ; Chandola et al., 2009 ; Aggarwal,

3. https://github.com/online-ml/river
4. https://scikit-multiflow.github.io/scikit-multiflow/package_map.html

2017), ne sont pas adaptées aux flux de données car elles nécessitent de nombreux passages sur le jeu de données. Elles nécessitent également de disposer de l'ensemble du jeu de données pour pouvoir détecter les anomalies. Nous trouvons dans la littérature quelques approches qui ont été adaptées ou de nouvelles méthodes conçues pour ADiDS. Les auteurs M Gupta, Gao, Aggarwal et al. présentent dans (M Gupta, Gao, Aggarwal et al., 2014), une étude des méthodes de détection d'anomalies qui peuvent être appliquées aux données temporelles, en mettant l'accent sur les flux de données. Ils ont présenté des modèles de prédiction évolutifs, une approche basée sur la distance et la détection des valeurs aberrantes dans les flux de données à grande dimension. D'autres études sur la détection d'anomalies dans le contexte des flux de données comme Thakkar et al. (2016), Salehi et Rashidi (2018) et Tellis et D'Souza (2018) peuvent être proposées. Dans l'étude (Thakkar et al., 2016), les auteurs présentent certains problèmes de la détection d'anomalies dans le flux de données, comme le concept drift, l'incertitude et le taux d'arrivée. Une étude détaillée sur les méthodes de détection d'anomalies dans les séries temporelles et les flux multidimensionnels est proposée dans le livre (Aggarwal, 2017, Chap. 9). Aggarwal y présente différentes approches telles que les méthodes basées sur les probabilités, les prédictions et la distance.

En général, les méthodes de détection d'anomalies sont basées sur le fait que les anomalies sont rares et ont un comportement différent par rapport aux données normales. Ces caractéristiques sont vraies pour les ensembles de données statiques et aussi pour les flux de données. Les approches de détection d'anomalies les plus utilisées sont celles statistiques, clustering, plus proches voisins. Nous allons présenter ces approches ci-dessous et nous concentrer sur l'approche basée sur l'isolation : Isolation Forest, proposée en 2008 par FT Liu et al. dans (FT Liu et al., 2008).

4.3.1 Méthodes statistiques

Les méthodes basées sur l'approche statistique établissent généralement un modèle qui caractérise le comportement normal en fonction de l'ensemble des données. Les nouvelles données entrantes qui ne correspondent pas au modèle ou

qui ont une très faible probabilité de correspondre au modèle sont considérées comme anormales. Certaines méthodes attribuent un score aux données en fonction du degré de déviation par rapport au modèle (Yamanishi et al., 2004). Les méthodes basées sur l'approche statistique peuvent être paramétriques, auquel cas elles doivent avoir une connaissance préalable de la distribution de l'ensemble des données. Elles peuvent aussi être non paramétriques, c'est-à-dire qu'elles apprennent de l'ensemble de données fourni pour en déduire la distribution sous-jacente. Dans le contexte du flux de données, une telle connaissance préalable n'est pas toujours disponible.

4.3.2 Clustering et Plus proches voisins

Les approches de clustering et de plus proches voisins sont basées sur la proximité entre les observations. Les méthodes de cette catégorie sont basées soit sur la distance, soit sur la densité. Les méthodes de clustering divisent le jeu de données en différents groupes (clusters) en fonction de la similarité entre les observations. Le groupe le plus éloigné ou celui qui présente la plus faible densité peut être considéré comme un groupe d'anomalies (Aggarwal, Han et al., 2003 ; Assent et al., 2012). Les méthodes des plus proches voisins déterminent les voisins d'une observation en calculant la distance entre toutes les observations du jeu de données. L'observation qui est éloignée de ses k plus proches voisins peut être considérée comme une anomalie (Angiulli et Fassetti, 2007). Elle est également caractérisée comme l'observation qui a le moins de voisins dans un rayon r (un paramètre fixe) (Pokrajac et al., 2007). Ces approches doivent calculer la distance ou la densité entre toutes les observations de l'ensemble de données ou elles doivent avoir des connaissances préalables sur l'ensemble de données, ce qui n'est pas toujours possible. De plus, elles présentent souvent l'inconvénient de nécessiter des ressources considérables en CPU, temps de traitement et/ou utilisation mémoire.

4.3.3 Méthodes basées sur l'isolation

Introduit par FT Liu et al. (2008), le principe de l'approche basée sur l'isolation est d'isoler les observations anormales de l'ensemble des données. Les

données anormales sont censées être très différentes des données normales. Elles sont également censées représenter une très faible proportion de l'ensemble des données. Ainsi, elles sont susceptibles d'être rapidement isolées. Certaines méthodes basées sur l'isolation sont présentées dans la section 4.4. Les méthodes basées sur l'isolation sont différentes des autres approches statistiques, clustering ou de plus proches voisins, car elles ne calculent pas de distance ou de densité à partir de l'ensemble de données. Par conséquent, elles sont moins complexes et plus évolutives. Elles ne souffrent pas du problème de la consommation de CPU, de mémoire ou de temps et sont ainsi mieux adaptées au contexte du flux de données.

Le tableau 4.1 résume les avantages et les inconvénients des approches existantes pour l'ADiDS.

Il existe de nombreuses méthodes adaptées ou conçues pour ADiDS dans la littérature. Elles utilisent généralement le concept de flux de données de la fenêtre pour calculer l'anomalie (Salehi et Rashidi, 2018). La figure 4.4 présente une classification de quelques méthodes de détection d'anomalies pour chaque catégorie existant dans la littérature et applicables aux flux de données.

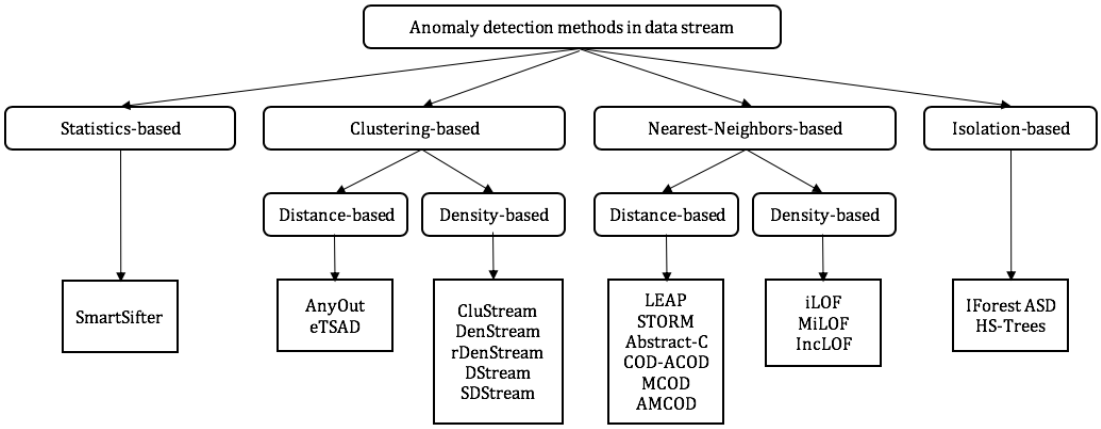


Figure 4.4 – Classification des méthodes de détection d'anomalies dans les flux de données.

Un travail récent (Togbe et al., 2020) sur les approches de détection d'anomalies a montré que l'algorithme Isolation Forest a de bonnes performances par rapport aux autres méthodes. Ainsi, nous avons décidé d'implémenter une adaptation de Isolation Forest au contexte des flux de données pour compléter

Tableau 4.1 – Comparaison des approches de ADiDS.

Approches	**Forces**	**Faiblesses**
Statistiques	Les méthodes non paramétriques sont adaptées au contexte des flux de données	— Les méthodes paramétriques sont difficiles à appliquées aux flux de données — Les méthodes non paramétriques ne peuvent être utilisées que pour les flux de données à faible dimensions
Plus proches voisins	Les méthodes basées sur la distance — adaptées pour la détection d'anomalies globales Les méthodes basées sur la densité — adaptées à la détection d'anomalies locales — plus efficaces que les méthodes basées sur la distance	Les méthodes basées sur la distance — ne sont pas adaptées aux densités hétérogènes — ont un coût très élevé pour l'exécution sur des flux de données à grandes dimensions Les méthodes basées sur la densité — ont une grande complexité — ne sont pas efficaces pour les flux de données de grandes dimensions
Clustering	Adaptées à l'identification des clusters	Non optimisées pour la détection d'anomalies individuelles
Isolation	— ont un faible temps d'exécution et une faible consommation de mémoire — sont efficaces pour la détection d'anomalies	— La performance dépend beaucoup des paramètres de fenêtrage et de la mise à jour du modèle — pas simple à adapter aux données catégorielles et textuelles

le framework scikit-multiflow.

4.4 Détection d'anomalies dans Scikit-Multiflow : HST et IForest ASD

La première méthode proposée dans la catégorie des méthodes basées sur l'isolation est Isolation Forest (IForest) (FT Liu et al., 2008 ; FT Liu et al., 2012). L'une des limites d'IForest est qu'elle a été conçue pour un ensemble de données statiques et non pour un flux de données. Ding et Fei proposent dans (Ding et Fei, 2013) une amélioration de IForest pour l'adapter au contexte des flux de données en utilisant des fenêtres glissantes. La méthode proposée est nommée Isolation Forest Algorithm for Stream Data (IForest ASD).

Il existe d'autres améliorations et adaptations d'Isolation Forest telles que Extended Isolation Forest (Hariri, Kind et Brunner, 2018b) ou Functional Isolation Forest (Staerman et al., 2019) mais elles sont conçues pour des paramètres statiques et ne sont pas adaptées aux flux de données.

Dans cette section, nous rappelons brièvement les caractéristiques importantes de l'algorithme IForest puis nous expliquons l'algorithme IForest ASD que nous avons implémenté dans scikit-multiflow. Nous abordons également, brièvement, l'unique algorithme de détection d'anomalies intégré à scikit-multiflow : Half-Space Trees (HST).

4.4.1 IForest et IForest ASD

IForest est une méthode efficace pour la détection d'anomalies avec une complexité, une consommation de CPU et de temps relativement faibles. Elle nécessite toutes les données au départ pour construire t échantillons aléatoires. Elle nécessite également de nombreux passages sur le jeu de données pour construire la forêt d'arbres aléatoires. Elle n'est donc pas adaptée au contexte des flux de données. Dans (Ding et Fei, 2013), les auteurs ont proposé la méthode dite IForest ASD qui est une amélioration de IForest pour ADiDS. IForest ASD utilise une fenêtre glissante pour traiter les données en continu. Sur la fenêtre complète actuelle, IForest ASD exécute la méthode IForest standard pour obtenir la forêt

aléatoire. Ceci est le modèle de détecteur IForest pour IForest ASD. La méthode IForest ASD peut également traiter le concept drift dans le flux de données en maintenant un taux d'anomalies souhaité en entrée (u). Si le taux d'anomalies dans la fenêtre considérée est supérieur à u, alors un concept drift s'est produit. IForest ASD supprime son détecteur IForest actuel et en construit un autre en utilisant toutes les données de la fenêtre considérée. La figure 4.5 représente le workflow utilisé dans IForest ASD pour mettre à jour le modèle. Dans la section 4.5, nous présentons les résultats de nos expériences sur notre implémentation de cet algorithme dans le framework scikit-multiflow en utilisant les méthodes d'ajustement et de prédiction incrémentales.

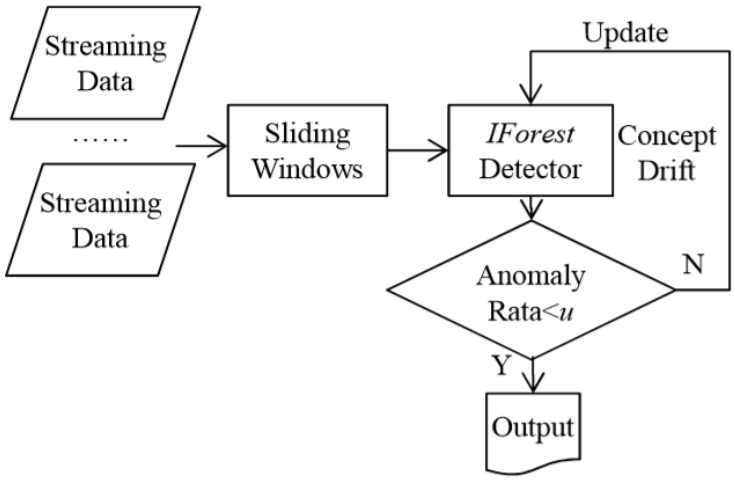


Figure 4.5 – Workflow de l'algorithme Isolation Forest ASD pour la détection de drift implémenté dans scikit-multiflow. Image extraite du papier original (Ding et Fei, 2013).

4.4.2 Half-Spaces Trees (HST)

Half-Space Trees (que nous désignerons par HST) est un algorithme de détection d'anomalies en ligne proposé par Tan et al. dans (Tan et al., 2011), qui traite des flux de données. Cette méthode utilise une approche d'apprentissage non supervisé qui peut apprendre des motifs à partir de flux de données avec des attributs numériques continus où le nombre d'attributs est constant.

Le principe de l'algorithme consiste à diviser l'espace des attributs en un ensemble de sous-espaces en utilisant des arbres binaires (appelés Half-space tree) pour construire l'ensemble. Chaque nœud des arbres a un attribut de scission choisi aléatoirement et représente un espace de travail donné (plage de valeurs pour un attribut donné). Les deux nœuds enfants font référence à deux sous-espaces représentant chacun, un demi-espace de l'espace du nœud parent.

Ainsi, toute donnée dans le domaine peut parcourir un chemin unique de la racine d'un HST à l'une de ses feuilles, en traversant tous les nœuds où la donnée est contenue. Le score d'anomalie de HST repose sur le calcul du profil de masse, qui fait référence au nombre de points ou d'instances observées dans un demi-espace donné. Plus la masse est faible, plus la probabilité d'anomalie est élevée.

Le flux de données est ensuite divisé en courtes fenêtres d'une longueur déterminée de données. À tout moment, nous ne considérons que deux fenêtres consécutives, la fenêtre actuelle, appelée fenêtre la plus récente, et la précédente, qui est appelée fenêtre de référence. Une fois que les données ont été enregistrées dans la fenêtre la plus récente, on considère qu'elle est pleine et elle devient la fenêtre de référence. Lorsque la donnée suivante arrive, une nouvelle dernière fenêtre commence. Le contenu d'un flux de données est utilisé pour calculer et mettre à jour le profil de masse de chaque nœud traversé par les instances du flux, où chaque nœud possède deux compteurs, r pour le profil de masse de référence et l pour le dernier profil de masse.

Un avantage clé de l'algorithme est que la phase d'apprentissage a une complexité et une consommation mémoire constantes. De plus, comparées aux autres méthodes phares comme les arbres de Hoeffding, HST présente de bonnes performances en termes de précision de détection.

4.4.3 Différences majeures entre HST et IForest ASD

HST et IForest ASD ont été proposées pour détecter les anomalies dans les flux de données ; elles se distinguent sur plusieurs points : les phases d'apprentissage, de test et de mise à jour du modèle.

1. **La phase d'apprentissage** : avant la construction des arbres, alors que

IForest ASD choisit dans le jeu de données original un échantillon de ψ données sans remplacement pour chaque arbre prévu pour la forêt, HST crée un espace de données fictif. En effet, pour chaque dimension de l'ensemble de données original, HST crée une valeur minimale et maximale :

$$min = sq - 2 \times max(sq, 1 - sq);$$

$$max = sq + 2 \times max(sq, 1 - sq).$$

sq étant une valeur aléatoire dans $[0, 1]$.

IForest ASD a besoin d'un échantillon de l'ensemble de données original pour construire l'ensemble, tandis que HST peut construire des structures d'arbres sans aucune donnée. Pendant la construction des arbres, pour diviser un nœud, alors que IForest ASD choisit une valeur de division aléatoire v entre min_d et max_d. Pour HST, v est la moyenne des valeurs (fictives) du nœud soit $v = (min\_d + max\_d)/2$. Notez que la taille maximale d'un arbre (max_depth) avec IForest ASD est fonction de la taille de l'échantillon ψ alors qu'elle est un paramètre défini par l'utilisateur pour HST. Par conséquent, pour construire les arbres de la forêt, HST n'a aucune connaissance de l'ensemble de données à utiliser, si ce n'est le nombre d'attributs. Alors que IForest ASD dépend étroitement de l'ensemble de données.

2. **Calcul du score** : pour calculer le score d'une donnée, IForest ASD se base sur la longueur du chemin de cette donnée dans chaque arbre (comme décrit dans la Section 4.4.1), alors que HST se base sur le score des données pour chaque arbre. Le score global des données est donc, pour IForest ASD, basé sur la moyenne du chemin parcouru dans tous les arbres de la forêt, alors qu'il est donné par la somme des scores qui ont été obtenus au niveau de chaque arbre de la forêt pour HST. Contrairement à IForest ASD, qui normalise la longueur moyenne avec $C(n)$ lors du calcul du score, HST limite le nombre d'instances qu'un nœud peut avoir dans un arbre, c'est-à-dire $sizeLimit$. Il s'agit d'un paramètre qui est prédéfini par l'utilisateur.
3. **Approche de la gestion du drift** : le concept drift consiste en un change-

ment global, considéré comme normal, des valeurs des observations. Les deux méthodes mettent à jour le modèle de base (la forêt) afin de gérer le concept drift. Cependant, alors que IForest ASD met à jour son modèle à condition que le taux d'anomalies dans une fenêtre soit supérieur au paramètre u prédéfini, HST met automatiquement à jour son modèle pour chaque nouvelle fenêtre entrante.

4. **Politiques de mise à jour du modèle** : HST met à jour le modèle à la fin de chaque fenêtre en réinitialisant, à 0, la valeur de masse de chaque nœud de chaque arbre. Cependant, IForest ASD met à jour le modèle lorsque le taux d'anomalies dans la fenêtre est supérieur à un paramètre prédéfini u en réentraînant un nouveau modèle de IForest sur la dernière fenêtre.

Pour récapituler, bien que IForest ASD et HST soient tous deux basés sur le principe d'isolation, ils présentent de nombreuses différences majeures. Afin de comparer les performances de ces deux méthodes de détection d'anomalies, nous avons réalisé, dans les sections suivantes, plusieurs expériences comparatives sur différents jeux de données.

4.5 Expérimentations et discussions

Dans cette section, sont présentées la méthodologie suivie dans nos tests et une discussion des résultats obtenus. Une étude comparative suivie d'une discussion sont effectuées sur l'impact de la taille de la fenêtre de données et du nombre d'estimateurs (nombre d'arbres) sur les performances de Half-Space Tree et IForest ASD en termes de score F_1, de temps d'entraînement et de test, et de taille du modèle.

Les performances sont mesurées selon la stratégie appelée "prequential evaluation" (méthode test-then-train) conçue spécifiquement pour les environnements de flux de données. Il est à noter que chaque échantillon sert à deux fins (test puis train) [5]. Les échantillons sont analysés séquentiellement, dans l'ordre d'arrivée, et deviennent immédiatement inaccessibles. Cette approche garantit

5. https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.evaluation.EvaluatePrequential.html

Tableau 4.2 – Jeux de données et hyper-paramètres.

	Shuttle	Forest-cover	SMTP
Nombre de dimensions	9	10	3
Pourcentage d'anomalies (u - Drift Rate)	7.15%	0.96%	0.03%
Taille du jeu de données	49,097	286,048	95,156
w - Tailles de fenêtre testées	[50 , 100, 500 , 1000]		
t - Nombre d'arbres	[30, 50, 100]		

que le modèle soit testé sur de nouveaux échantillons qui n'ont pas encore été utilisés lors de l'apprentissage. Tous les résultats et les mesures de performance rapportés dans ce chapitre correspondent à la sortie de scikit-multiflow du "prequential evaluator".

4.5.1 Jeux de données, paramètres d'expérimentation et métriques

Jeux de données : Pour prouver l'efficacité de notre implémentation d'IForest ASD, nous l'avons comparée à la méthode Half-Space Trees sur une variété de configurations (12 pour chaque jeu de données). Nous avons utilisé quelques jeux de données publics bien connus (Dua et Graff, 2017) qui ont été utilisés dans l'article sur IForest ASD (Ding et Fei, 2013). Leurs caractéristiques sont résumées dans le tableau 4.2.

Paramètres d'expérimentation : Le réglage des paramètres que nous avons utilisés pour réaliser diverses expériences visant à comparer les deux modèles HSTrees et IForest ASD est présenté dans le tableau 4.2.

Le seuil d'anomalie a été fixé à 0.5 et le nombre d'observations (échantillons) a été limité à 10 000 pour les deux ensembles de données pour chaque expérience. Pour le paramètre u, nous avons fixé la valeur au pourcentage d'anomalies dans les jeux de données (tableau 4.2) comme l'ont fait les auteurs de l'algorithme IForest ASD (Ding et Fei, 2013). Cette valeur est utilisée pour détecter le concept drift et réinitialiser le modèle de détection d'anomalies, comme décrit dans la figure 4.5.

Le code source permettant de reproduire les expériences se trouve sur Github à l'adresse suivante : `https://github.com/MariamBARRY/skmultiflow_IForestASD`.

Métriques : Dans le tableau des résultats, nous avons indiqué 3 métriques : F1, le ratio Temps d'exécution (apprentissage et test) correspondant à (IForest ASD / HST) et le ratio Taille du modèle (HST / IForest ASD).

Ratio du temps d'exécution (ratio du coefficient IForest ASD) - IRa : Puisque le modèle Half-Space Trees (HST) est toujours plus rapide que le modèle IForest ASD (noté IFA dans cette section pour raison d'espace), tant pour le temps d'apprentissage que pour le temps de test, d'un ordre de 400 pour le pire des cas, nous avons rapporté le ratio entre le temps de fonctionnement (apprentissage et test) des deux modèles : IFA/HST. Par exemple, pour les données de Forest Cover, l'IFA est 3 fois plus lent que le modèle HST pour $W = 50, T \in \{30, 50, 100\}$. Pour $W = 1000, T = 100$ le rapport des temps d'exécution est de 391 dans le tableau 4.3, ce qui signifie que l'IFA est 391 fois plus lent que le modèle HST. Les valeurs absolues du temps d'exécution (en secondes) et leur évolution en fonction de la variation des paramètres sont indiquées dans la figure 4.7.

Ratio de taille du modèle (ratio de coefficient HST) - HRa : A l'inverse du temps d'exécution, lorsque nous considérons la taille du modèle, nous observons que l'IFA utilise toujours moins de mémoire que HST. Ainsi, pour comparer ces deux méthodes, nous calculons le rapport de la valeur la plus élevée (HST) sur la valeur la plus faible (IFA) pour obtenir le coefficient HST. La figure 4.6. rapporte l'impact du choix des paramètres sur les ressources utilisées (taille du modèle en valeur absolue en Kilo-bytes) pour les deux arbres HST et l'évolution des ressources utilisées pour les 3 jeux de données lorsque la taille de la fenêtre et le nombre d'arbres varient. Les résultats ainsi que les interprétations sont présentées ci-dessous.

Tableau 4.3 – Comparaison entre HS-Tress(HST) et IForest ASD (IFA) - Nous avons fixé la taille de la fenêtre w (50, 100, 50, 1000) et varié le nombre d'arbres t pour chaque jeu de données Forest Cover, Shuttle et SMTP (36 configurations) - HRa and IRa représentent respectivement HSTRees ratio et IFA ratio décrits dans la Section 5.4.1.

		Forest cover				Shuttle			
		F1		*Time*	*Size*	*F1*		*Time*	*Size*
w	*t*	**HST**	**IFA**	**IRa**	**HRa**	**HST**	**IFA**	**IRa**	**HRa**
50	30	**0.36**	0.22	**3**	702	0.13	**0.64**	**3**	817
50	50	**0.36**	0.23	**3**	694	0.13	**0.64**	**3**	854
50	100	**0.36**	0.22	**3**	737	0.13	**0.64**	**3**	950
100	30	**0.39**	0.30	12	367	0.14	**0.71**	12	461
100	50	**0.39**	0.29	12	404	0.13	**0.72**	14	505
100	100	**0.39**	0.30	12	416	0.13	**0.71**	16	551
500	30	0.39	**0.49**	158	108	0.14	**0.80**	519	28
500	50	0.39	**0.47**	152	129	0.13	**0.80**	393	36
500	100	0.39	**0.47**	156	140	0.15	**0.80**	363	92
1000	30	**0.54**	0.40	372	**63**	0.17	**0.78**	1757	**21**
1000	50	**0.54**	0.40	387	**74**	0.14	**0.78**	1175	**28**
1000	100	**0.55**	0.40	391	**86**	0.14	**0.77**	1678	**23**

		SMTP			
		F1		*Time*	*Size*
w	*t*	**HST**	**IFA**	**IRa**	**HRa**
50	30	0	**0.34**	**3**	926
50	50	0	**0.34**	**3**	940
50	100	0	**0.34**	**3**	986
100	30	0	**0.36**	9	596
100	50	0	**0.36**	9	734
100	100	0	**0.36**	9	808
500	30	0	**0.39**	106	288
500	50	0	**0.40**	111	372
500	100	0	**0.40**	111	434
1000	30	0	**0.39**	264	**127**
1000	50	0	**0.39**	272	**155**
1000	100	0	**0.39**	272	**176**

4.5.2 Résultats et analyses

Le tableau 4.3 synthétise les résultats obtenus pour chaque combinaison de paramètres (taille de la fenêtre et nombre d'arbres) pour les trois (03) jeux de données.

Nous avons observé que, sur la base du score F1, IForest ASD est plus performant que HST pour les deux (02) jeux de données Shuttle et SMTP, quelle que soit la taille des fenêtres ou le nombre d'arbres. Trois autres points sont observés :

Premièrement, nous avons remarqué que le nombre d'arbres n'a pas d'impact significatif sur les performances de prédiction pour tous les jeux de données. Cependant, lorsque la taille de la fenêtre augmente (variant de 50, 100, 500 à 1000), le résultat est meilleur car le score F1 augmente pour tous les jeux de données.

Deuxièmement, alors que pour HST, le score F1 s'améliore pour une taille de fenêtre >= 500, il diminue pour IForest ASD. Cela peut s'expliquer par le fait que contrairement à HST, IForest ASD supprime son modèle actuel de détection d'anomalies (Isolation Forest) et en construit un autre en utilisant toutes les données de la fenêtre actuelle lorsque le taux d'anomalies dans la fenêtre est supérieur à u.

Troisièmement, il existe une grande différence entre les performances des deux modèles en fonction de l'ensemble de données. En particulier pour Shuttle qui présente un taux d'anomalies plus élevé de 7%, IForest ASD dépasse HST avec 80% de F1. Nous supposons donc qu'IForest ASD est plus performant sur les ensembles de données présentant un taux d'anomalies relativement élevé.

En plus de la performance prédictive, on note deux notions importantes dans les applications de flux de données : le temps et l'utilisation de la mémoire des modèles. Nous analysons ici l'impact des paramètres (taille des fenêtres w et nombre d'arbres t) sur la quantité de ressources utilisées par chaque modèle et nous les comparons.

Comparaison de la consommation mémoire des modèles : Pour ce qui concerne l'évolution de la taille du modèle à partir de la figure 4.6, où la taille des deux

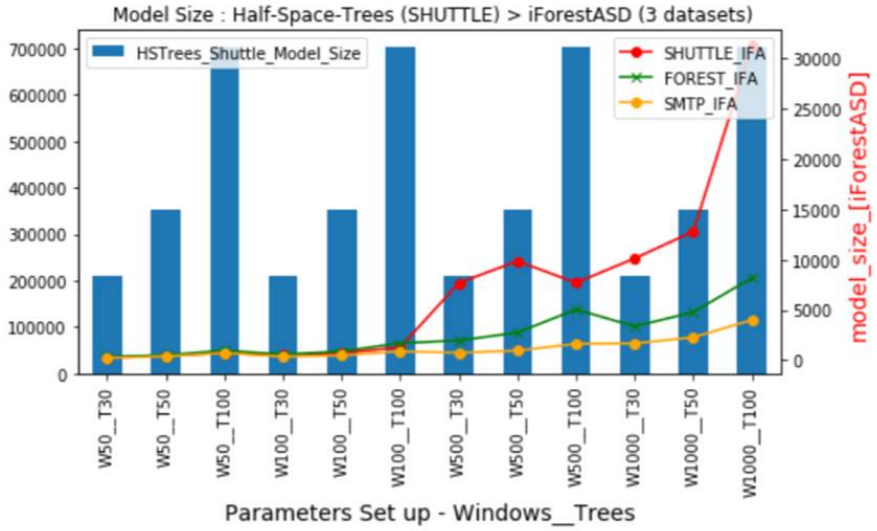


FIGURE 4.6 – Mesure de la taille du modèle (Kilo-octets) : Nous avons fixé la taille de la fenêtre (50, 100, 500, 100) et varié le nombre d'arbres (30, 50, 100). Pour chaque paramètre, nous avons exécuté HSTrees (barplot - axe Y à gauche) avec Shuttle et IForest ASD (Lignes - axe Y à droite) pour chaque jeu de données - 36 mesures - et avons tracé les lignes (Axe Y à droite). HST est s $\approx 20x$ plus grand que IFA

modèles est représentée sur 2 axes Y (IForest ASD à droite), nous pouvons souligner 3 points :

— IForest ASD a utilisé moins de mémoire que HST (environ 20 fois moins). Ceci s'explique par le fait qu'avec IForest ASD, la politique de mise à jour consiste à supprimer complètement l'ancien modèle lorsque le taux d'anomalies dans la fenêtre glissante est supérieur à u (mise à jour par remplacement). Par contre, HST met continuellement à jour le modèle à chaque observation.

— Pour HST, la taille de la fenêtre w n'a aucun impact sur la taille du modèle, seul le nombre d'arbres t augmente la mémoire utilisée (diagrammes à barres). Ceci est cohérent avec la politique de mise à jour de HST qui consiste à mettre à jour les statistiques dans les nœuds d'arbres avec une taille d'ensemble fixe.

— Pour IForest ASD, la taille de la fenêtre w et le nombre d'arbres t ont tous un impact positif sur la taille du modèle, pour les trois ensembles de

données (dans le graphique en 3 lignes). Cela est dû au fait qu'IForest ASD utilise toutes les instances de la fenêtre (w) pour construire l'ensemble avec t arbres.

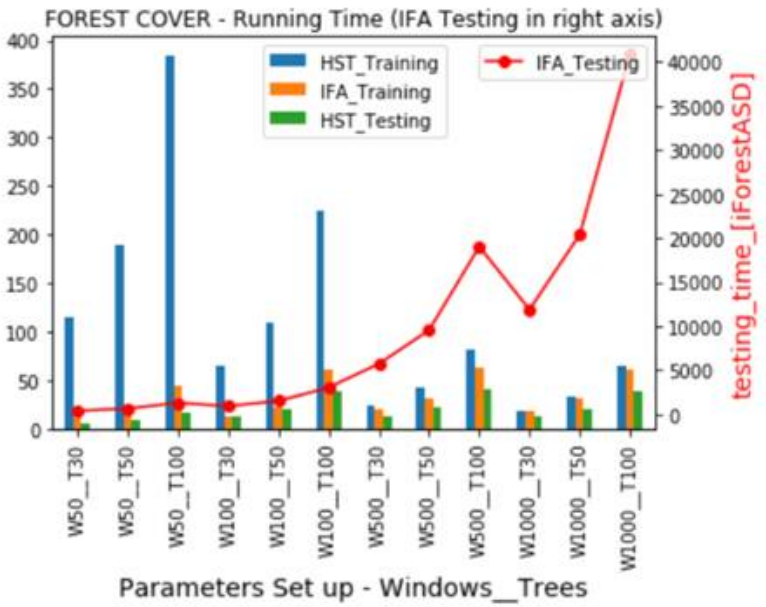


FIGURE 4.7 – Mesure du temps d'exécution (en secondes) : Jeu de données Forest-Cover - nous avons fixé la taille de la fenêtre (50, 100, 500, 100) et varié le nombre d'arbres (30, 50, 100) pour comparer le temps d'exécution de HST et IForest ASD. Le temps de test de IForest ASD est représenté dans une autre échelle sur l'axe Y de droite et tracé en ligne rouge.

Lorsqu'il s'agit de certaines applications de streaming critiques, comme un modèle prédictif d'intrusion dans la sécurité d'un réseau, un modèle rapide mais moins précis est souvent préféré à un modèle lent et plus précis. En effet, il est préférable d'avoir un taux élevé de fausses alertes (faux positifs) à étudier que de manquer des vraies anomalies critiques (vrais positifs = attaque). Par conséquent, nous cherchons à analyser ci-dessous le comportement des modèles en termes de temps d'entraînement et de test.

Nous pouvons observer sur la figure 4.7 que IForest ASD est plus rapide avec une petite taille de fenêtre alors que Half-Space Trees est plus rapide avec une plus grande taille de fenêtre. Le nombre d'arbres augmente le temps d'exécution de chacun d'entre eux. Ce résultat est cohérent avec la complexité temporelle des

deux estimateurs de base : iTree et Half-Space Tree. En effet, dans le pire des cas, la complexité temporelle de l'apprentissage d'un modèle IForest est de $O(t\psi_1^2)$ (FT Liu et al., 2012) tandis que pour un modèle HST, la complexité temporelle moyenne est de $O(t(h+\psi_2))$ dans le pire des cas (Ding et Fei, 2013), où t est le nombre d'arbres dans l'ensemble, ψ_1 la taille d'un échantillon de IForest, ψ_2 la taille de la fenêtre dans HST et h la profondeur maximale d'un arbre.

De plus, le temps de test d'IForest ASD – sur l'axe de droite en ligne rouge – peut être 100 fois plus long que le temps de test de HST (40,000 contre 400), ce qui est une contrainte pour les applications critiques qui ont besoin de noter les anomalies rapidement. La différence est significative entre les deux modèles car dans IForest ASD, chaque instance de la fenêtre est testée dans chacun des arbres d'isolation pour calculer le score d'anomalies.

Nos résultats soulignent qu'un compromis doit être fait entre la consommation de ressources, le temps de traitement et la performance prédictive (le score F1 pour les anomalies) pour obtenir des résultats acceptables. Dans le contexte de la détection d'anomalies (performance prédictive), nous pouvons conclure qu'en termes de score F1, IForest ASD est meilleur que HST. Cependant, dans le contexte des applications de streaming de données, le temps d'exécution est un facteur important et les modèles rapides sont préférés. Dans ce sens, HST est une bonne option en raison de son temps de traitement plus faible, contrairement à IForest ASD qui est exponentiel par rapport à la taille des fenêtres.

Guide de configuration en fonction des besoins applicatifs Le nombre d'expériences (36 configurations : 3 estimateurs x 4 tailles de fenêtres x 3 jeux de données) utilisées pour obtenir les résultats et les discussions précédents donne un aperçu des bonnes grilles de paramètres et du modèle à utiliser dans tel ou tel contexte.

Comme les performances des modèles peuvent varier considérablement d'un choix à l'autre, en fonction des caractéristiques des données et du réglage des paramètres, nous fournissons quelques lignes directrices pour décider entre ces méthodes de détection d'anomalies. Ainsi, sur la base de nos expériences, en fonction des exigences spécifiques des applications et des contraintes de ressources (temps d'exécution et taille du modèle), on peut noter :

— **Fenêtre w pour F1** : Si la priorité est la performance F1 pour la détection d'anomalies, alors fixer w près de 500 donne de meilleurs résultats avec IForest ASD.
— **Temps et Mémoire** : Si un modèle rapide et surtout un temps de calcul rapide sont nécessaires, HST devrait être l'option privilégiée car il reste la référence parmi les modèles de streaming rapide.
— **IForest ASD configuration** : Lorsque le modèle rapide n'est pas une exigence prioritaire, l'IFA donne de meilleurs résultats avec un temps d'exécution faible lorsqu'on utilise des fenêtres plus petites (inférieures à 100) et une taille de modèle faible avec un nombre d'estimateurs t compris entre 30 et 50.
— **HST configuration** : Lorsqu'un modèle rapide est requis, HST avec $w = 1000$ et un nombre d'estimateurs $t = 30$, conduit à de meilleurs résultats avec un coût de ressources optimal.

Nous avons vu que le meilleur modèle peut être soit HST soit IForest ASD en fonction du jeu de données et des exigences de l'application. Nous recommandons donc de tester les modèles avec un échantillon de jeu de données en utilisant le framework scikit-multiflow pour identifier les meilleurs paramètres pour chaque application.

4.6 Discussion

Nous avons présenté une vue d'ensemble structurée et complète des algorithmes de détection d'anomalies pour les flux de données et nous les avons catégorisés par approches basées sur les anomalies : statistiques, plus proches voisins, clustering et isolation. Pour chaque catégorie, nous avons mis en évidence les avantages et les inconvénients des différentes approches. D'après cette étude, IForest semble être une méthode précise mais non adaptée aux flux de données (Togbe et al., 2020). Nous avons donc proposé une implémentation d'une approche de détection d'anomalies basée sur IForest pour les flux de données utilisant une fenêtre glissante (algorithme IForest ASD) dans Scikit-multiflow. Scikit-multiflow est un framework d'apprentissage automatique pour les flux de données et multi-sorties/multi-labels. La motivation derrière ce travail est

qu'il existe très peu de méthodes de détection d'anomalies dans les framework de streaming examinés, et la contribution sur scikit-multiflow peut aider les entreprises, les chercheurs et la communauté python croissante à exploiter les méthodes et à collaborer pour le domaine difficile de la détection d'anomalies dans le contexte du streaming.

La méthode IForest ASD nécessite un paramètre pour la détection du drift qui est fixé manuellement. En perspective, nous pouvons nous concentrer sur l'optimisation de l'approche pour qu'elle soit plus efficace pour la détection du concept drift dans les flux de données. Nous pouvons utiliser des méthodes existantes dans scikit-multiflow telles que ADaptive WINdowing (ADWIN)(Albert Bifet et Gavalda, 2007) pour adapter automatiquement la taille de la fenêtre coulissante. Une amélioration majeure de ce travail serait de mettre à jour correctement le modèle de détection d'anomalies en prenant en compte les détecteurs d'anomalies et les observations précédentes au lieu de les écarter complètement. Cela peut être fait en utilisant des approches d'apprentissage adaptatif. Dans le chapitre suivant, nous approfondissons cette idée.

Nous tenons à remercier Mme Mariam Barry, M. Jacob Montiel, M. Vinh-Thuy Tran et M. Albert BIFET pour leurs participations active dans les travaux décrits dans ce chapitre ainsi que pour leurs discussions enrichissantes.

Chapitre 5

Détection d'anomalies : concept drift et IForest

Sommaire

Nos travaux présentés dans ce chapitre ont fait l'objet de la publication suivante :

Maurras Ulbricht TOGBE, Yousra Chabchoub, Aliou Boly, Mariam Barry, Raja Chiky, Maroua Bahri. Anomalies Detection Using Isolation in Concept-Drifting Data Streams. Computers 2021, 10, 13. https://doi.org/10.3390/computers10010013

5.1 Introduction

La fouille dans les flux de données est confrontée à différentes contraintes. La gestion efficace du concept drift constitue un véritable défi pour les algorithmes de fouille de flux de données en particulier pour la détection d'anomalies. Parmi les méthodes de détection d'anomalies basées sur l'isolation, peu sont adaptées aux contraintes des flux de données. IForest ASD est une adaptation de IForest au contexte du streaming. Dans le chapitre précédent, nous avons proposé une implémentation de IForest ASD dans le framework scikit-multiflow et avons validé notre implémentation avec différentes expérimentations. Pour une meilleure prise en compte du concept drift, nous approfondissons nos travaux sur IForest ASD. Dans ce chapitre, nous présentons dans un premier temps quelques généralités sur le concept drift (section 5.2). Dans la section 5.3, nous identifions la faiblesse majeure de la gestion du concept drift proposée dans la méthode IForest ASD et proposons différentes améliorations en particulier pour la méthode de détection de drift. Pour évaluer nos propositions, nous effectuons des expérimentations et analysons les résultats obtenus à la section 5.4. La section 5.5 sera consacrée à une discussion sur les travaux effectués dans ce chapitre.

5.2 Concept drift

5.2.1 Définition et types de concept drift

Définition Le concept drift est un changement dans le temps de la distribution des données (Hoens et al., 2012). La prise en compte de ce changement représente l'un des plus grands défis de l'analyse des flux de données (Joao Gama, 2010).Le concept drift a un impact considérable sur les performances des méthodes de traitement de flux de données. Puisque la distribution des données a changé, le modèle pourrait devenir obsolète. Il est donc important que toutes les méthodes d'analyse de flux de données et dans notre cas particulier, toutes les méthodes de détection d'anomalies dans les flux de données prennent en compte la gestion du concept drift.

Il existe différents types de concept drift que les méthodes doivent considérer notamment pour la mise à jour du modèle.

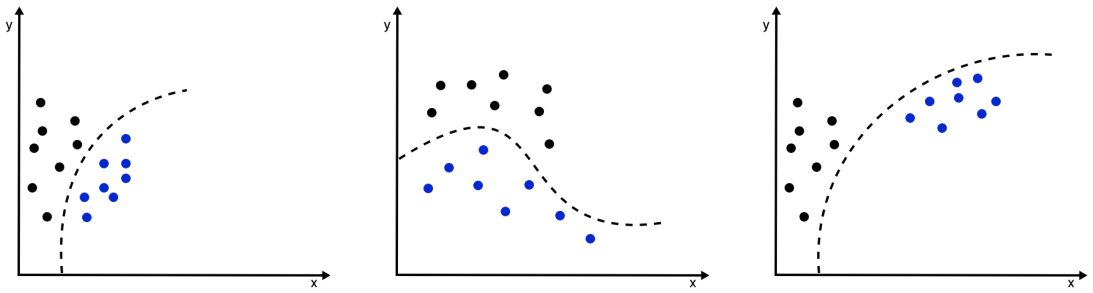
Types de concept drift : Les flux de données sont continus avec une distribution des données non stationnaire c'est-à-dire que ses propriétés statistiques (espérance, variance, auto-corrélation) peuvent varier dans le temps. En fonction du type de changement de la distribution des données, on peut distinguer deux types de concept drift : real concept drift illustré à la figure 5.1b et virtual concept drift illustré à la figure 5.1c. Comme on peut le constater sur la figure 5.1, le changement de la distribution dans le temps peut rendre obsolète le modèle de classification. Généralement, seul le real concept drift change les frontières de la classe et rend donc obsolète le modèle précédent. Un virtual concept drift peut aussi changer le modèle en cas de présence d'une nouvelle classe (nouveautés).
Puisque le changement se fait dans le temps, la vitesse d'apparition du nouveau concept permet de distinguer quatre types de changements :

- Reoccuring drift soit dérive récurrente (figure 5.2a) : Ce type de dérive traduit le fait que l'ancienne distribution réapparaît au bout d'un certain temps. Il est donc important dans ce cas-ci de ne pas supprimer l'ancien modèle car il pourrait être utilisable plus tard. Une dérive récurrente peut être cyclique. C'est le cas des prix des vols qui augmentent chaque année pendant les périodes estivales puis retrouvent leurs valeurs normales. La dérive peut également être acyclique.
- Gradual drift soit dérive graduelle (figure 5.2b) : L'ancienne distribution disparaît decrescendo dans le temps, laissant place à une nouvelle distribution. Il s'agit d'un véritable défi pour les méthodes de détection de concept drift car il faut maintenir plusieurs fenêtres avec differentes échelles temporelles pour identifier ce type de drift. En effet, la nouvelle distribution ne devient stable qu'après un certain temps. C'est le cas par exemple du changement de température entre l'été et l'hiver.
- Incremental drift soit dérive incrémentale (figure 5.2c) : Consiste à un changement incrémental de l'ancienne distribution par la nouvelle. C'est l'exemple d'un capteur de température dans une pièce suite au démarrage

du chauffage. Les températures relevées dans la pièce augmenteront petit à petit jusqu'à stabilisation de la température de la pièce.

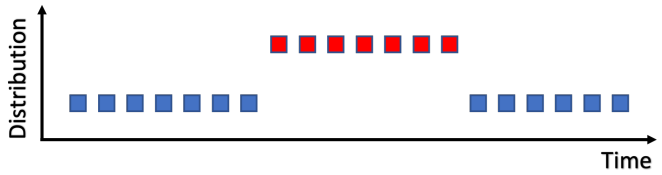
— Sudden drift soit dérive soudaine (figure 5.2d) : On parle de dérive soudaine quand en un court instant, presque brutalement, une nouvelle distribution remplace une autre. C'est le cas par exemple du prix du pain qui du jour au lendemain augmente ou diminue.

Des études plus approfondies sur les différents types de drift sont notées dans (João Gama, Žliobaitė et al., 2014; Lu et al., 2018; Agrahari et Singh, 2021). Elles présentent une revue détaillée sur le concept drift, ses défis pour l'analyse des flux de données et différentes méthodes pour le détecter. João Gama, Žliobaitė et al. présentent une revue des méthodes de détection de concept drift avec une classification en fonction des types de concept drift (João Gama, Žliobaitė et al., 2014). Quand à Agrahari et Singh, ils proposent dans (Agrahari et Singh, 2021) une étude complète du concept drift avec une classification des méthodes de détection de drift suivant plusieurs critères. Ainsi, ils présentent quelques méthodes basées sur la similarité et la dissimilarité, des méthodes statistiques de détection de drift, des méthodes basées sur le fenêtrage, etc. Cette dernière catégorie de méthodes basées sur le système de fenêtrage nous intéresse particulièrement dans ce chapitre pour la détection d'anomalies dans les flux de données tenant compte du concept drift. Le système de fenêtrage est bien adapté au contexte des flux comme ils sont continus et illimités.

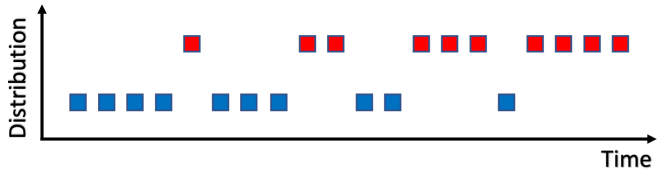


(a) Original data. (b) Real drift. (c) Virtual drift.

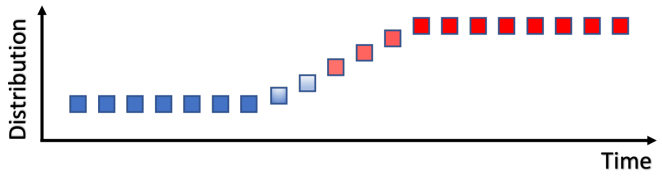
Figure 5.1 – Types de concept drift.



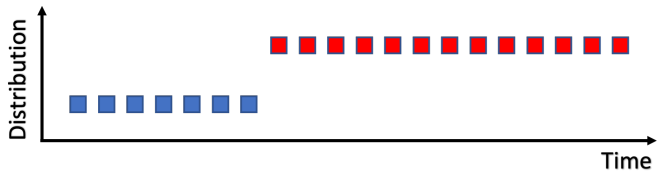
(a) Reoccurring drift : l'ancienne distribution réapparaît au bout d'un certain temps.



(b) Gradual drift : La nouvelle distribution remplace l'ancienne graduellement.



(c) Incremental drift : La nouvelle distribution remplace l'ancienne de façon incrémentale.



(d) Sudden drift : La nouvelle distribution apparaît soudainement et remplace l'ancienne distribution en un court instant.

FIGURE 5.2 – Types de concept drift.

5.3 Isolation Forest et la détection de drift

5.3.1 Détection de drift avec IForest ASD

IForest ASD (DING et FEI, 2013) (également appelé IFA dans le reste du chapitre) est une adaptation d'IForest aux flux de données en ajoutant simplement

une approche de fenêtrage (application d'IForest à une fenêtre de données). Les auteurs ont également ajouté une phase de détection de drift. Comme le flux évolue, la distribution sous-jacente peut changer au fil du temps. Par conséquent, le modèle actuellement construit peut ne plus être représentatif des prochaines instances entrantes, ce qui peut avoir un impact sur la qualité de la classification. Afin de faire face à ce problème, les algorithmes de flux sont généralement couplés à un mécanisme de détection de drift afin de traiter les nouvelles tendances dès qu'elles apparaissent.

Afin de détecter le concept drift, IForest ASD maintient un paramètre u qui correspond au taux d'anomalies qui ne doit pas être dépassé dans les fenêtres. Ce paramètre est donné en entrée de l'algorithme. Ainsi, lorsque dans une fenêtre, le taux d'anomalies détecté par le modèle est supérieur à u, IForest ASD met à jour son modèle avec les données de la fenêtre courante, tout en supposant qu'une dérive des données a eu lieu. Deux scénarios sont possibles : l'utilisateur donne une valeur de u inférieure à la réalité, dans ce cas le modèle sera mis à jour à chaque fenêtre. La seconde possibilité est de donner une valeur de u supérieure à la réalité ce qui implique que le modèle ne sera jamais mis à jour puisque la dérive ne sera jamais détectée même si elle s'est produite. Ainsi, u est un paramètre crucial qui doit être correctement calibré afin de détecter la dérive dans le flux de données. Dans un mode non supervisé où il n'y a aucune information ou connaissance a priori sur les données, le choix de u devient problématique, et la capacité d'IForest ASD à détecter la dérive en dépend. C'est pourquoi, dans ce chapitre, nous étudions une amélioration de IForest ASD en proposant deux approches basées sur une détection de dérive qui peut se faire à deux niveaux : détection de dérive de modèle ADWIN (ADaptive WINdowing) (Albert Bifet et Gavalda, 2007) ou détection de dérive de données KSWIN (Kosmolgorov Simirnov WINdows) (Raab et al., 2020).

5.3.2 IForest ASD basé sur ADWIN

Plusieurs méthodes de détection de drift ont été proposées dans la littérature (João Gama, Žliobaitė et al., 2014). Par exemple, ADWIN (Albert Bifet et Gavalda, 2007), qui est un détecteur et estimateur de changement bien connu,

consiste à maintenir une fenêtre de longueur variable W qui conserve les instances les plus récentes du flux avec la contrainte que la fenêtre ait la longueur maximale statistiquement cohérente avec l'hypothèse "il n'y a pas eu de changement dans la valeur moyenne à l'intérieur de la fenêtre" (Albert Bifet et Gavalda, 2007). Techniquement, ADWIN compare la différence moyenne de deux sous-fenêtres quelconques, W_1 d'instances anciennes et W_2 d'instances récentes, par rapport à W. Si la différence est statistiquement significative, alors ADWIN efface toutes les instances de W_1, qui représentent un ancien concept, et conserve W_2.

Dans ce qui suit, nous utilisons ADWIN, dans une première tentative, pour détecter les changements dans la distribution. ADWIN a fait ses preuves dans un large éventail d'algorithmes de flux, tels que adaptive Hoeffding trees (Albert Bifet et Gavaldà, 2009) et les méthodes de l'ensemble learning (Gomes, Barddal et al., 2017).

ADWIN fournit deux méthodes pour détecter la dérive. Premièrement, chaque instance est ajoutée à ADWIN en utilisant la fonction *add_element()*. Cette fonction prend une valeur numérique unidimensionnelle (int ou float) et l'ajoute à la fenêtre adaptative. La deuxième fonction *detected_change()* effectue la détection de la dérive expliquée ci-dessus dans la fenêtre adaptative actuelle et renvoie une valeur booléenne qui indique si la dérive a été détectée ou non. On peut noter Albert Bifet et Gavalda (2007) et Montiel et al. (2018) pour plus d'informations concernant l'implémentation d'ADWIN.

Dans le modèle IForest, deux résultats sont renvoyés : le score des données et la prédiction du modèle (0 ou 1), les deux peuvent être utilisés comme valeurs d'entrée pour ADWIN. La figure 5.3 présente les améliorations de IForest ASD basées sur ADWIN que nous proposons pour la détection des dérives. La phase de détection de dérive est gérée par ADWIN. Deux améliorations ont été proposées : une basée sur la prédiction du modèle, appelée PADWIN IFA (Section 5.3.2.2), et l'autre basée sur le score des données qui est issu du modèle, appelé SADWIN IFA (Section 5.3.2.1).

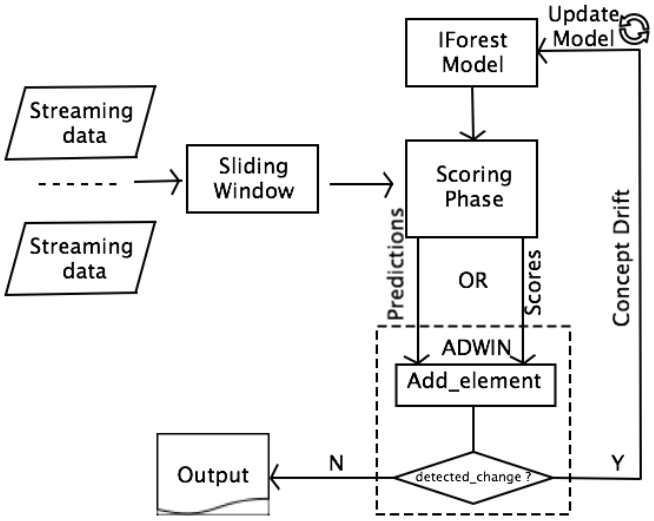


FIGURE 5.3 – Workflow de la méthode IForest ASD basée sur ADWIN : PADWIN IFA si les prédictions sont utilisées, SADWIN IFA si les scores sont considérés. Figure inspirée de DING et FEI (2013).

5.3.2.1 SADWIN IForest ASD

La prédiction d'IForest est basée sur un score calculé pour chaque donnée. Ainsi, le score est plus représentatif des données réelles que leur classe prédictive car la prédiction dépend d'un seuil prédéfini. Par conséquent, nous proposons d'utiliser le score comme une entrée de la fonction *add_element()* d'ADWIN. Si ADWIN détecte la dérive, alors nous mettons à jour le modèle. Sinon, nous traitons les fenêtres suivantes en utilisant le modèle IForest actuel. Nous appelons cette nouvelle version SADWIN IFA.

5.3.2.2 PADWIN IForest ASD

L'idée derrière cette amélioration est qu'en cas de drift des données, la prédiction dérivera également. C'est pourquoi nous décidons d'utiliser la prédiction qui est donnée par IForest comme entrée à ADWIN. Dans la version PADWIN IFA, la détection de la dérive est basée sur les prédictions du modèle (1 ou 0) qui sont données à la fonction *add_element()*. Si ADWIN détecte la dérive, nous mettons à jour le modèle. Sinon, nous traitons les fenêtres suivantes en utilisant le modèle existant.

Comme expliqué ci-dessus, toutes les améliorations basées sur ADWIN détectent la dérive du modèle après le calcul de la détection d'anomalies. Cela signifie que la fenêtre précédente est perdue, car toutes les prédictions sont probablement fausses. Pour résoudre ce problème, nous proposons une autre amélioration qui détecte la dérive et met à jour le modèle avant de détecter les anomalies. Comme ADWIN est conçue pour évaluer la qualité d'un modèle, elle ne peut pas être appliquée directement sur des données brutes. En 2020, une nouvelle méthode de détection de drift des données, appelée Kolmogorov–Smirnov WINdowing (KSWIN) (Raab et al., 2020), a été proposée dans la littérature et a été implémentée dans le framework scikit-multiflow.

5.3.3 IForest ASD basé sur KSWIN

5.3.3.1 KSWIN et NDKSWIN IForest ASD

KSWIN (Raab et al., 2020) est une méthode récente de détection de concept drift qui repose sur le test statistique bien connu de Kolmogorov–Smirnov (KS) (Massey Jr, 1951 ; Lopes, 2011). Le test de KS est un test non paramétrique qui ne nécessite aucune hypothèse sur la distribution sous-jacente des données. Le test de KS ne peut traiter que des données unidimensionnelles. Il compare la distance $dist(W, R)$ absolue entre deux distributions cumulatives empiriques.

Comme ADWIN, KSWIN implémente également deux fonctions : *add_element()* pour ajouter les nouvelles données unidimensionnelles dans la fenêtre glissante locale Ψ. La seconde fonction *detected_change()*, indique si la dérive est détectée ou non. Ψ ayant une taille fixe, avant d'ajouter de nouvelles données dans Ψ, les anciennes données sont supprimées et les nouvelles sont ajoutées à la queue de la fenêtre glissante. La figure 5.4 décrit ce processus.

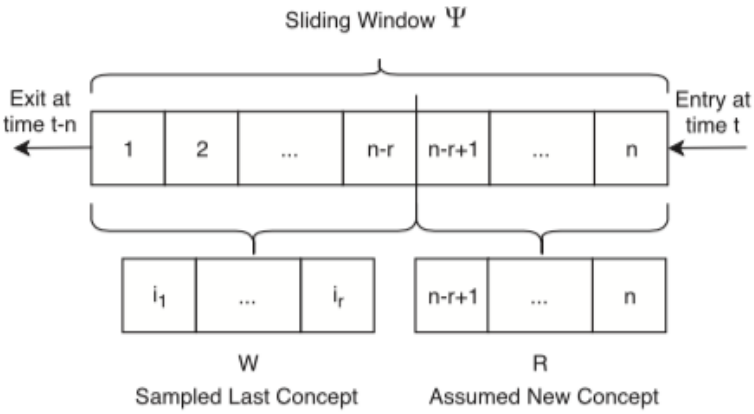


FIGURE 5.4 – La stratégie de mémoire utilisée par RAAB et al. dans RAAB et al. (2020) pour la détection du concept drift : à chaque pas de temps, r échantillons sont sélectionnés aléatoirement. Ils sont comparés aux r échantillons les plus récents de la fenêtre parmi les n échantillons stockés dans une fenêtre glissante Ψ.

Comme le test KS nécessite deux fenêtres pour calculer la distance, dans KSWIN, les auteurs divisent la fenêtre coulissante locale Ψ en deux parties. La première R contient les r données les plus récentes, r étant un paramètre prédéfini : $R = \{x_i \in \Psi\}_{i=n-r+1}^{n}$. La deuxième fenêtre W échantillonne les données en sélectionnant uniformément les données parmi les anciennes données dans Ψ. La distribution uniforme utilisée est :

$$U(x|1, n-r) = \frac{1}{n-r}$$

Donc,

$$W = \{x_i \in \Psi | i < n-r+1 \wedge p(x) = U(x_i|1, n-r)\}$$

Le concept drift est détecté dans KSWIN lorsque $dist(R, W) > \sqrt{-\frac{ln(\alpha)}{r}}$, où α est la probabilité de la statistique du test de KS.

Le test de KS est une méthode unidimensionnelle. Cependant, les flux de données peuvent être multidimensionnels, il est donc important d'adapter KSWIN à ce contexte. Dans ce but, nous proposons un N-Dimensionnal KSWIN, que nous appelons NDKSWIN dans le reste du chapitre. L'idée de NDKSWIN est de déclarer une dérive si un changement est détecté pour au moins une des n

dimensions (tout en utilisant KSWIN).

La figure 5.5 montre le workflow de l'utilisation de NDKSWIN comme amélioration de IForest ASD afin de détecter les dérives avant les anomalies. Une première fenêtre est utilisée pour créer l'instance NDKSWIN avec tous les paramètres nécessaires. Elle est également utilisée pour construire le modèle de détection d'anomalies en utilisant l'algorithme IForest. Pour les fenêtres suivantes, avant d'appliquer la détection d'anomalies, toutes les fenêtres doivent passer par l'étape NDKSWIN. La fonction *add_element()* choisit aléatoirement n échantillons de la fenêtre et sélectionne aléatoirement n dimensions. Pour chaque donnée sélectionnée, chaque valeur de chaque dimension sélectionnée doit passer le test KSWIN *detected_change()*. Si un changement est détecté, le modèle est mis à jour avec toutes les données de la fenêtre actuelle. Quelque soit le résultat du test NDKSWIN, la fenêtre actuelle doit passer par le modèle IForest (mis à jour ou non) pour la détection d'anomalies.

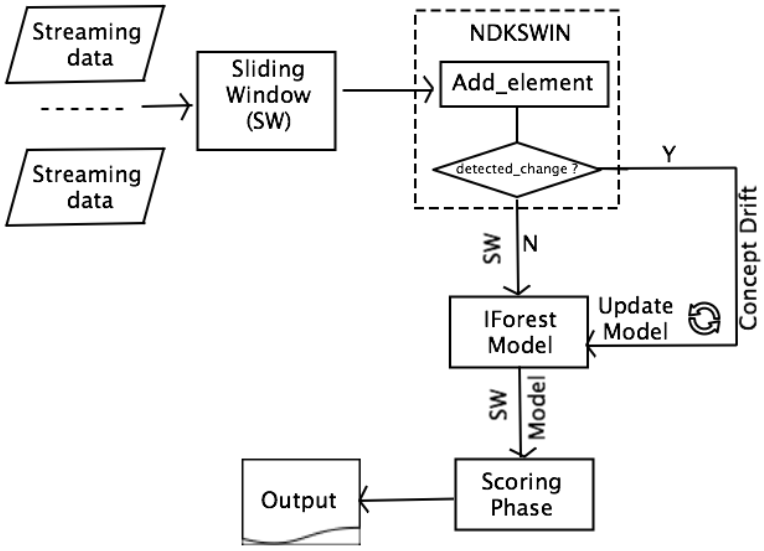


Figure 5.5 – Workflow de IForest ASD basé sur NDKSWIN (NDKSWIN IFA).

5.4 Expérimentations et discussions

Dans cette section, nous discutons de l'évaluation expérimentale obtenue en comparant la méthode IForest ASD avec les trois nouvelles méthodes proposées SADWIN IFA (SAd), PADWIN IFA (PAd) et NDKSWIN IFA (KSi). Pour ce faire, nous avons utilisé les jeux de données réels et le flux de données synthétiques SEA qui sont présentés dans le Tableau 5.1. Dans les expériences suivantes, nous cherchons à prouver qu'au lieu de mettre à jour le modèle à chaque fenêtre comme le fait IForest ASD dans le pire des cas, il est préférable d'implémenter une détection performante de dérive pour collaborer avec IForest ASD afin de traiter à la fois les anomalies et les dérives.

5.4.1 Jeux de données et métriques

5.4.1.1 Description des jeux de données

Nous avons testé les différentes méthodes présentées ci-dessus sur trois jeux de données réels bien connus (Dua et Graff, 2017), qui ont été utilisés dans l'article original d'IForest ASD.

Nous avons également utilisé un flux de données simulé avec différentes configurations pour comparer IForest ASD aux améliorations que nous proposons. Nous avons effectué un benchmark avec la méthode Half-space trees sur une variété de configurations (12 pour chaque jeu de données). Nous générons le flux simulé en utilisant le générateur SEAGenerator qui est disponible sur scikit-multiflow et qui est basé sur le générateur de flux de données proposé dans (Street et Kim, 2001). Comme décrit dans la documentation de scikit-multiflow [1], "Il génère 3 attributs numériques, qui varient de 0 à 10, où seulement 2 d'entre eux sont pertinents pour la tâche de classification. Une fonction de classification est choisie, parmi quatre possibles. Ces fonctions comparent la somme des deux attributs pertinents avec une valeur seuil, qui est unique pour chacune des fonctions de classification. En fonction de la comparaison, le générateur classera une instance dans l'une des deux étiquettes possibles". Ces flux de données, appelées

1. https://github.com/scikit-multiflow/scikit-multiflow/blob/master/src/skmultiflow/data/sea_generator.py

Tableau 5.1 – Jeux de données et paramètres.

	Shuttle	Forest-cover	SMTP	SEA
Nombre de dimension	9	10	3	3
Taux d'anomalies (u, Drift Rate)	7.15%	0.96%	0.03%	0.1%
Taille du jeu de données	49,097	286,048	95,156	10,000
Taille de fenêtres (W)	[50 , 100, 500 , 1000]			
Nombre d'arbres (t)	[30, 50, 100]			

SEA, ont du bruit mais ne présentent pas de dérive. Ainsi, nous nous attendons à ce que les méthodes expérimentées ne détectent aucune dérive. Le tableau 5.1 résume les caractéristiques des jeux de données.

5.4.1.2 Configuration de l'expérimentation

Afin de comparer IForest ASD à SADWIN IFA, PADWIN IFA et NDKSWIN IFA, nous avons réalisé des expériences en utilisant les paramètres décrits dans le Tableau 5.1.

Le nombre d'observations (échantillons) est de 10 000 pour les jeux de données dans chaque expérience. Le seuil d'anomalies est fixé à 0.5. Nous fixons la valeur du paramètre u au taux réel d'anomalies dans les jeux de données, comme cela a été fait pour IForest ASD dans (Ding et Fei, 2013). Cette valeur est utilisée afin de détecter la dérive et de réinitialiser le modèle de détection d'anomalies, comme le montre la figure 4.5.

D'après les résultats obtenus dans le tableau 4.3, avec 30 arbres, IForest ASD donne de bonnes performances. Pour ce qui concerne la taille de la fenêtre, les performances diminuent après 500. Par conséquent, nous utilisons $W \in \{100, 500\}$ et $t = 30$ pour la comparaison entre IForest ASD IFA, SADWIN IFA, PADWIN IFA, et NDKSWIN IFA.

5.4.1.3 Métriques d'évaluation

Nous avons étudié les trois paramètres suivants : Le score F1, le ratio du temps d'exécution (entraînement + test) (IForest ASD/Autre méthode), et le ratio de la taille du modèle (Autre méthode/IForest ASD) pour la mise en œuvre

d'Iforest ASD.

La performance des modèles est mesurée en suivant la stratégie appelée "prequential evaluation" [2] (méthode test-then-train) qui est spécifiquement conçue pour le contexte de flux. Chaque échantillon sert à deux fins (test puis train). Les échantillons sont analysés séquentiellement, dans l'ordre d'arrivée, et deviennent immédiatement inaccessibles. Cette approche garantit que le modèle soit testé sur de nouveaux échantillons qui n'ont pas encore été utilisés lors de la phase d'entraînement. Toutes les mesures ont été fournies par le framework scikit-multiflow en utilisant le "prequential evaluator".

5.4.2 Résultats

Sur la base des résultats expérimentaux précédents, nous avons observé que le nombre d'arbres n'a pas d'impact significatif sur les performances des modèles pour plus de 30 d'arbres, et nous avons également obtenu peu d'améliorations pour des tailles de fenêtre supérieures à 500. Par conséquent, dans les expériences suivantes, nous utilisons une taille de fenêtre de 100 et 500, avec 30 arbres dans l'ensemble. Nous avons comparé quatre méthodes sur les deux jeux de données réels (Shuttle, SMTP) et le jeu de données synthétique, SEA. Nous avons particulièrement choisi ces jeux de données réels, car ils présentent différentes caractéristiques. En effet, Shuttle est composé de 9 attributs, alors que SMTP n'a que 3 dimensions. Shuttle contient plusieurs anomalies par rapport à SMTP, qui ne compte que 0.03% d'anomalies. Nous avons généré le flux SEA, comme mentionné ci-dessus, afin de tester les méthodes dans la condition neutre, c'est-à-dire dans un flux de données réel qui ne contient aucune anomalie, mais seulement $0,1\%$ de bruits.

De plus, la valeur du paramètre drift_rate qui est fournie à IForest ASD correspond au pourcentage de bruit ou d'anomalies présent dans le flux de données. Cependant, pour les autres méthodes, aucune information n'a été fournie concernant le pourcentage d'anomalies existant dans le jeu de données.

Dans le tableau 5.2, nous fournissons les performances en termes de $F1$-score

2. https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.evaluation.EvaluatePrequential.html

des quatre méthodes avec les différentes configurations mentionnées ci-dessus. Nous avons mis en évidence les meilleurs résultats en les mettant en gras. Pour les tailles de fenêtre 100 et 500, nous avons remarqué que le temps de test nécessaire pour prédire les anomalies des 3 nouvelles méthodes (SAd, PAd, KSi) est inférieur à celui du streaming IForest ASD. Ainsi, nous avons fourni une métrique, qui est égale au ratio du temps d'exécution total de IForest ASD sur le temps d'exécution des autres méthodes (SAd, PAd, Ksi) (si la métrique est égale à 3, cela signifie que la méthode est trois fois plus rapide que la méthode originale). Nous avons effectué la même opération pour la métrique de la taille du modèle. Les résultats sont présentés dans le tableau 5.2 sous forme de rapport de temps et de rapport de taille, respectivement.

Le code source des différentes expériences est disponible sur Github à l'adresse `https://github.com/Elmecio/IForestASD_based_methods_in_scikit_Multiflow`.

5.4.3 Analyses

Le tableau 5.2 montre que la méthode IForest ASD met presque systématiquement à jour le modèle dans chaque fenêtre pour SMTP et SEA. Comme il y a très peu d'anomalies dans SMTP et aucune anomalie dans SEA, les fausses alarmes de la méthode conduisent IForest ASD à conclure à une dérive car le pourcentage prédéfini est dépassé. Ceci est dû au fait qu'IForest ASD n'intègre pas un mécanisme explicite de détection de dérive. Basée sur le pourcentage d'anomalies détectées dans la fenêtre, IForest ASD est fortement dépendante de la capacité du modèle à détecter les anomalies et surtout les vraies anomalies.

La meilleure performance est obtenue sur le jeu de données Shuttle qui contient de nombreuses anomalies. En termes de détection de dérive, nous avons remarqué que les méthodes basées sur ADWIN n'ont pas détecté de dérive sur Shuttle et SEA. Ce comportement attendu est dû à l'absence de dérives dans le jeu de données SEA. En revanche, NDKSWIN IFA détecte une dérive dans certaines fenêtres et met à jour le modèle en conséquence grâce à sa stratégie qui prend en compte plusieurs dimensions lors de la recherche de dérive. En conséquence, si les statistiques changent pour une dimension, la

Tableau 5.2 – Comparaison entre IForest ASD (IFA), SADWIN IFA (SAd), PADWIN IFA (PAd) et NDKSWIN IFA (KSi), sur les jeux de données Shuttle, SMTP, et SEA avec $W \in \{100, 500\}$ et $t = 30$.

		Shuttle				SMTP			
Metrics	**W**	**IFA**	**SAd**	**PAd**	**KSi**	**IFA**	**SAd**	**PAd**	**KSi**
F1	100	0.72	**0.74**	0.73	0.70	**0.37**	0.33	0.33	0.34
	500	0.80	**0.81**	0.80	0.80	**0.41**	0.40	0.39	0.39
Model	100	0	0	0	3	88	6	10	11
updated	500	0	0	0	2	15	3	5	2
Time	100	-	0.99	1.01	1.05	-	1.09	1.03	1.08
Ratio	500	-	1.00	1.00	1.00	-	1.07	1.09	1.10
Size	100	-	0.94	0.97	0.94	-	1.32	1.12	1.24
Ratio	500	-	0.97	0.98	0.98	-	0.89	1.39	1.31

		SEA			
Metrics	**W**	**IFA**	**SAd**	**PAd**	**KSi**
F1	100	0.41	0.39	**0.42**	0.41
	500	0.39	**0.40**	0.39	0.38
Model	100	99	0	0	7
updated	500	19	0	0	2
Time	100	-	1.03	1.05	1.05
Ratio	500	-	1.01	1.02	1.02
Size	100	-	0.96	0.93	1.06
Ratio	500	-	0.98	0.96	1.03

méthode suppose qu'une dérive s'est produite. Ceci pose le problème du choix de la dimension à considérer et de l'importance à donner à chaque dimension. Pour atténuer l'effet de ce problème, une technique de sélection des caractéristiques peut être intégrée afin de rendre la méthode plus robuste et efficace. L'utilisation du même modèle depuis le début, sans le mettre à jour lorsque le flux évolue, conduit à des résultats similaires, en termes de score F1 et de temps d'exécution des méthodes IForest ASD, SADWIN IFA et PADWIN IFA. Cependant, SADWIN IFA et PADWIN IFA sont légèrement plus lentes que IForest ASD, ce qui est normal dans ces circonstances.

Les méthodes proposées sont généralement plus rapides que IForest ASD standard. En effet, lors de la mise à jour de son modèle, IForest ASD consomme

du temps supplémentaire pour construire les 30 arbres à partir des 100 ou 500 instances de la fenêtre, selon le tableau 5.2. On pourrait initialement penser que les méthodes SADWIN IFA, PADWIN IFA et NDKSWIN IFA devraient prendre plus de temps que IForest ASD, car elles intègrent ADWIN et NDKSWIN, qui devraient également consommer du temps lors de la recherche de dérives. Or, ce n'est pas le cas, puisque ces méthodes ne mettent à jour le modèle que lorsqu'une dérive est détectée. Par conséquent, IForest ASD prend finalement plus de temps qu'elles. Par contre, IForest ASD consomme moins de mémoire que ces méthodes, ce qui s'explique par le fait qu'IForest ASD supprime complètement l'ancien modèle pour en créer un nouveau. Néanmoins, ADWIN et NDKSWIN gardent en mémoire une fenêtre glissante afin de détecter d'éventuelles dérives et chaque fois que de nouvelles données sont ajoutées à la fenêtre, ADWIN et NDKSWIN effectuent des calculs statistiques pour la détection des dérives. Dans le tableau 5.2, nous avons remarqué que les performances en termes de score F1 des nouvelles méthodes sont soit similaires à celles d'IForest ASD, soit meilleures. Nous remarquons aussi qu'IForest ASD effectue davantage de mises à jour du modèle que les nouvelles méthodes. Cela implique qu'il n'est pas nécessaire de mettre à jour le modèle dans chaque fenêtre. D'où l'importance d'intégrer une méthode de détection de dérive explicite.

En résumé, les principaux points de conclusion sont les suivants :

— IForest ASD ne dispose pas d'une véritable méthode de détection de dérive car la politique de mise à jour du modèle dépend d'un seuil défini par l'utilisateur (taux d'anomalies), qui n'est pas toujours disponible pour une application réelle.

— L'intégration d'une méthode de détection de dérive en conjonction avec l'IForest ASD réduit le temps d'apprentissage tout en maintenant des performances similaires.

5.5 Discussion

Dans ce chapitre, nous avons mis en évidence le principal inconvénient de IForest ASD. Duu fait qu'il ne soit pas adapté au traitement des flux de données évolutifs et à la détection de drift. Ainsi, nous avons proposé différentes versions

d'IForest ASD qui incluent des méthodes de détection de drift bien connues à savoir ADWIN et KSWIN qui est plus récente. Nous avons étendu cette dernière pour pouvoir traiter des flux de données à n dimensions et l'avons appelée NDKSWIN. SADWIN IFA et PADWIN IFA sont des améliorations de IForest ASD basées sur ADWIN. NDKSWIN IFA est une amélioration de IForest ASD basée sur KSWIN. L'expérimentation menée sur des jeux de données réels et simulés montre la capacité de nos méthodes proposées à détecter les dérives en utilisant moins de ressources (en termes de temps et de mémoire) que la méthode IForest ASD tout en gardant des performances similaires (ou meilleures dans certains cas) en termes de taux de détection d'anomalies.

Nos approches prometteuses peuvent être approfondies afin d'améliorer leur efficacité et leur précision en réalisant davantage d'expériences et en les analysant en profondeur. Les performances de NDKSWIN IFA peuvent être améliorées en utilisant un réducteur de dimensions afin d'identifier les dimensions les plus pertinentes pour évaluer la détection de drift. En effet, les différents attributs n'ont pas la même importance dans le jeu de données et la sélection des attributs peut réduire de manière significative le temps d'exécution de NDKSWIN.

Une autre amélioration majeure de ce travail serait de mettre à jour correctement le modèle de détection d'anomalies en prenant en compte les détecteurs d'anomalies et les observations précédentes au lieu de les écarter complètement. Cela peut être fait en utilisant des approches d'apprentissage adaptatif. Afin de réduire le temps d'exécution et de se rapprocher des principes d'IForest, il faudrait créer la forêt avec une partie des données de la fenêtre et non la fenêtre entière, comme le fait IForest ASD. Une autre approche complémentaire serait de faire une mise à jour partielle du modèle, c'est-à-dire qu'au lieu de supprimer toute la forêt, on pourrait mettre à jour seulement les arbres qui sont affectés par la dérive.

Nous tenons à remercier Mariam BARRY et Maroua BAHRI pour leur contribution aux travaux décrits dans ce chapitre.

Chapitre 6

Détection distribuée d'anomalies

Sommaire

Les résultats obtenus dans ce chapitre ont été publiés dans la conférence internationale ICCCI 2022 :
Maurras Ulbricht TOGBE, Yousra CHABCHOUB, Aliou BOLY et Raja CHIKY. Distributed anomalies detection using Isolation Forest and Spark. 14th International Conference on Computational Collective Intelligence, ICCCI 2022

6.1 Introduction

Comme la plupart des méthodes de détection d'anomalies, IForest a été conçu pour fonctionner sur une seule machine de manière centralisée. Une telle architecture voit rapidement ses limites face à un énorme volume de données

générées par les différents systèmes. Dans un contexte de Big data, le traitement et le stockage des données sur une seule machine n'est pas envisageable. Avec l'émergence du cloud, les architectures distribuées ont permis une meilleure évolutivité, et ont amélioré la disponibilité ainsi que le temps de réponse grâce à un traitement plus rapide. Récemment, les chercheurs se sont particulièrement intéressés aux solutions basées sur des algorithmes d'apprentissage automatique distribués (voir (Das et al., 2011 ; Zeng et al., 2012 ; Qasem et al., 2022)). En 2016, Google a introduit l'idée d'apprentissage fédéré (Pandey et al., 2022). Il consiste en un apprentissage local au niveau de chaque nœud. Chaque nœud envoie son modèle au serveur qui agrège les modèles pour construire un super modèle qui sera partagé avec les différents nœuds. Cette technique est souvent utilisée dans l'Internet des objets où chaque nœud construit son modèle sur ses données locales (N Hussain et al., 2022). Une autre technique, largement utilisée dans la littérature, est l'apprentissage distribué. La distribution consiste à faire participer plusieurs nœuds à la construction d'un modèle général en partageant les données et les tâches de traitement entre ces nœuds. La distribution nécessite une communication entre les différents nœuds pour construire le résultat final ou pour générer une décision collective, ce qui induit un temps de traitement supplémentaire par rapport à une version centralisée. Ce temps de communication doit être minimisé par rapport au temps gagné par la distribution des tâches. Une comparaison entre les deux techniques de traitement des données mentionnées ci-dessus est présenté dans (Asad et al., 2021).

La détection d'anomalies est un domaine de recherche où les algorithmes d'apprentissage automatique distribués sont très pertinents. En effet, de nombreuses adaptations des méthodes traditionnelles de détection d'anomalies au contexte de la distribution sont notées dans la littérature comme Solaimani et al. (2014), C Wang et al. (2018) et Bogatinovski et Nedelkoski (2020). Puisque, pendant sa phase d'apprentissage, IForest construit des arbres binaires indépendants basés sur différents échantillons de données, cet algorithme est très adapté à la distribution. Deux versions distribuées de IForest basées sur Spark ont été proposées : LDIForest (*LDIForest* 2019) et FDIForest (Yang et contributors, 2018). Dans ce chapitre, nous nous intéressons à l'étude et à la comparaison de ces deux implémentations distribuées d'IForest.

Le reste du chapitre est structuré comme suit : Dans la section 6.2, nous présentons d'abord les différents composants du framework de distribution Spark. Ensuite dans la section 6.3, nous expliquons les deux solutions IForest distribuées existantes qui sont basées sur Spark. Et enfin dans la section 6.4, nous comparons expérimentalement ces deux solutions sur différents jeux de données réels. La conclusion et les travaux futurs sont présentés dans la section 6.5.

6.2 Outils de distribution

Les algorithmes de machine learning comportent deux phases : la phase d'entraînement et la phase de test. Chacune de ces phases peut être distribuée. Verbraeken et al. ont présenté dans (Verbraeken et al., 2020) une étude complète de différents systèmes de distribution et leurs performances. Deux concepts de distribution du traitement des données sont recensés : la distribution des données et la distribution du modèle. La distribution des données consiste à diviser les données à disposition en des sous-ensembles répartis entre les noeuds existants selon les critères définis. Au niveau de chaque noeud, il s'agit du même algorithme qui est exécuté et le résultat est agrégé au niveau du noeud principal, appelé noeud maître. Concernant la distribution du modèle, l'algorithme est divisé en différents modules pouvant s'exécuter en parallèle, indépendemment les uns des autres. Ici, chaque noeud reçoit la totalité des données pour exécuter son module. Les données sont donc dupliquées au niveau de chaque noeud. Et ensuite, le noeud maître agrège les résultats. Généralement, la distribution des données est privilégiée par rapport à la distribution du modèle car la distribution des données est plus facile à implémenter et est proposée par la plupart des outils de distribution existants dans la littérature. Il existe différents frameworks et librairies pour le traitement distribué des données. Dans cette section, nous présenterons brièvement quelques uns des frameworks les plus utilisés dans la littérature.

6.2.1 Frameworks et librairies de traitement distribué

La figure 6.1 présente quelques frameworks d'entraînement distribué utilisés pour le deep learning et/ou pour le machine learning. Pytorch (Paszke et al., 2019) a été développé par Facebook et permet de faire de l'entraînement distribué de manière efficace en implémentant des modules de réseaux de neurones. Pytorch permet de faire un entraînement parallèle sur une seule machine en utilisant plusieurs GPUs (Graphics Processing Unit). Il offre également la possibilité de distribuer le modèle et non uniquement les données. DeepSpeed [1] est un framework d'entraînement axé sur la distribution du modèle. Il est proposé par Microsoft et fonctionne avec Pytorch. Proposé par Google, distributed TensorFlow (Abadi et al., 2015) est principalement tourné vers la distribution des données. Il existe des extensions [2] de TensorFlow comme Mesh TensorFlow pour l'entraînement de modèles de grande échelle. Mesh TensorFlow est orienté TPU (Tensor Processing Unit) (YE Wang et al., 2019). Développé par Yahoo, TensorFlowOnSpark permet de faire de l'entraînement distribué avec Apache Spark et Apache Hadoop. TensorFlow Federated est une autre extension pour faire du machine learning avec des données décentralisées. D'autres outils existent comme BigDL (Dai et al., 2019) proposé par Intel et Horovod (Sergeev et Balso, 2018) inventé en 2017 par Uber. Le traitement distribué de données peut être coûteux pour plusieurs entreprises et aux utilisateurs. La 2ème alternative est l'utilisation des services du cloud pour le traitement distribué des données. Sur la figure 6.1, nous pouvons voir les plateformes principales de traitement distribué de données dans le cloud comme Google Cloud Computing [3] et le notebook Google Colab ou Colaboratory [4]. Il permet l'exécution de code python en ligne sans aucune configuration requise et avec un accès gratuit au GPU. Il y a également Amazon SageMaker [5], Microsoft Azure [6] etc... Plusieurs études comparatives ont été menées sur ces plateformes principales de cloud computing (Wankhede et al., 2020 ; B Gupta et al., 2021 ; Kaushik et al., 2021).

1. https://www.deepspeed.ai/
2. https://neptune.ai/blog/extensions-for-tensorflow
3. https://cloud.google.com/
4. https://colab.research.google.com/?hl=fr
5. https://aws.amazon.com/fr/sagemaker/
6. https://azure.microsoft.com/fr-fr/

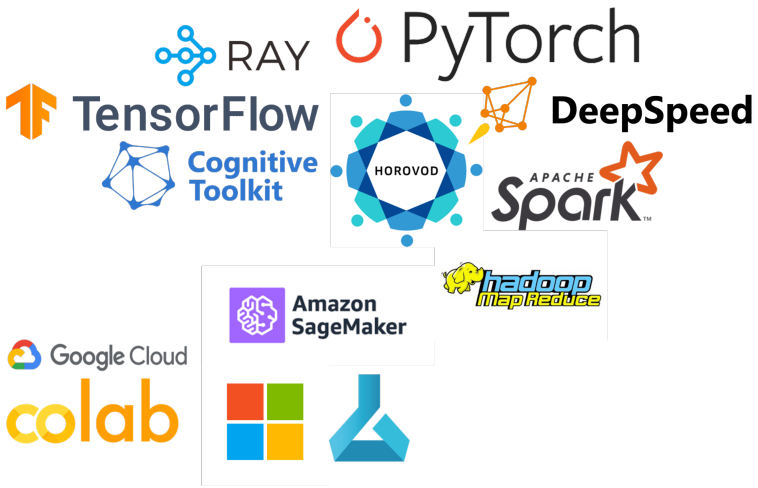


FIGURE 6.1 – Outils de traitement distribué de données.

Apache Spark est l'un des frameworks de traitement distribué de données les plus utilisés dans la littérature. Dans la suite, nous présenterons plus en détails Apache Spark et ses principales notions et librairies utiles pour la compréhension de nos travaux sur la détection distribuée d'anomalies.

6.2.2 Environnement Spark

Les deux solutions proposant une version distribuée d'IForest sont basées sur Apache Spark [7]. L'objectif de cette sous-section est d'expliquer l'architecture de ce framework. Une étude plus complète des outils d'apprentissage automatique distribués est présentée dans (VERBRAEKEN et al., 2020).

6.2.2.1 Apache Spark

Apache Spark (MENG et al., 2016 ; SALLOUM et al., 2016), est un framework pour le traitement de données à grande échelle. L'architecture globale de la distribution à l'aide de Spark est présentée dans la figure 6.2. Spark fonctionne avec plusieurs nœuds. Pour chaque programme Spark, un contexte spark doit être créé pour héberger et gérer le programme au niveau du programme pilote.

7. https://spark.apache.org/

Le nœud maître est le nœud qui contient le contexte spark. Le nœud maître travaille avec un gestionnaire de clusters qui lui permet de gérer tous les nœuds esclaves du cluster (Worker nodes). Le gestionnaire de clusters peut gérer différents travaux et chaque travail peut être divisé en plusieurs tâches qui seront distribuées au niveau du nœud esclave.

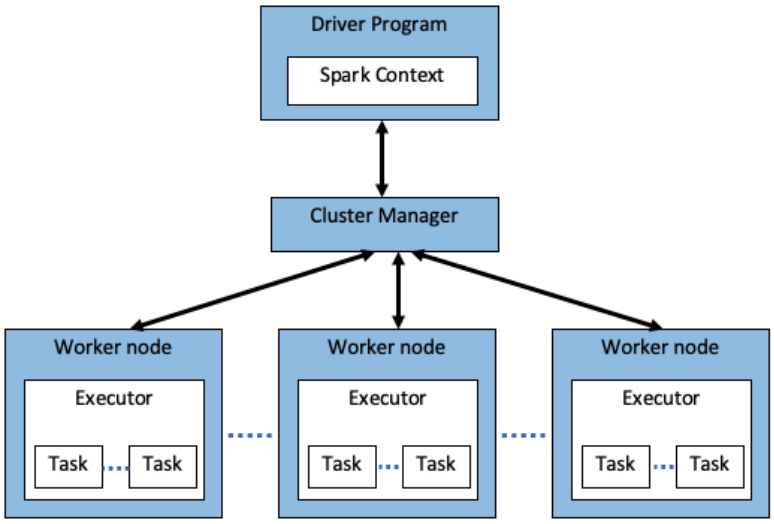


FIGURE 6.2 – Architecture de Apache Spark.

La figure 6.3 présente une structure simplifiée de l'écosystème Spark. Apache Core est le cœur de la puissance de Spark pour le traitement parallèle et distribué à grande échelle des données. Le Resilient Distributed Dataset (RDD) est la technologie de représentation et de distribution des données. Spark dispose d'une puissante bibliothèque pour l'interrogation et le traitement de données structurées : Spark SQL. GraphX est le module Spark pour le traitement des graphes. Spark Streaming est le module de gestion des flux de données et de calcul incrémental. L'apprentissage automatique est assuré par le module MLIB. Dans le cadre de ce chapitre, nous nous concentrons sur RDD, Spark SQL et MLlib (Machine Learning Library).

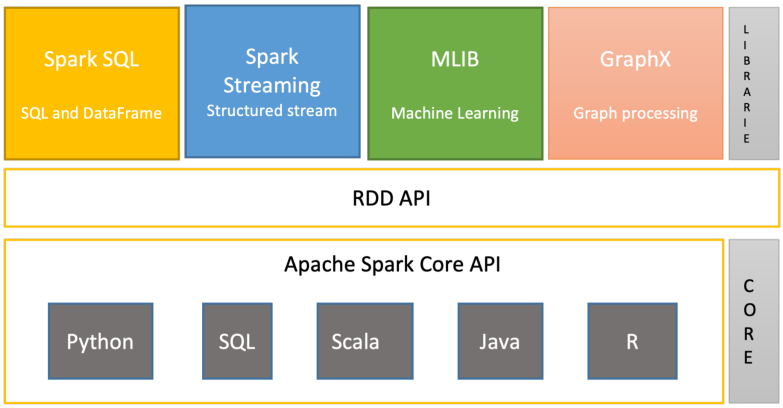


FIGURE 6.3 – Ecosystème de Apache Spark.

6.2.2.2 Resilient Distributed Dataset (RDD)

Avec Spark, les données sont représentées dans une structure particulière qui permet à Spark d'effectuer un traitement distribué fiable, ce qui constitue une innovation majeure. Il s'agit d'une collection d'éléments tolérants aux pannes qui peuvent être gérés en parallèle. La reprise après une panne est possible grâce à la conservation de l'historique de constitution du RDD. Il s'agit donc d'une représentation résiliente d'une collection de données partitionnées, distribuables sur plusieurs nœuds d'un même cluster. Les RDD utilisent la mémoire et l'espace disque selon les besoins. Spark offre la possibilité d'exécuter les RDD en mémoire ou de les faire persister en cache pour un traitement plus rapide. Enfin, afin de gérer l'accès simultané aux données, notamment dans un contexte de données à grande échelle, les RDD sont accessibles en lecture seule.

6.2.2.3 Spark SQL

Apache Spark permet, grâce à sa bibliothèque Spark SQL, d'interroger des données à l'aide de SQL. Cette bibliothèque permet également de représenter des ensembles de données sous la forme d'un dataframe. Un dataframe est structuré en colonnes contenant les dimensions et en lignes représentant les données. La représentation sous forme de jeu de données est également possible. La grande différence entre un dataframe et un dataset est que le dataset est hautement typé

et immuable. Couplé avec le RDD, cela permet une exécution très rapide des requêtes SQL.

6.2.2.4 MLlib

La bibliothèque d'apprentissage automatique (MLlib) fournit aux développeurs plusieurs méthodes de traitement de données. Les principales fonctions sont *fit()* pour l'apprentissage et *transform()* pour la phase de test (prédiction). Différentes mesures de performance sont également disponibles telles que l'aire sous la courbe ROC (ROC AUC), la matrice de confusion, le rappel, etc. MLlib fournit un environnement complet pour exécuter une méthode d'apprentissage automatique tout en profitant de la puissance de Spark pour la distribution.

Basées sur Spark, les deux versions distribuées d'IForest, LDIForest et FDIForest utilisent la bibliothèque MLlib. La figure 6.4 montre l'architecture générale des algorithmes d'apprentissage automatique dans MLib :

Phase de préparation des données : Une fois les données récupérées, elles sont prétraitées. Le prétraitement consiste généralement à créer un vecteur à partir des différentes colonnes de la trame de données.

Phase d'entraînement : Avec MLlib, la méthode $fit()$ de la classe $Estimator$ est utilisée pour créer les modèles pendant la phase d'apprentissage. Dans le contexte de IForest distribué, la création des échantillons d'entraînement et la construction des arbres binaires (i-trees) peuvent être distribuées. Les collections RDD sont bien sûr utilisées pour construire ces arbres. Avec MLlib, des méthodes sont disponibles pour persister le modèle construit, ce qui est très utile pour réutiliser le modèle (cas des flux de données) ou pour comparer ou mettre à jour les modèles précédemment créés.

Phase de test : Pendant la phase de test ou de scoring, le modèle créé est utilisé sur les données de test pour une classification ou une prédiction. Avec MLlib, la méthode $transform()$ de la classe $Transformer$ est utilisée à ce stade. Cette phase est divisée en deux opérations principales : la première est le calcul des

scores de chaque donnée de test, la seconde est la classification des données de test par comparaison avec le seuil. Le seuil de décision peut être un paramètre statique prédéfini (les auteurs ont proposé de le fixer à 0,5 dans (FT Liu et al., 2008)), ou calculé sur la base des scores obtenus. Cette dernière façon de générer dynamiquement le seuil de décision est celle qui est souvent privilégiée dans la littérature (voir Yang et contributors (2018) et *LDIForest* (2019)).

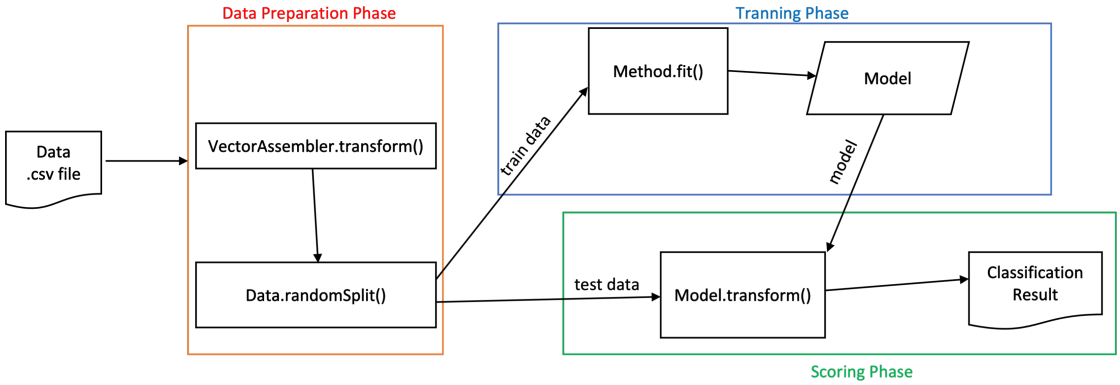


Figure 6.4 – Architecture générale d'un algorithme de machine learning utilisant MLlib

Dans cette section, nous donnons une description générale du framework de distribution Spark et expliquons la base commune qu'il fournit pour les deux versions distribuées d'IForest : LDIForest et FDIForest. Par la suite, après un état de l'art sur les méthodes distribuées de détection d'anomalies, nous analysons en profondeur ces deux implémentations afin de mettre en évidence leurs principales différences. Pour finir, nous les testons sur différents jeux de données réels afin de comparer leurs performances.

6.3 Détection d'anomalies distribuée : Etat de l'art

Un grand défi pour toute méthode de machine learning est de pouvoir traiter les données de manière efficace et à grande échelle pour une meilleure aide à la prise de décision. Dans le cadre du machine learning distribué, les revues O'Reilly et al. (2014), Ramotsoela et al. (2018) et Samara et al. (2022) présentent

différentes méthodes et approches de détection d'anomalies appliquées à l'IoT et aux réseaux de capteurs. Différentes catégories de méthodes peuvent être recensées dans la littérature : des méthodes basées sur le clustering présentées par Rajasegarar et al. dans (Rajasegarar et al., 2006), des méthodes basées sur les plus proches voisins revues par (Chhabra et al., 2008), des méthodes basées sur le deep learning exposées dans (T Luo et Nagarajan, 2018) etc. Pour ce qui concerne l'approche basée sur l'isolation, très peu de travaux ont été publiés dans la littérature. IForest, a été à l'origine conçue pour fonctionner de manière centralisée. Dans (Tao et al., 2018), Tao et al. se sont basés sur l'indépendance des arbres créés par IForest pour concevoir la méthode Spark Parallel IForest (SPIF). SPIF parallélise la création du modèle et la phase de scoring des données. SPIF a un temps d'exécution beaucoup plus petit que celui de la version originale de IForest. La parallélisation des tâches a fait gagner du temps d'exécution mais SPIF n'est pas résilient.

Il existe deux implémentations d'adaptations de la méthode IForest au contexte de la distribution. Il s'agit d'une implémentation appelée ici LDIForest de l'équipe Anti-abus de la société LinkedIn (*LDIForest* 2019) et de l'implémentation proposée par Yang et contributors dans (Yang et contributors, 2018) ici appelée FDIForest. Ces deux dernières implémentations open source, profitent pleinement de la librairie MLlib de Spark pour proposer une version distribuée de IForest.

LDIForest (*LDIForest* 2019) et FDIForest (Yang et contributors, 2018) sont deux implémentations différentes d'IForest distribué utilisant Spark. LDIForest est développé en Scala tandis que FDIForest est disponible en Scala et Python. Ces deux implémentations sont fonctionnellement identiques. La figure 6.5 montre l'architecture générale de ces deux implémentations.

Tout d'abord, pendant la phase de préparation des données, les données sont converties dans un format adapté à MLlib : format DataFrame ou Dataset. Ensuite, lors de la phase d'entraînement, t échantillons aléatoires de taille ψ sont générés. Chaque échantillon est transformé en un RDD pour bénéficier de la distribution sous Spark et sera utilisé pour la construction d'un arbre. Les arbres seront répartis dans les différents nœuds et représentent les différents modèles qui seront utilisés pour la phase de test. La phase de test (ou scoring) est

également distribuée et chaque élément sera classé comme anomalie ou donnée normale en fonction de son score comparé à un seuil de décision unique. Les auteurs d'IForest ont recommandé d'utiliser un seuil statique égal à $0,5$. Cependant, LDIForest et FDIForest génèrent ce seuil de décision de manière dynamique en fonction des scores des données traitées. En effet, afin d'avoir un seuil de décision bien adapté au jeu de données utilisé, ce seuil est calculé à partir de la méthode statistique $approxQuantile()$ de la classe $DataFrameStatFunctions$ de Spark SQL. Il s'agit d'une variante de l'algorithme de Greenwald-Khanna. L'idée est de configurer ce seuil en fonction du taux d'anomalies dans le jeu de données appelé taux de contamination ou facteur de contamination dans plusieurs implémentations d'IForest comme dans scikit-learn. Cela suppose que le taux d'anomalies dans le jeu de données soit connu à l'avance, ce qui n'est pas évident dans un contexte non supervisé. De plus, nous notons que FDIForest calcule ce seuil de décision à partir des scores de toutes les données de test, alors que LDIForest n'utilise que les données d'entraînement pour calculer le seuil. Ceci augmente le temps d'exécution de la phase de scoring de FDIForest et nécessite plusieurs passages sur le jeu de données.

LDIForest est une implémentation industrielle, open source, développée par LinkedIn. Cette implémentation est régulièrement mise à jour et améliorée. Cependant, le code de FDIForest est plus ancien et n'a pas été mis à jour récemment.

6.4 Comparaison entre LDIForest et FDIForest

Afin de comparer les performances des deux implémentations distribuées d'IForest : LDIForest et FDIForest, nous avons testé ces deux versions sur quatre (4) jeux de données réels différents. Nous avons fixé le nombre d'arbres, t, à 100, en suivant les recommandations des auteurs. Pour le choix de la taille de l'échantillon ψ, les auteurs recommandent d'utiliser 256 données pour la construction de chaque arbre. Une étude plus complète sur l'impact de ce paramètre sur les résultats et sa dépendance à la complexité des données (taille, dimensions, variabilité...) est proposée dans (Chabchoub, Togbe et al., 2022). Nous avons donc adapté ce paramètre au jeu de données considéré. Le choix de ψ, ainsi que les

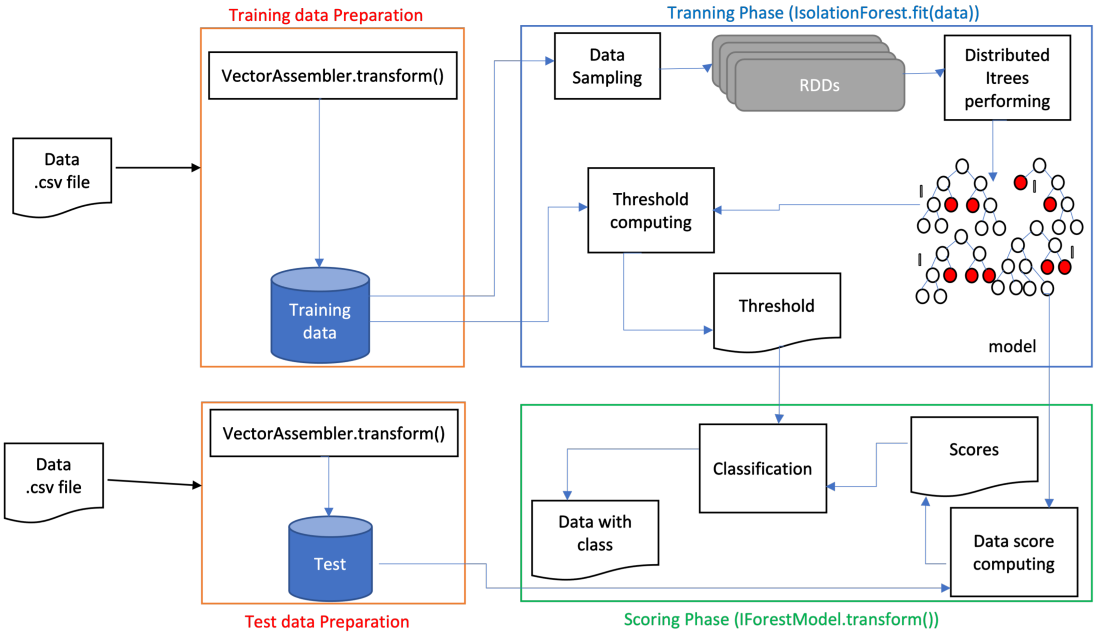


FIGURE 6.5 – Architecture général de LDIForest et de FDIForest

TABLEAU 6.1 – Caractéristiques des jeux de données publics utilisés.

Dataset	Size (n)	Dimensions	Anomalies	ψ
Mammography	11 183	6	2.32%	256
Shuttle	49 097	9	7.15%	256
SMTP	95 156	3	0.03%	2048
Forest Cover	286 048	10	0.96%	4096

caractéristiques des 4 jeux de données considérés sont présentés dans le tableau 6.1. Nous avons évalué ces méthodes avec la métriquee ROC AUC pour voir les performances des méthodes en terme de détection d'anomalies. La principale motivation de la distribution est d'améliorer la vitesse d'exécution afin de traiter les données reçues à un rythme de plus en plus élevé. Par conséquent, le temps d'exécution est un paramètre important à considérer lors de la comparaison des versions distribuées d'IForest. En particulier, nous avons étudié l'impact du nombre de cœurs de distribution et de la taille des données sur le temps d'exécution de LDIForest et FDIForest.

6.4.1 Résultats et analyses

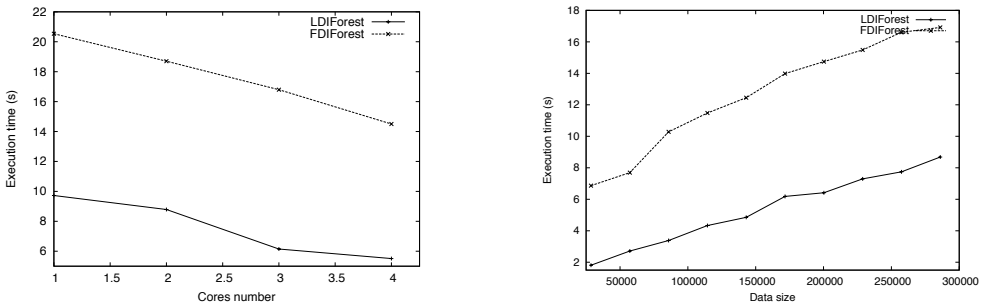
Le tableau 6.2 montre les résultats de l'exécution de LDIForest et FDIForest sur les 4 jeux de données décrits ci-dessus. Les expériences ont été menées sur un ordinateur mac OS avec un processeur Intel core i5 à 2,6 GHz et 8 Go de mémoire. Le code source est disponible sur Github : `https://github.com/Elmecio/Spark_Based_Distributed_Isolation_Forest`

Ce tableau montre que les deux implémentations de la distribution sont performantes pour la détection d'anomalies avec des valeurs très similaires de ROC AUC, assez proches de 1, pour les 4 jeux de données considérés. Pour chaque jeu de données, le temps d'exécution total de FDIForest et LDIForest est donné par le temps d'exécution moyen de 10 exécutions successives. Il s'agit de la somme des temps d'exécution des phases d'apprentissage et de test. Le tableau 6.2 montre aussi que, pour les 4 jeux de données, les deux méthodes sont assez rapides (avec un temps d'exécution inférieur à 18s pour le plus grand jeu de données). On peut également remarquer que LDIForest est beaucoup plus rapide que FDIForest. Le temps d'exécution de LDIForest est environ la moitié de celui de FDIForest. Cela est dû à la complexité du processus de génération du seuil de décision dans FDIForest : à partir du score de tous les éléments de données, alors que LDIForest se base uniquement sur le score des échantillons pour calculer le seuil de décision.

Tableau 6.2 – Comparaison de LDIForest et de FDIForest.

Dataset	ROC AUC		Execution Time (ms)	
	LDIForest	FDIForest	LDIForest	FDIForest
Mammography	0.73	0.73	**1 655**	3 867
Shuttle	0.98	0.98	**2 843**	4 372
SMTP	0.83	0.83	**4 669**	7 929
Forest Cover	0.84	**0.85**	**8 279**	17 316

Les figures 6.6a et 6.6b montrent respectivement l'impact du nombre de cœurs de distribution et de la taille des données sur le temps d'exécution de LDIForest et FDIForest. Pour cette expérience, nous avons utilisé le plus grand jeu de données, Forest Cover. Le nombre d'arbres construits pendant la phase d'apprentissage est fixé à 100. Chaque arbre utilise un échantillon de $\psi = 4096$



(a) Impact du nombre de coeurs. (b) Impact de la taille du jeu de données.

FIGURE 6.6 – Impact du nombre de cœurs et de la taille des données sur le temps d'exécution de LDIForest et FDIForest, jeu de données : Forest Cover.

éléments aléatoirement choisis dans l'ensemble du jeu de données. Pour faire varier la taille des données, nous avons tronqué l'ensemble de données pour chaque exécution.

Les résultats confirment la rapidité de LDIForest par rapport à FDIForest. De plus, nous constatons que le temps d'exécution diminue significativement avec le nombre de cœurs impliqués, grâce à la distribution implémentée par Spark à travers les RDDs. Enfin, le temps d'exécution total augmente linéairement avec la taille du jeu de données considéré, ce qui est cohérent avec la complexité de la phase de test donnée par $O(nt \log \psi)$. La complexité de la phase d'apprentissage est indépendante de la taille des données. Elle est seulement liée au nombre d'arbres à construire (t) et à la taille de l'échantillon qui génère chaque arbre (ψ).

6.5 Discussion

Dans ce chapitre, nous avons brièvement présenté l'écosystème de distribution : Apache Spark. Nous avons présenté et comparé deux implémentations majeures d'IForest distribué utilisant Spark : FDIForest et LDIForest. Nos résultats expérimentaux montrent que les deux solutions sont performantes pour la détection d'anomalies, avec un avantage notable de LDIForest sur FDIForest

en termes de temps d'exécution. Nous avons également abordé l'avantage de la distribution en évaluant l'impact du nombre de cœurs et de la taille des données sur le temps d'exécution. La détection distribuée d'anomalies dans les flux de données n'est pas assez explorée dans la littérature. IForest ASD est une évolution d'IForest adaptée au contexte des flux de données. Nos futurs travaux de recherche porteront sur la conception d'une version distribuée, avec Spark, de nos méthodes PADWIN IForest ASD, SADWIN IForest ASD et NDKSWIN IForest ASD en se basant sur l'étude réalisée dans ce chapitre.

Chapitre 7

Conclusion et travaux futurs

Sommaire

7.1 Contributions

Dans cette thèse, nous avons présenté nos travaux sur la détection d'anomalies dans les flux de données. Nous avons abordé ce sujet selon deux approches : l'approche centralisée dans un premier temps et l'approche distribuée dans un second temps. Nous avons commencé par présenter un état de l'art général des méthodes de détection d'anomalies dans différents contextes. En effet, après avoir listé les principales revues existantes, nous les avons classées en fonction des différents types de données (flux de données, séries temporelles, graphes, données spatiales, etc.), des techniques utilisées (statistiques, clustering, classification, plus proches voisins, etc.) et des domaines d'applications (détection

d'intrusion, fraude, santé, etc.). Nous avons proposé différentes améliorations de l'algorithme Isolation Forest pour réduire son temps d'exécution et pour mieux l'adapter au concept drift dans les flux de données. Ensuite nous avons étudié la distribution de la détection d'anomalies pour Isolation Forest. Toutes nos méthodes proposées ont été évaluées par différentes expérimentations sur des jeux de données réels et publics et sur des jeux de données synthétiques. Nos contributions ont donné lieu à des publications dans des conférences internationales et des journaux internationaux. Nos travaux, effectués dans cette thèse, peuvent être regroupés en trois parties.

7.1.1 Isolation Forest et Majority Voting IForest

L'étude comparative que nous avons effectuée entre les méthodes Local Outlier Factory (LOF), One-Class Support Vector Machine (OC-SVM) et Isolation Forest (IForest), trois des méthodes les plus connues et efficaces pour la détection d'anomalies au chapitre 2 a montré une supériorité de la méthode IForest. Nous avons donc décidé de l'étudier en profondeur. Dans le chapitre 3, nous avons effectué une étude approfondie de la méthode Isolation Forest. Après avoir analysé cette méthode suivant plusieurs aspects (ses limites, ses évolutions existantes, etc.), nous avons proposé une amélioration de Isolation Forest que nous appelons Majority Voting Isolation Forest. Les expérimentations menées ont prouvé que notre méthode est plus rapide avec une efficacité de détection similaire à Isolation Forest.

Etude de Isolation Forest : Proposée en 2008 par FT Liu et al., Isolation Forest (FT Liu et al., 2008) est une méthode non supervisée ensembliste de classification binaire spécialement conçue pour la détection d'anomalies. Elle se base sur une forêt d'arbres binaires, indépendants et aléatoires. Lors de la phase de construction du modèle, soit la forêt de t arbres, chaque arbre est construit à partir d'un échantillon de ψ données aléatoirement choisies dans le jeu de données initial, sans remise. Chaque nœud de l'arbre est scindé en deux jusqu'à isolation complète des ψ données ou jusqu'à la profondeur maximale de l'arbre qui est fonction de ψ, soit $\log_2(\psi)$. Les auteurs ont recommandé d'utiliser $t = 100$

et $\psi = 256$ comme valeurs par défaut. Dans notre étude, nous avons montré que le nombre d'arbres (t) n'a pas un impact majeur sur la performance de détection de IForest et avons vérifié que cent arbres est un bon compromis entre le temps d'exécution et la performance. En ce qui concerne la taille de l'échantillon (ψ), nos expérimentations ont prouvé que ψ devrait dépendre des caractéristiques du jeu de données, nous avons donc réfuté la recommandation d'utiliser $\psi = 256$. La taille optimale de l'échantillon dépend de la dimension et de la variance du jeu de données adressé. Avec IForest, les anomalies sont identifiées sur la base d'une décision collective de l'ensemble des arbres. Nous avons proposé une autre version basée sur la décision de la majorité.

Majority Voting IForest : La formule de calcul du score qu'utilise IForest fait intervenir le chemin de l'observation x dans tous les arbres de la forêt soit le chemin moyen. Pour réduire le temps d'exécution, notre MVIForest calcule le score de x pour chaque arbre et s'arrête quand la majorité est atteinte soit $t/2 + 1$ pour l'une des deux classes. Les expérimentations sur des jeux de données réels et synthétiques montrent que MVIForest est plus rapide avec des performances de détection similaires, sinon meilleures que IForest et sa version étendue EIF (Extended IForest).

7.1.2 Isolation Forest pour les flux de données

A la différence de la majorité des méthodes de détection d'anomalies, IForest ne calcule ni la densité ni la distance pour isoler les données anormales. La complexité de IForest et son mode de fonctionnement en font une méthode très adaptée aux contraintes des flux de données. En 2013, Ding et Fei ont proposé une nouvelle version de IForest adaptée aux flux de données, appelée IForest for Anomaly detection in Streaming Data (IForest ASD). Dans les chapitres 4 et 5, nous avons proposé une implémentation de IForest ASD dans le framework scikit-multiflow (River). Pour une meilleure gestion du concept drift, nous avons étendu IForest ASD en intégrant différentes façons de tenir compte du concept drift, augmentant ainsi la robustesse de IForest ASD. En effet, nous avons proposé trois nouvelles méthodes basées sur ADaptive WINdowing (ADWIN) et

Kolmogorov-Smirnov WINdowing (KSWIN). Les méthodes ADWIN et KSWIN sont bien connues pour leur efficacité dans la détection du concept drift.

Amélioration de IForest ASD : Nous avons dans un premier temps implémenté IForest ASD dans le framework dédié au traitement de flux de données scikit-multiflow (River). Nos expérimentations ont montré l'efficacité de notre implémentation. La gestion du concept drift est un défi majeur dans la gestion des flux de données. IForest ASD détecte le concept drift en prenant en compte un paramètre d'entrée u représentant le taux d'anomalies dans chaque fenêtre. IForest ASD étant une méthode non supervisée, il s'agit d'une limite importante car ce paramètre n'est pas connu à l'avance. Pour résoudre ce problème, nous avons proposé trois méthodes basées sur ADWIN et KSWIN.

Score-based ADWIN IFA, Prediction-based ADWIN IFA et NDKSWIN–based IFA : ADWIN et KSWIN sont deux méthodes de détection du concept drift. ADWIN détecte principalement le drift du modèle pendant que KSWIN détecte le drift au niveau des données. Nous avons amélioré IForest ASD en intégrant ADWIN et KSWIN dans IForest ASD. La première version proposée se base sur les scores pour détecter le drift du modèle d'où Score-based ADWIN IFA. La deuxième version se base sur les prédictions pour détecter si le modèle doit changer ou pas, soit Prediction-based ADWIN IFA. La dernière version NDKSWIN-based IFA détecte le drift directement dans les données avant même la phase de test.

7.1.3 Distribution de Isolation Forest

Dans certains contextes comme les réseaux de capteurs, la collecte des données se fait de façon distribuée. Il convient dans ces situations de distribuer aussi les traitements pour éviter l'envoi de données massives vers un serveur central et pour accélérer le traitement. Pour une détection distribuée des anomalies avec IForest, deux versions distribuées de IForest ont été récemment proposées soient LDIForest (*LDIForest* 2019) et FDIForest (Yang et contributors, 2018). ces deux versions se basent sur le framework de distribution Spark et présentent quelques

différences. Notre étude comparative de ces deux versions sur différents jeux de données réels a montré la supériorité de LDIForest. LDIForest est open source et a été régulièrement maintenue. Un travail futur à court terme est d'implémenter une version distribuée de nos méthodes adaptées aux flux de données : SADWIN-based IForest ASD, PADWIN-based IForest ASD et NDKSWIN-based IForest ASD.

7.2 Travaux futurs

Isolation Forest est une méthode performante pour la détection d'anomalies avec une faible complexité. Certaines de ses limites ont été adressées dans cette thèse. Les évolutions de IForest proposées dans cette thèse ont été validées sur différents jeux de données réels et synthétiques. Néanmoins, d'autres limites restent à considérer. Dans cette section, nous évoquons quelques pistes de travaux futurs à court et à long terme.

7.2.1 Travaux futurs à court terme

Seuil de décision dynamique : Lors de la phase de test, IForest fournit pour chaque donnée un score calculé sur la base de la longueur moyenne des chemins de cette donnée sur les t arbres. Le seuil de décision suggéré par les auteurs est statique, égal à $0,5$. En pratique, lorsque le score d'une donnée est supérieur à $0,5$ la donnée est classée anormale. Nos expérimentations sur des jeux de données réels et synthétiques ont montré que cette règle n'est pas adaptée à tous les types de données et génère parfois beaucoup de fausses alarmes. Le seuil de décision idéal selon nos expérimentations dépend du degré de ressemblance entre les données normales et anormales. En effet, la différence de densité et aussi la distance entre les données normales et anormales ont un impact considérable sur l'efficacité de la détection pour IForest et devraient être prises en considération lors du choix du seuil de détection. L'intégration de ces paramètres dans le choix d'un seuil de décision dynamique fera partie de nos futurs travaux de recherche.

Anomalies multidimensionnelles : Nos expérimentations ont montré que quand l'anomalie est portée par plusieurs dimensions, elle est difficile à détecter par IForest vu son mode de fonctionnement. Le nombre de dimensions est donc un paramètre très important, à prendre en compte pour améliorer le passage à l'échelle d'IForest et l'adapter aux données de grandes dimensions. Une évolution possible d'IForest serait de choisir la taille de l'échantillon ψ en fonction de la taille des données et aussi de leurs dimensions. En effet, afin de détecter une anomalie portée par m dimensions, il est évident qu'un arbre de profondeur minimale m doit être utilisé. Cependant, la profondeur de l'arbre est bornée par $depth\_max = \lceil log_2(\psi) \rceil$. Cette formule doit être modifiée pour inclure m soit directement, soit en liant m à ψ.

NDKSWIN-based IFA et ADWIN-based IFA : Nos méthodes proposées dans cette thèse pour la détection d'anomalies dans les flux de données, peuvent être améliorées notamment pour la réduction du temps d'exécution et pour la prise en compte des différents types de concept drift. Concernant la réduction du temps d'exécution, les jeux de données sont généralement multidimensionnels et la prise en compte de chaque dimension pour la recherche du drift dans NDKSWIN peut s'avérer gourmande en temps d'exécution surtout avec des jeux de données de grandes dimensions. À court terme, NDKSWIN peut être améliorée avec une sélection d'attributs significatifs.

Un autre axe de travail futur serait d'implémenter une version distribuée avec Spark de nos méthodes adaptées aux flux de données : PADWIN IForest ASD, SADWIN IForest ASD et NDKSWIN IForest ASD.

7.2.2 Travaux futurs à long terme

Deep learning et apprentissage par renforcement : Le deep learning est une sous-catégorie du machine learning qui s'inspire du fonctionnement du système nerveux des êtres vivants. Basées sur les réseaux de neurones, les méthodes du deep learning ont permis de répondre à des problèmes complexes dans différents domaines d'application. Par exemple, les Réseaux de Neurones Convolutifs (CNN) servent à traiter les images et les Réseaux de Neurones Récurrents (RNN)

pour le traitement des textes. Comparées aux méthodes de machine learning classiques, le deep learning nécessite un volume de données conséquent et par conséquent de grandes puissances de traitement pour atteindre de bonnes performances. Quand le problème à résoudre est compliqué et complexe comme le déclenchement automatique d'une action ou réaction dans une situation totalement imprévisible, l'apprentissage par renforcement ou Reinforcement Learning devient incontournable (Mao et al., 2022 ; EF Morales et Escalante, 2022). L'apprentissage par renforcement consiste à laisser l'agent apprendre à atteindre un but dans un environnement complexe et incertain en essayant plusieurs façons possibles et en apprenant de ses erreurs. Il fonctionne par récompense ou punition. Il peut être supervisé ou non supervisé. Il peut être appliqué à tous les types de données. L'apprentissage par renforcement est utilisé pour les voitures autonomes (DeepRacer [1], Wayve.ai [2]). L'apprentissage par renforcement est également utilisé dans l'automatisation industrielle (exemple des datacenters de Google dont le refroidissement est géré par une IA qui décide quand il faut agir), la finance et le trading pour prédire et décider de quand vendre ou acheter des actions, le traitement de langage naturel, la médecine où des prothèses peuvent aider à faciliter la marche d'un personne, etc.
Quelques travaux récents existent dans la littérature pour la détection d'anomalies à travers l'apprentissage par renforcement (Pang, Hengel et al., 2021 ; Lischke et al., 2022 ; Watts et al., 2022). Bien que des avancées ont été réalisées dans l'apprentissage par renforcement, plusieurs défis restent à relever (Alwarafy et al., 2022 ; Hutsebaut-Buysse et al., 2022 ; Müller et al., 2022). J'envisage dans mes futurs travaux de plus explorer les domaines du deep learning et l'apprentissage par renforcement pour la détection d'anomalies.

Explanable anomalies detection (XAD) : Le processus d'aide à la prise de décision implique trois acteurs principaux : le chercheur invente un algorithme pour le traitement des données. Le data scientist suivant ses données à disposition, les contraintes, ses connaissances à priori sur les données ou sur le domaine

1. La voiture autonome de Amazon. https://aws.amazon.com/fr/deepracer/
2. Apprendre à une voiture la conduite avec l'apprentissage par renforcement https://wayve.ai/

concerné, sélectionne l'algorithme le mieux adapté (classification, prédiction, régression, etc.). Généralement, le data scientist implémente l'algorithme dans l'outil utilisé par l'entreprise et le met donc à la disposition des utilisateurs finaux. Les utilisateurs finaux en question l'appliqueront donc au besoin et obtiendront des résultats. Ces résultats bien que précis sont de plus en plus difficiles à interpréter ou à expliquer pour les utilisateurs finaux d'où l'explanable Artificial Intelligence (XAI) (Sejr et Schneider-Kamp, 2021 ; Krajna et al., 2022). XAI a plusieurs domaines d'application comme l'internet des objets (Kok et al., 2022), la santé (Tjoa et Guan, 2020 ; SM Hussain et al., 2022), etc.

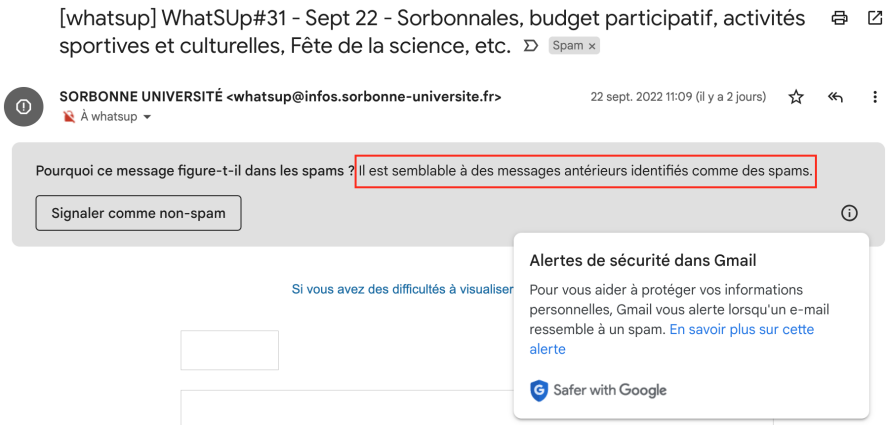


Figure 7.1 – Un mail classé Spam par Gmail et l'explication donnée encadrée en rouge

Le même principe s'applique aux algorithmes de détection d'anomalies dont les résultats deviennent de plus en plus difficiles à interpréter pour les data scientists qui les appliquent en conditions réelles et surtout pour les utilisateurs finaux qui prennent les décisions. Un exemple type est la classification des mails notamment en Spam ou courrier indésirable. Les figures 7.1 et 7.2 montrent deux exemples de mail classés Spam par Gmail et Outlook. Aucune explication formelle n'a été donnée à l'utilisateur concernant la classification. La (Figure 7.1) montre un exemple d'explication très vague.

Explanable Anomalies Detection (XAD) est donc un nouveau sujet d'actualité et important pour réduire le fossé entre les avancées dans le domaine de la détection d'anomalies et les utilisateurs finaux. Bien que quelques travaux ont

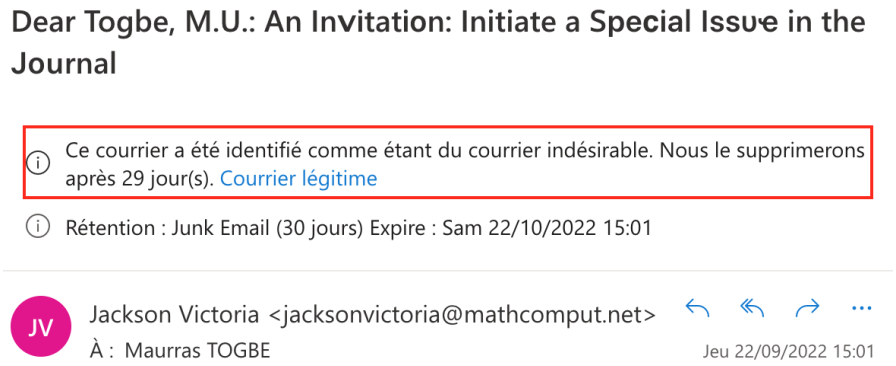


FIGURE 7.2 – Un mail classé indésirable par Outlook. Aucune explication n'a été donnée.

été réalisés dans ce nouveau domaine (ANGIULLI, FASSETTI et al., 2022 ; HUONG et al., 2022), il reste non suffisamment exploré.

Bibliographie

1. Massey Jr Frank J. The Kolmogorov-Smirnov test for goodness of fit. Journal of the American statistical Association 1951 ; 46:68-78 (cf. p. 117)
2. Hawkins Douglas M. Identification of outliers. T. 11. Springer, 1980 (cf. p. 16)
3. Keller James M, Gray Michael R et Givens James A. A fuzzy k-nearest neighbor algorithm. IEEE transactions on systems, man, and cybernetics 1985 :580-5 (cf. p. 28)
4. Barnett V et Lewis T. Outliers in Statistical Data John Wiley and Sons. New York 1994 (cf. p. 2)
5. Ester Martin, Kriegel Hans-Peter, Sander Jörg, Xu Xiaowei et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *Kdd*. T. 96. 1996 :226-31 (cf. p. 21)
6. Desforges MJ, Jacob PJ et Cooper JE. Applications of probability density estimation to the detection of abnormal conditions in engineering. Proceedings of the Institution of Mechanical Engineers, Part C : Journal of Mechanical Engineering Science 1998 ; 212:687-703 (cf. p. 23)
7. Breunig Markus M, Kriegel Hans-Peter, Ng Raymond T et Sander Jörg. LOF : identifying density-based local outliers. *ACM sigmod record*. T. 29. 2. ACM. 2000 :93-104 (cf. p. 24, 28)
8. Lucas James M et Crosier Ronald B. Fast initial response for CUSUM quality-control schemes : give your CUSUM a head start. Technometrics 2000 ; 42:102-7 (cf. p. 23)
9. Schölkopf Bernhard, Williamson Robert C, Smola Alex J, Shawe-Taylor John et Platt John C. Support vector method for novelty detection. *Advances in neural information processing systems*. 2000 :582-8 (cf. p. 26)
10. Hampel Frank R. Robust statistics : A brief introduction and overview. *Research report/Seminar für Statistik, Eidgenössische Technische Hochschule (ETH)*. T. 94. Seminar für Statistik, Eidgenössische Technische Hochschule. 2001 (cf. p. 5)

11. Schölkopf Bernhard, Platt John C, Shawe-Taylor John, Smola Alex J et Williamson Robert C. Estimating the support of a high-dimensional distribution. Neural computation 2001 ; 13:1443-71 (cf. p. 26)

12. Street W Nick et Kim YongSeog. A streaming ensemble algorithm (SEA) for large-scale classification. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001 :377-82 (cf. p. 120)

13. Angiulli Fabrizio et Pizzuti Clara. Fast outlier detection in high dimensional spaces. *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2002 :15-27 (cf. p. 24)

14. Chawla Nitesh V, Bowyer Kevin W, Hall Lawrence O et Kegelmeyer W Philip. SMOTE : synthetic minority over-sampling technique. Journal of artificial intelligence research 2002 ; 16:321-57 (cf. p. 47)

15. Aggarwal Charu C, Han Jiawei, Wang Jianyong et Yu Philip S. A framework for clustering evolving data streams. *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment. 2003 :81-92 (cf. p. 91)

16. Lazarevic Aleksandar, Ertoz Levent, Kumar Vipin, Ozgur Aysel et Srivastava Jaideep. A comparative study of anomaly detection schemes in network intrusion detection. *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM. 2003 :25-36 (cf. p. 35)

17. Gama Joao, Medas Pedro, Castillo Gladys et Rodrigues Pedro. Learning with drift detection. *Brazilian symposium on artificial intelligence*. Springer. 2004 :286-95 (cf. p. 48)

18. Hodge Victoria et Austin Jim. A survey of outlier detection methodologies. Artificial intelligence review 2004 ; 22:85-126 (cf. p. xvii, 20, 89)

19. Motwani Mukesh C, Gadiya Mukesh C, Motwani Rakhi C et Harris Frederick C. Survey of image denoising techniques. *Proceedings of GSPX*. 2004 :27-30 (cf. p. 5)

20. Ramsay James O. Functional data analysis. Encyclopedia of Statistical Sciences 2004 ; 4 (cf. p. 46)

21. Yamanishi Kenji, Takeuchi Jun-Ichi, Williams Graham et Milne Peter. Online unsupervised outlier detection using finite mixtures with discounting learning algorithms. Data Mining and Knowledge Discovery 2004 ; 8:275-300 (cf. p. 21, 23, 24, 91)

22. Torres Jose Luis, Garcia Almudena, De Blas Marian et De Francisco Adolfo. Forecast of hourly average wind speed with ARMA models in Navarre (Spain). Solar Energy 2005 ; 79:65-77 (cf. p. 23)

23. Cao Feng, Estert Martin, Qian Weining et Zhou Aoying. Density-based clustering over an evolving data stream with noise. *Proceedings of the 2006 SIAM international conference on data mining*. SIAM. 2006 :328-39 (cf. p. 21)

24. Rajasegarar Sutharshan, Leckie Christopher, Palaniswami Marimuthu et Bezdek James C. Distributed anomaly detection in wireless sensor networks. *2006 10th IEEE Singapore international conference on communication systems*. IEEE, 2006 :1-5 (cf. p. 136)

25. Abdessalem Talel, Chiky Raja, Hébrail Georges, Vitti Jean Louis et Paris GET-ENST. Traitement de données de consommation électrique par un Système de Gestion de Flux de Données. *EGC*. 2007 :521-32 (cf. p. 82)

26. Aggarwal Charu C. Data streams : models and algorithms. T. 31. Springer Science & Business Media, 2007 (cf. p. 87)

27. Aggarwal Charu C et Philip S Yu. A survey of synopsis construction in data streams. *Data Streams*. Springer, 2007 :169-207 (cf. p. 86)

28. Angiulli Fabrizio et Fassetti Fabio. Detecting distance-based outliers in streams of data. *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM. 2007 :811-20 (cf. p. 91)

29. Bifet Albert et Gavalda Ricard. Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007 :443-8 (cf. p. 10, 107, 114, 115)

30. Chen Yixin et Tu Li. Density-based clustering for real-time stream data. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2007 :133-42 (cf. p. 21)

31. Gama João et Gaber Mohamed Medhat. Learning from data streams : processing techniques in sensor networks. Springer, 2007 (cf. p. 86)

32. Ikonomovska Elena, Loskovska Suzana et Gjorgjevik Dejan. A survey of stream data mining. *ETAI*. 2007 :19-21 (cf. p. 86)

33. Martin RA, Schwabacher Mark, Oza Nikunj et Srivastava Ashok. Comparison of unsupervised anomaly detection methods for systems health management using space shuttle. *Main Engine Data," Proceedings of the Joint Army Navy NASA Air Force Conference on Propulsion, 2007*. Citeseer. 2007 (cf. p. 35)

34. Patcha Animesh et Park Jung-Min. An overview of anomaly detection techniques : Existing solutions and latest technological trends. Computer networks 2007 ; 51:3448-70 (cf. p. xvii, 18, 20)

35. Pokrajac Dragoljub, Lazarevic Aleksandar et Latecki Longin Jan. Incremental local outlier detection for data streams. *2007 IEEE symposium on computational intelligence and data mining*. IEEE. 2007 :504-15 (cf. p. 25, 91)

36. Chhabra Parminder, Scott Clayton, Kolaczyk Eric D et Crovella Mark. Distributed spatial anomaly detection. *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*. IEEE, 2008 :1705-13 (cf. p. 136)

37. Chiky Raja, Dessertaine Alain et Hébrail Georges. Échantillonnage sur les flux de données : état de l'art. Méthodes de sondages 2008 :314-8 (cf. p. 83)

38. Liu Fei Tony, Ting Kai Ming et Zhou Zhi-Hua. Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008 :413-22 (cf. p. 8, 27, 28, 33, 35, 44, 45, 56, 57, 90, 91, 94, 135, 144)

39. Wu Elizabeth, Liu Wei et Chawla Sanjay. Spatio-temporal outlier detection in precipitation data. *International Workshop on Knowledge Discovery from Sensor Data*. Springer. 2008 :115-33 (cf. p. 21)

40. Bifet Albert et Gavaldà Ricard. Adaptive learning from evolving data streams. *Intelligent Data Analysis (IDA)*. Springer. 2009 :249-60 (cf. p. 115)

41. Chandola Varun, Banerjee Arindam et Kumar Vipin. Anomaly detection : A survey. ACM computing surveys (CSUR) 2009 ; 41:15 (cf. p. xvii, 2, 5, 18-21, 89)

42. Ren Jiadong et Ma Ruiqing. Density-based data streams clustering over sliding windows. *2009 Sixth international conference on fuzzy systems and knowledge discovery*. T. 5. IEEE. 2009 :248-52 (cf. p. 21)

43. Bifet Albert, Holmes Geoff, Kirkby Richard et Pfahringer Bernhard. MOA : Massive Online Analysis. J. Mach. Learn. Res. 2010 ; 11:1601-4. Available from: http://portal.acm.org/citation.cfm?id=1859903 (cf. p. 87)

44. Gama Joao. Knowledge discovery from data streams. CRC Press, 2010 (cf. p. 110)

45. Liu Fei Tony, Ting Kai Ming et Zhou Zhi-Hua. On detecting clustered anomalies using SCiForest. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2010 :274-90 (cf. p. 45)

46. Ng Willie et Dash Manoranjan. Discovery of frequent patterns in transactional data streams. *Transactions on large-scale data-and knowledge-centered systems II*. Springer, 2010 :1-30 (cf. p. 83, 87)

47. Das Kamalika, Bhaduri Kanishka et Votava Petr. Distributed anomaly detection using 1-class SVM for vertically partitioned data. Statistical Analysis and Data Mining : The ASA Data Science Journal 2011 ; 4:393-406 (cf. p. 128)

48. Lopes Raul H. C. Kolmogorov-Smirnov Test. *International Encyclopedia of Statistical Science*. Sous la dir. de Lovric Miodrag. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011 :718-20. isbn : 978-3-642-04898-2. doi : 10.1007/978-3-642-04898-2_326. Available from: https://doi.org/10.1007/978-3-642-04898-2_326 (cf. p. 117)

49. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M. et Duchesnay E. Scikit-learn : Machine Learning in Python. Journal of Machine Learning Research 2011 ; 12:2825-30 (cf. p. 37)

50. Tan Swee Chuan, Ting Kai Ming et Liu Tony Fei. Fast anomaly detection for streaming data. *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011 (cf. p. 48, 49, 81, 82, 95)

51. Assent Ira, Kranen Philipp, Baldauf Corinna et Seidl Thomas. Anyout : Anytime outlier detection on streaming data. *International Conference on Database Systems for Advanced Applications*. Springer. 2012 :228-42 (cf. p. 91)

52. Gupta Manish, Gao Jing, Sun Yizhou et Han Jiawei. Integrating community matching and outlier detection for mining evolutionary community outliers. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012 :859-67 (cf. p. 21)

53. Hoens T Ryan, Polikar Robi et Chawla Nitesh V. Learning from streaming data with concept drift and imbalance : an overview. Progress in Artificial Intelligence 2012 ; 1:89-101 (cf. p. 110)

54. Liu Fei Tony, Ting Kai Ming et Zhou Zhi-Hua. Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data (TKDD) 2012 ; 6:3 (cf. p. 27, 28, 33, 44, 45, 56, 94, 105)

55. Unkel Steffen, Farrington C Paddy, Garthwaite Paul H, Robertson Chris et Andrews Nick. Statistical methods for the prospective detection of infectious disease outbreaks : a review. Journal of the Royal Statistical Society : Series A (Statistics in Society) 2012 ; 175:49-82 (cf. p. 23)

56. Zeng Li, Li Ling, Duan Lian, Lu Kevin, Shi Zhongzhi, Wang Maoguang, Wu Wenjuan et Luo Ping. Distributed data mining : a survey. Information Technology and Management 2012 ; 13:403-9 (cf. p. 128)

57. Ding Zhiguo et Fei Minrui. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proceedings Volumes 2013 ; 46:12-7. doi : https://doi.org/10.3182/20130902-3-CN-3020.00044 (cf. p. xx, 10, 35, 48, 49, 56, 81, 82, 94, 95, 99, 105, 113, 116, 121, 145)

58. Frénay Benoit et Verleysen Michel. Classification in the presence of label noise : a survey. IEEE transactions on neural networks and learning systems 2013 ; 25:845-69 (cf. p. 5)

59. Zhang Ji. Advancements of outlier detection : A survey. ICST Transactions on Scalable Information Systems 2013 ; 13:1-26 (cf. p. xvii, 20, 21)

60. Bifet A. et Morales G. D. F. Big Data Stream Learning with SAMOA. *2014 IEEE International Conference on Data Mining Workshop*. 2014 :1199-202. doi : 10.1109/ICDMW.2014.24 (cf. p. 88)

61. Chabchoub Yousra, Chiky Raja et Dogan Betul. How can sliding HyperLogLog and EWMA detect port scan attacks in IP traffic ? EURASIP Journal on Information Security 2014 ; 2014:5. issn : 1687-417X. doi : 10.1186/1687-417X-2014-5. Available from: https://doi.org/10.1186/1687-417X-2014-5 (cf. p. 23)

62. Dunning Ted et Friedman Ellen. Practical machine learning : a new look at anomaly detection. " O'Reilly Media, Inc.", 2014 (cf. p. 16)

63. Gama João, Žliobaitė Indrė, Bifet Albert, Pechenizkiy Mykola et Bouchachia Abdelhamid. A survey on concept drift adaptation. ACM computing surveys (CSUR) 2014 ; 46:1-37 (cf. p. 48, 86, 112, 114)

64. Gupta Manish, Gao Jing, Aggarwal Charu C et Han Jiawei. Outlier detection for temporal data : A survey. IEEE Transactions on Knowledge and Data Engineering 2014 ; 26:2250-67 (cf. p. xvii, 18-21, 90)

65. O'Reilly Colin, Gluhak Alexander, Imran Muhammad Ali et Rajasegarar Sutharshan. Anomaly detection in wireless sensor networks in a non-stationary environment. IEEE Communications Surveys & Tutorials. 2014 ; 16. Publisher : IEEE:1413-32 (cf. p. 135)

66. Salehi Mahsa, Leckie Christopher A, Moshtaghi Masud et Vaithianathan Tharshan. A relevance weighted ensemble model for anomaly detection in switching data streams. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2014 :461-73 (cf. p. 21)

67. Solaimani Mohiuddin, Iftekhar Mohammed, Khan Latifur, Thuraisingham Bhavani et Ingram Joey Burton. Spark-based anomaly detection over multi-source VMware performance data in real-time. *IEEE symposium on computational intelligence in cyber security (CICS)*. IEEE. 2014 :1-8 (cf. p. 128)

68. Ted Dunning et Ellen Friedman. Practical machine learning : a new look at anomaly detection. Sous la dir. de Media O'Reilly. O'Reilly Media, 2014 (cf. p. 2)

69. Wu Ke, Zhang Kun, Fan Wei, Edwards Andrea et Philip S Yu. RS-Forest : A rapid density estimator for streaming anomaly detection. *2014 IEEE International Conference on Data Mining*. IEEE. 2014 :600-9 (cf. p. 48)

70. Abadi Martín, Agarwal Ashish, Barham Paul, Brevdo Eugene, Chen Zhifeng, Citro Craig, Corrado Greg S., Davis Andy, Dean Jeffrey, Devin Matthieu, Ghemawat Sanjay, Goodfellow Ian, Harp Andrew, Irving Geoffrey, Isard Michael, Jozefowicz Rafal, Jia Yangqing, Kaiser Lukasz, Kudlur Manjunath, Levenberg Josh, Mané Dan, Schuster Mike, Monga Rajat, Moore Sherry, Murray Derek, Olah Chris, Shlens Jonathon, Steiner Benoit, Sutskever Ilya, Talwar Kunal, Tucker Paul, Vanhoucke Vincent, Vasudevan Vijay, Viégas Fernanda, Vinyals Oriol, Warden Pete, Wattenberg Martin, Wicke Martin, Yu Yuan et Zheng Xiaoqiang. TensorFlow, Large-scale machine learning on heterogeneous systems. 2015. doi : 10.5281/zenodo.4724125 (cf. p. 130)

71. Bifet Albert, Maniu Silviu, Qian Jianfeng, Tian Guangjian, He Cheng et Fan Wei. StreamDM : Advanced Data Mining in Spark Streaming. *International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2015. doi : 10.1109/ICDMW.2015.140. Available from: https://hal.inria.fr/hal-01270606 (cf. p. 88)

72. Chen Wo-Ruo, Yun Yong-Huan, Wen Ming, Lu Hong-Mei, Zhang Zhi-Min et Liang Yi-Zeng. Representative subset selection and outlier detection via isolation forest. Analytical methods 2016; 8:7225-31 (cf. p. 47)

73. D'Urso Ciro. EXPERIENCE : glitches in databases, how to ensure data quality by outlier detection techniques. Journal of Data and Information Quality (JDIQ) 2016; 7:1-22 (cf. p. 5)

74. Goldstein Markus et Uchida Seiichi. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PloS one 2016; 11:e0152173 (cf. p. 28, 35)

75. Haas Peter J. Data-stream sampling : basic techniques and results. Data stream management 2016 :13-44 (cf. p. 83)

76. Lee Jihwan et Cho Nam-Wook. Fast outlier detection using a grid-based algorithm. PloS one 2016 ; 11:e0165972 (cf. p. 26)

77. Meng Xiangrui, Bradley Joseph, Yavuz Burak, Sparks Evan, Venkataraman Shivaram, Liu Davies, Freeman Jeremy, Tsai DB, Amde Manish, Owen Sean et al. Mllib : Machine learning in apache spark. The Journal of Machine Learning Research 2016 ; 17:1235-41 (cf. p. 131)

78. Salehi Mahsa, Leckie Christopher, Bezdek James C, Vaithianathan Tharshan et Zhang Xuyun. Fast memory efficient local outlier detection in data streams. IEEE Transactions on Knowledge and Data Engineering 2016 ; 28:3246-60 (cf. p. 25)

79. Salloum Salman, Dautov Ruslan, Chen Xiaojun, Peng Patrick Xiaogang et Huang Joshua Zhexue. Big data analytics on Apache Spark. International Journal of Data Science and Analytics 2016 ; 1:145-64 (cf. p. 131)

80. Souiden Imen, Brahmi Zaki et Toumi Hajer. A survey on outlier detection in the context of stream mining : review of existing approaches and recommadations. *International Conference on Intelligent Systems Design and Applications*. Springer. 2016 :372-83 (cf. p. xvii, 18, 20)

81. Thakkar Pooja, Vala Jay et Prajapati Vishal. Survey on outlier detection in data stream. Int. J. Comput. Appl 2016 ; 136:13-6 (cf. p. 18, 90)

82. Aggarwal Charu C. Outlier Analysis. Second Edition. Springer International Publishing AG 2017, 2017. isbn : 978-3-319-47577-6. doi : 10.1007/978-3-319-47578-3 (cf. p. xvii, 2, 5, 18-21, 23, 89, 90)

83. Behera R. K., Das S., Jena M., Rath S. K. et Sahoo B. A Comparative Study of Distributed Tools for Analyzing Streaming Data. *International Conference on Information Technology (ICIT)*. 2017 :79-84 (cf. p. 88)

84. Dua Dheeru et Graff Casey. UCI Machine Learning Repository. 2017. Available from: http://archive.ics.uci.edu/ml (cf. p. 34, 99, 120)

85. Gomes Heitor Murilo, Barddal Jean Paul, Enembreck Fabricio et Bifet Albert. A survey on ensemble learning for data stream classification. Computing Surveys (CSUR) 2017 ; 50:23 (cf. p. 115)

86. Gomes Heitor Murilo, Bifet Albert, Read Jesse, Barddal Jean Paul, Enembreck Fabrício, Pfahringer Bernhard, Holmes Geoff et Abdessalem Talel. Adaptive random forests for evolving data stream classification. Machine Learning 2017 :1-27. doi : 10.1007/s10994-017-5642-8 (cf. p. 89)

87. Marteau Pierre-François, Soheily-Khah Saeid et Béchet Nicolas. Hybrid Isolation Forest-Application to Intrusion Detection. arXiv preprint arXiv :1705.03800 2017 (cf. p. 46)

88. Witten Ian H., Frank Eibe, Hall Mark A. et Pal Christopher J. Data Mining : Practical Machine Learning Tools and Techniques. 4e éd. Amsterdam : Morgan Kaufmann, 2017. isbn : 978-0-12-804291-5 (cf. p. 87)

89. Xu Dong, Wang Yanjun, Meng Yulong et Zhang Ziying. An Improved Data Anomaly Detection Method Based on Isolation Forest. *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*. T. 2. IEEE. 2017 :287-91 (cf. p. 46)

90. Zhang Xuyun, Dou Wanchun, He Qiang, Zhou Rui, Leckie Christopher, Kotagiri Ramamohanarao et Salcic Zoran. LSHiForest : a generic framework for fast tree isolation based ensemble anomaly analysis. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE. 2017 :983-94 (cf. p. 47)

91. Bifet Albert, Gavaldà Ricard, Holmes Geoff et Pfahringer Bernhard. Machine Learning for Data Streams with Practical Examples in MOA. https://moa.cms.waikato.ac.nz/book/. MIT Press, 2018 (cf. p. 87, 88)

92. Dia A. Fall, Togbe M. Ulbricht, Boly A., Kasi-Aoul Z. et Metais E. Efficient Graph-Oriented Summary for Optimized Resource Description Framework Streams Processing Using Extended Centrality Measures. *ICDM 2018 : 20th International Conference on Data Mining*. Istanbul, Turkey, 2018 :1430-41 (cf. p. 83)

93. Domingues Rémi, Filippone Maurizio, Michiardi Pietro et Zouaoui Jihane. A comparative evaluation of outlier detection algorithms : Experiments and analyses. Pattern Recognition 2018 ; 74:406-21 (cf. p. 21)

94. El Sibai Rayane, Chabchoub Yousra et Fricker Christine. Using spatial outliers detection to assess balancing mechanisms in bike sharing systems. *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE. 2018 :988-95 (cf. p. 21)

95. Hariri Sahand et Kind Matias Carrasco. Batch and online anomaly detection for scientific applications in a Kubernetes environment. *Proceedings of the 9th Workshop on Scientific Cloud Computing*. 2018 :1-7 (cf. p. 49)

96. Hariri Sahand, Kind Matias Carrasco et Brunner Robert J. Extended Isolation Forest. arXiv preprint arXiv :1811.02141 2018 (cf. p. 44, 45, 51, 55)

97. Hariri Sahand, Kind Matias Carrasco et Brunner Robert J. Extended Isolation Forest. arXiv preprint arXiv :1811.02141 2018 (cf. p. 94)

98. Liao Liefa et Luo Bin. Entropy isolation forest based on dimension entropy for anomaly detection. *International Symposium on Intelligence Computation and Applications*. Springer. 2018 :365-76 (cf. p. 46)

99. Lu Jie, Liu Anjin, Dong Fan, Gu Feng, Gama Joao et Zhang Guangquan. Learning under concept drift : A review. IEEE Transactions on Knowledge and Data Engineering 2018 ; 31:2346-63 (cf. p. 112)

100. Luo Tie et Nagarajan Sai G. Distributed anomaly detection using autoencoder neural networks in wsn for iot. *2018 ieee international conference on communications (icc)*. IEEE, 2018 :1-6 (cf. p. 136)

101. Manapragada Chaitanya, Webb Geoffrey I. et Salehi Mahsa. Extremely Fast Decision Tree. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom : Association for Computing Machinery, 2018. isbn : 9781450355520. Available from: https://doi.org/10.1145/3219819.3220005 (cf. p. 89)

102. Montiel Jacob, Read Jesse, Bifet Albert et Abdessalem Talel. Scikit-Multiflow : A Multi-output Streaming Framework. Journal of Machine Learning Research 2018 ; 19:1-5. Available from: http://jmlr.org/papers/v19/18-251.html (cf. p. 10, 49, 81, 88, 115)

103. Ramotsoela Daniel, Abu-Mahfouz Adnan et Hancke Gerhard. A survey of anomaly detection in industrial wireless sensor networks with critical water system infrastructure as a case study. Sensors. 2018 ; 18. Publisher : Multidisciplinary Digital Publishing Institute:2491 (cf. p. 135)

104. Rousseeuw Peter J et Hubert Mia. Anomaly detection by robust statistics. Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery 2018 ; 8:e1236 (cf. p. 5)

105. Salehi Mahsa et Rashidi Lida. A survey on anomaly detection in evolving data :[with application to forest fire risk prediction]. ACM SIGKDD Explorations Newsletter 2018 ; 20:13-23 (cf. p. xvii, 18-21, 90, 92)

106. Sergeev Alexander et Balso Mike Del. Horovod : fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv :1802.05799 2018 (cf. p. 130)

107. Tao Xiaoling, Peng Yang, Zhao Feng, Zhao Peichao et Wang Yong. A parallel algorithm for network traffic anomaly detection based on Isolation Forest. International Journal of Distributed Sensor Networks. 2018 ; 14. Publisher : SAGE Publications Sage UK : London, England:1550147718814471 (cf. p. 136)

108. Tellis Venisha Maria et D'Souza Divya Jennifer. Detecting Anomalies in Data Stream Using Efficient Techniques : A Review. *2018 International Conference on Control, Power, Communication and Computing Technologies (ICCPCCT)*. IEEE. 2018 :296-8 (cf. p. xvii, 18-20, 90)

109. Wang Chundong, Zhao Zhentang, Gong Liangyi, Zhu Likun, Liu Zheli et Cheng Xiaochun. A distributed anomaly detection system for in-vehicle network using HTM. IEEE Access 2018 ; 6:9091-8 (cf. p. 128)

110. Yang Fangzhou et contributors. FDIForest. 2018. Available from: https://github.com/titicaca/spark-iforest (cf. p. 128, 135, 136, 146)

111. Ahmed Mohiuddin. Data summarization : a survey. Knowledge and Information Systems 2019 ; 58:249-73 (cf. p. 83)

112. Chalapathy Raghavendra et Chawla Sanjay. Deep learning for anomaly detection : A survey. arXiv preprint arXiv :1901.03407 2019 (cf. p. xvii, 19-21, 23)

113. Cortes David. Distance approximation using isolation forests. arXiv preprint arXiv :1910.12362 2019 (cf. p. 46)

114. Dai Jason (Jinquan), Wang Yiheng, Qiu Xin, Ding Ding, Zhang Yao, Wang Yanzhang, Jia Xianyan, Zhang Li (Cherry), Wan Yan, Li Zhichao, Wang Jiao, Huang Shengsheng, Wu Zhongyuan, Wang Yang, Yang Yuhao, She Bowen, Shi Dongjie, Lu Qi, Huang Kai et Song Guoqiong. BigDL : A Distributed Deep Learning Framework for Big Data. *Proceedings of the ACM Symposium on Cloud Computing*. SoCC'19. Association for Computing Machinery, 2019 :50-60. doi : 10.1145/3357223.3362707. Available from: https://arxiv.org/pdf/1804.05839.pdf (cf. p. 130)

115. Kwon Donghwoon, Kim Hyunjoo, Kim Jinoh, Suh Sang C, Kim Ikkyun et Kim Kuinam J. A survey of deep learning-based network anomaly detection. Cluster Computing 2019 ; 22:949-61 (cf. p. 23)

116. LDIForest. 2019. Available from: https://github.com/linkedin/isolation-forest (cf. p. 128, 135, 136, 146)

117. Maronna Ricardo A, Martin R Douglas, Yohai Victor J et Salibián-Barrera Matıas. Robust statistics : theory and methods (with R). John Wiley & Sons, 2019 (cf. p. 5)

118. Mishra Supriya et Chawla Meenu. A comparative study of local outlier factor algorithms for outliers detection in data streams. *Emerging Technologies in Data Mining and Information Security*. Springer, 2019 :347-56 (cf. p. 21)

119. Paszke Adam, Gross Sam, Massa Francisco, Lerer Adam, Bradbury James, Chanan Gregory, Killeen Trevor, Lin Zeming, Gimelshein Natalia, Antiga Luca, Desmaison Alban, Kopf Andreas, Yang Edward, DeVito Zachary, Raison Martin, Tejani Alykhan, Chilamkurthy Sasank, Steiner Benoit, Fang Lu, Bai Junjie et Chintala Soumith. PyTorch : An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*. Sous la dir. de Wallach H., Larochelle H., Beygelzimer A., d'Alché-Buc F., Fox E. et Garnett R. Curran Associates, Inc., 2019 :8024-35. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cf. p. 130)

120. Staerman Guillaume, Mozharovskyi Pavlo, Clémençon Stephan et d'Alché-Buc Florence. Functional Isolation Forest. 2019. arXiv : 1904.04573 [stat.ML] (cf. p. 46, 94)

121. Sublime Jérémie et Kalinicheva Ekaterina. Automatic post-disaster damage mapping using deep-learning techniques for change detection : Case study of the Tohoku tsunami. Remote Sensing 2019 ; 11:1123 (cf. p. 3)

122. Wang Yu Emma, Wei Gu-Yeon et Brooks David. Benchmarking TPU, GPU, and CPU platforms for deep learning. arXiv preprint arXiv :1907.10701 2019 (cf. p. 130)

123. Zheng Yifeng, Li Guohe et Zhang Teng. An Improved Over-sampling Algorithm based on iForest and SMOTE. *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. 2019 :75-80 (cf. p. 47)

124. Bogatinovski Jasmin et Nedelkoski Sasho. Multi-source anomaly detection in distributed it systems. *International Conference on Service-Oriented Computing*. Springer. 2020 :201-13 (cf. p. 128)

125. Halford Max, Bolmier Geoffrey, Sourty Raphael, Vaysse Robin et Zouitine Adil. creme, a Python library for online machine learning. Version 0.6.1. 2020. Available from: https://github.com/MaxHalford/creme (cf. p. 88)

126. Mohotti Wathsala Anupama et Nayak Richi. Efficient outlier detection in text corpus using rare frequency and ranking. ACM Transactions on Knowledge Discovery from Data (TKDD) 2020 ; 14:1-30 (cf. p. 22)

127. Raab Christoph, Heusinger Moritz et Schleif Frank-Michael. Reactive Soft Prototype Computing for Concept Drift Streams. Neurocomputing 2020 (cf. p. 10, 114, 117, 118)

128. Tjoa Erico et Guan Cuntai. A survey on explainable artificial intelligence (xai) : Toward medical xai. IEEE transactions on neural networks and learning systems 2020 ; 32:4793-813 (cf. p. 150)

129. Togbe Maurras Ulbricht, Chabchoub Yousra, Boly Aliou et Chiky Raja. Etude comparative des méthodes de détection d'anomalies. Revue des Nouvelles Technologies de l'Information 2020 ; Extraction et Gestion des Connaissances , RNTI-E-36:109-20 (cf. p. 92, 106)

130. Verbraeken Joost, Wolting Matthijs, Katzy Jonathan, Kloppenburg Jeroen, Verbelen Tim et Rellermeyer Jan S. A survey on distributed machine learning. ACM Computing Surveys (CSUR) 2020 ; 53:1-33 (cf. p. 129, 131)

131. Wang Ruoying, Nie Kexin, Wang Tie, Yang Yang et Long Bo. Deep learning for anomaly detection. *Proceedings of the 13th international conference on web search and data mining*. 2020 :894-6 (cf. p. 23)

132. Wankhede Pallavi, Talati Minaiy et Chinchamalatpure Rutuja. Comparative study of cloud platforms-microsoft azure, google cloud platform and amazon EC2. J. Res. Eng. Appl. Sci 2020 ; 5:60-4 (cf. p. 130)

133. Agrahari Supriya et Singh Anil Kumar. Concept drift detection in data stream mining : A literature review. Journal of King Saud University-Computer and Information Sciences 2021 (cf. p. 112)

134. Asad Muhammad, Moustafa Ahmed et Ito Takayuki. Federated Learning Versus Classical Machine Learning : A Convergence Comparison. arXiv :2107.10976 2021 (cf. p. 128)

135. Gupta Bulbul, Mittal Pooja et Mufti Tabish. A review on Amazon web service (AWS), Microsoft azure & Google cloud platform (GCP) services. 2021 (cf. p. 130)

136. Heigl Michael, Anand Kumar Ashutosh, Urmann Andreas, Fiala Dalibor, Schramm Martin et Hable Robert. On the improvement of the isolation forest algorithm for outlier detection with streaming data. Electronics 2021 ; 10:1534 (cf. p. 48)

137. Kaushik Prakarsh, Rao Ashwin Murali, Singh Devang Pratap, Vashisht Swati et Gupta Shubhi. Cloud Computing and Comparison based on Service and Performance between Amazon AWS, Microsoft Azure, and Google Cloud. *2021 International Conference on Technological Advancements and Innovations (ICTAI)*. 2021 :268-73. doi : 10.1109/ICTAI53825.2021.9673425 (cf. p. 130)

138. Pang Guansong, Hengel Anton van den, Shen Chunhua et Cao Longbing. Toward deep supervised anomaly detection : Reinforcement learning from partially labeled anomaly data. *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 2021 :1298-308 (cf. p. 149)

139. Pang Guansong, Shen Chunhua, Cao Longbing et Hengel Anton Van Den. Deep learning for anomaly detection : A review. ACM Computing Surveys (CSUR) 2021 ; 54:1-38 (cf. p. 23)

140. Sejr Jonas Herskind et Schneider-Kamp Anna. Explainable outlier detection : What, for Whom and Why ? Machine Learning with Applications 2021 ; 6:100172 (cf. p. 150)

141. Shvetsova Nina, Bakker Bart, Fedulova Irina, Schulz Heinrich et Dylov Dmitry V. Anomaly detection in medical imaging with deep perceptual autoencoders. IEEE Access 2021 ; 9:118571-83 (cf. p. 3)

142. Vervaet Arthur. MoniLog : An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures. *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE. 2021 :2739-43 (cf. p. 3)

143. Yu Peipeng, Xia Zhihua, Fei Jianwei et Lu Yujiang. A survey on deepfake video detection. Iet Biometrics 2021 ; 10:607-24 (cf. p. 5)

144. Alwarafy Abdulmalik, Abdallah Mohamed, Ciftler Bekir Sait, Al-Fuqaha Ala et Hamdi Mounir. The frontiers of deep reinforcement learning for resource management in future wireless HetNets : Techniques, challenges, and research directions. IEEE Open Journal of the Communications Society 2022 (cf. p. 149)

145. Angiulli Fabrizio, Fassetti Fabio, Nisticò Simona et Palopoli Luigi. Outlier Explanation Through Masking Models. *European Conference on Advances in Databases and Information Systems*. Springer. 2022 :392-406 (cf. p. 151)

146. Chabchoub Yousra, Togbe Maurras Ulbricht, Boly Aliou et Chiky Raja. An in-depth study and improvement of Isolation Forest. IEEE Access 2022 ; 10:10219-37 (cf. p. 137)

147. Divya D, Marath Bhasi et Kumar MB Santosh. Review of fault detection techniques for predictive maintenance. Journal of Quality in Maintenance Engineering 2022 (cf. p. 3)

148. Haque Nur Imtiazul, Rahman Mohammad Ashiqur et Ahamed Sheikh Iqbal. DeepCAD : A Stand-alone Deep Neural Network-based Framework for Classification and Anomaly Detection in Smart Healthcare Systems. *2022 IEEE International Conference on Digital Health (ICDH)*. IEEE. 2022 :218-27 (cf. p. 3)

149. Hilal Waleed, Gadsden S Andrew et Yawney John. Financial Fraud : : A Review of Anomaly Detection Techniques and Recent Advances. 2022 (cf. p. 3)

150. Hu Jia, Kaur Kuljeet, Lin Hui, Wang Xiaoding, Hassan Mohammad Mehedi, Razzak Imran et Hammoudeh Mohammad. Intelligent anomaly detection of trajectories for IoT empowered maritime transportation systems. IEEE Transactions on Intelligent Transportation Systems 2022 (cf. p. 3)

151. Huong Truong Thu, Bac Ta Phuong, Ha Kieu Ngan, Hoang Nguyen Viet, Hoang Nguyen Xuan, Hung Nguyen Tai et Tran Kim Phuc. Federated Learning-Based Explainable Anomaly Detection for Industrial Control Systems. IEEE Access 2022 ; 10:53854-72 (cf. p. 151)

152. Hussain Naziya, Rani Preeti, Chouhan Harsha et Gaur Urvashi Sharma. Cyber Security and Privacy of Connected and Automated Vehicles (CAVs)-Based Federated Learning : Challenges, Opportunities, and Open Issues. *Federated Learning for IoT Applications*. Springer, 2022 :169-83 (cf. p. 128)

153. Hussain Sardar Mehboob, Buongiorno Domenico, Altini Nicola, Berloco Francesco, Prencipe Berardino, Moschetta Marco, Bevilacqua Vitoantonio et Brunetti Antonio. Shape-Based Breast Lesion Classification Using Digital Tomosynthesis Images : The Role of Explainable Artificial Intelligence. Applied Sciences 2022 ; 12:6230 (cf. p. 150)

154. Hutsebaut-Buysse Matthias, Mets Kevin et Latré Steven. Hierarchical Reinforcement Learning : A Survey and Open Research Challenges. Machine Learning and Knowledge Extraction 2022 ; 4:172-221 (cf. p. 149)

155. Kok Ibrahim, Okay Feyza Yildirim, Muyanli Ozgecan et Ozdemir Suat. Explainable Artificial Intelligence (XAI) for Internet of Things : A Survey. arXiv preprint arXiv :2206.04800 2022 (cf. p. 150)

156. Krajna Agneza, Kovac Mihael, Brcic Mario et Šarčević Ana. Explainable Artificial Intelligence : An Updated Perspective. *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE. 2022 :859-64 (cf. p. 150)

157. Lischke Cameron, Liu Tongtong, McCalmon Joe, Rahman Md Asifur, Halabi Talal et Alqahtani Sarra. LSTM-Based Anomalous Behavior Detection in Multi-Agent Reinforcement Learning. *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE. 2022 :16-21 (cf. p. 149)

158. Mao Weichao, Yang Lin, Zhang Kaiqing et Basar Tamer. On improving model-free algorithms for decentralized multi-agent reinforcement learning. *International Conference on Machine Learning*. PMLR. 2022 :15007-49 (cf. p. 149)

159. Morales Eduardo F et Escalante Hugo Jair. A brief introduction to supervised, unsupervised, and reinforcement learning. *Biosignal Processing and Classification Using Computational Learning and Intelligence*. Elsevier, 2022 :111-29 (cf. p. 149)

160. Müller Robert, Illium Steffen, Phan Thomy, Haider Tom et Linnhoff-Popien Claudia. Towards Anomaly Detection in Reinforcement Learning. *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 2022 :1799-803 (cf. p. 149)

161. Pandey Mohit, Pandey Shubhangi et Kumar Ajit. Introduction to Federated Learning. *Federated Learning for IoT Applications*. Springer, 2022 :1-17 (cf. p. 128)

162. Qasem Mais Haj, Hudaib Amjad, Obeid Nadim, Almaiah Mohammed Amin, Almomani Omar et Al-Khasawneh Ahmad. Multi-agent Systems for Distributed Data Mining Techniques : An Overview. Big Data Intelligence for Smart Applications 2022 :57-92 (cf. p. 128)

163. Ryciak Piotr, Wasielewska Katarzyna et Janicki Artur. Anomaly Detection in Log Files Using Selected Natural Language Processing Methods. Applied Sciences 2022; 12:5089 (cf. p. 3)

164. Samara Mustafa Al, Bennis Ismail, Abouaissa Abdelhafid et Lorenz Pascal. A Survey of Outlier Detection Techniques in IoT : Review and Classification. Journal of Sensor and Actuator Networks. 2022; 11. Number : 1 Publisher : Multidisciplinary Digital Publishing Institute:4. issn : 2224-2708. doi : 10.3390/jsan11010004. Available from: https://www.mdpi.com/2224-2708/11/1/4 [Accessed on: 2022 Mar 11] (cf. p. 135)

165. Sarangi Biswaranjan et Tripathy Biswajit. Outlier detection in Wireless Sensor Network for Health Care : A statistical approach. *2022 International Conference for Advancement in Technology (ICONAT)*. IEEE. 2022 :1-4 (cf. p. 3)

166. Shao Yangyi, Zhang Wenbin, Liu Peishun, Huyue Ren, Tang Ruichun, Yin Qilin et Li Qi. Log Anomaly Detection method based on BERT model optimization. *2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE. 2022 :161-6 (cf. p. 3)

167. Tschuchnig Maximilian E et Gadermayr Michael. Anomaly Detection in Medical Imaging-A Mini Review. Data Science–Analytics and Applications 2022 :33-8 (cf. p. 3)

168. Umer Muhammad Azmi, Junejo Khurum Nazir, Jilani Muhammad Taha et Mathur Aditya P. Machine learning for intrusion detection in industrial control systems : Applications, challenges, and recommendations. International Journal of Critical Infrastructure Protection 2022 :100516 (cf. p. 3)

169. Watts Jeremy, Wyk Franco van, Rezaei Shahrbanoo, Wang Yiyang, Masoud Neda et Khojandi Anahita. A dynamic deep reinforcement learning-Bayesian framework for anomaly detection. IEEE Transactions on Intelligent Transportation Systems 2022 (cf. p. 149)

170. Rauwerda Annie. Are humans better than AI at detecting deepfakes? It's complicated. en. Available from: https://www.inputmag.com/tech/are-humans-better-than-ai-at-detecting-deepfakes [Accessed on: 2022 Aug 15] (cf. p. 5)