



TECHNISCHE
UNIVERSITÄT
WIEN
Vienna | Austria

Fast Internal Relative Evaluation of Outlier Solutions (Fast IREOS)

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Adrian Tobisch, BSc

Matrikelnummer 01227508

an der Fakultät für Informatik

der Technischen Universität Wien


Betreuung: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

Mitwirkung: Senior Scientist Dr.techn. Félix Iglesias Vázquez

Wien, 31. August 2023

Adrian Tobisch

Tanja Zseby

	Signatory	Tanja Zseby
	Date/Time-UTC	2023-09-07T21:33:33+02:00
	Verification	Information about the verification of the electronic signature can be found at: https://www.signaturpruefung.gv.at
Note	This document is signed with a qualified electronic signature. According to Art. 25 para. 2 of the Regulation (EU) No 910/2014 of 23. July 2014 ("eIDAS-Regulation") it shall have the equivalent legal effect of a handwritten signature.	



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



TECHNISCHE
UNIVERSITÄT
WIEN
Vienna | Austria

Fast Internal Relative Evaluation of Outlier Solutions (Fast IREOS)

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Adrian Tobisch, BSc

Registration Number 01227508

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

Assistance: Senior Scientist Dr.techn. Félix Iglesias Vázquez

Vienna, 31st August, 2023

Adrian Tobisch

Tanja Zseby



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Adrian Tobisch, BSc

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 31. August 2023

Adrian Tobisch



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich möchte die Gelegenheit nutzen und mich an dieser Stelle bei den Menschen bedanken, die mich im Laufe meines Studiums unterstützt und auf diesem Weg begleitet haben.

In erster Linie möchte ich mich bei meinem Betreuer Dr.techn. Félix Iglesias bedanken, der mir mit seiner fachlichen Expertise bei jeglicher Problemstellung stets zur Seite stand und jederzeit ein offenes Ohr für Fragen hatte. Weiters möchte ich mich, insbesondere bei meiner Supervisorin, Univ. Prof. Dr.-Ing. Tanja Zseby für die gute Zusammenarbeit und das hilfreiche Feedback bedanken.

Besonderer Dank gebührt meinen Eltern Ilse und Franz und meiner Schwester Sarah, die mich über die Zeit meines gesamten Studiums unterstützt haben. Weiters möchte ich meiner Freundin Selina Brem für Ihre Ermutigung, Ihren Zuspruch und dass Sie mir immer zugehört hat wenn ich über thesirelevante Probleme klagte, danken. Nicht zuletzt gilt mein Dank auch allen Freundinnen und Freunden, die mit mir den Weg durch dieses Studium gegangen sind. Ohne Eure Hilfe wäre mir dieser Lebensabschnitt nicht so gelungen, wie er mir gelungen ist.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to take this opportunity to thank the people who supported me during my studies and accompanied me along the way.

First and foremost, I want to thank my supervisor, Dr.techn. Félix Iglesias, who supported me with his professional expertise in any problem and was always available to answer my questions. Furthermore, I would like to thank my professor and supervisor, Prof. Dr.-Ing. Tanja Zseby, for the excellent cooperation and helpful feedback.

Special thanks go to my parents Ilse und Franz and my sister Sarah, who always supported me throughout my studies. Furthermore, I would like to thank my girlfriend Selina Brem for her encouragement and for always listening to me when talking about thesis-related problems. Last but not least, I also want to thank all my friends who accompanied or supported me during my studies. Without your help, I would not have been as successful in this phase of my life as I have been.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die automatische Erkennung von Ausreißern ist ein wichtiger Schritt während der Datenaufbereitung und -verarbeitung von modernen Anwendungen, die maschinelles Lernen und Data-Mining nutzen. Eine verlässliche Erkennung dieser Datenanomalien spielt eine besonders wichtige Rolle bei Anwendungen in der Medizin, im Finanzwesen und in der Industrie. Aus diesem Grund ist die Weiterentwicklung und Optimierung von effizienten und effektiven Algorithmen zur Ausreißerererkennung wünschenswert.

Die Hauptanwendungsfälle von unüberwachtem Lernen zur Extraktion von unbekanntem Informationen sind einerseits das Clustern, als auch die Erkennung von Ausreißern. Jedoch sind sie oft fehleranfällig, da sie auf Kriterien und Parametern beruhen, die Modelle vorgeben die den Daten nicht entsprechen. Dies macht eine Form von Validierung notwendig, welche entweder extern (ground truth) oder intern (Struktur der Daten) geschehen kann. Obwohl es in der Literatur einige Ressourcen zu interner Validierung für Clustering gibt, so sind diese in der Domäne der Ausreißerererkennung unterrepräsentiert.

Ursprünglich veröffentlicht im Jahr 2015, ist IREOS eine der wenigen existierenden, internen Validierungsmethoden im Gebiet der Ausreißerererkennung. Das Ziel dieser Diplomarbeit ist eine Untersuchung von IREOS bezüglich Performance, sowie Möglichkeiten, um den Algorithmus weiter zu beschleunigen. Die Verwendung verschiedener Klassifikatoren wie lineare und baumartige Klassifikatoren wurde in Bezug auf die ursprüngliche Technik untersucht, indem mehrere Datensets auf einer alternativen Implementierung von IREOS unter Verwendung der Programmiersprache Julia ausgeführt wurden. Neben dem direkten Vergleich mit der Leistung von IREOS, ging es bei dieser Arbeit vor allem darum, die Laufzeit zu verbessern und gleichzeitig die Qualität der Ergebnisse des Algorithmus beizubehalten, indem sichergestellt wird, dass die resultierenden Zahlen innerhalb des Bereichs der von IREOS erzielten Werte bleiben.

Einige der getesteten Klassifikatoren zeigten statistisch gesehen ähnliche Ergebnisse wie die ursprüngliche Implementierung. Der schnellste Klassifikator konnte eine Berechnung für die IREOS über 50 Minuten benötigte, auf etwa 1 Sekunde reduzieren. In unseren Experimenten lieferten insbesondere baumartige Klassifikatoren und nicht-lineare SVMs qualitativ sehr ähnliche und konsistente Ergebnisse, mit nur einem Bruchteil der Ausführungszeit von IREOS. Die Ergebnisse zeigen das Potenzial von weniger komplexen Klassifikatoren, die intern für die Erkennung von Anomalien verwendet werden, und die Einschränkungen in Bezug auf ihre Stabilität bei verschiedenen Parametereinstellungen.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Automatic outlier detection marks an important step in the process of data preparation and processing for modern applications of machine learning and data mining. Accurately identifying and eliminating anomalies within data plays an important role in real-world applications found in medical, financial, and industrial fields. Thus, the development and optimization of efficient and accurate implementations of outlier detection algorithms is highly desired.

Unsupervised learning, either clustering or anomaly/outlier detection, are very common tools to extract a priori unknown information about the data under analysis. However, they often fail, as they are based on criteria or parameters that may impose models that are not natural to the data. Validation is necessary, both externally (i.e., compared with a ground truth) and internally (i.e., based on the inherent properties of data). And, although there are many methods for internal validation of clustering, the internal validation of outlier solutions is still underrepresented in the literature and remains a challenge in the field of data mining.

Originally released in 2015, IREOS is one of the few existing internal validation methods for anomaly detection algorithms. This thesis aims to investigate the behavior and performance of the recently introduced technique IREOS and to find ways to speed up the applied algorithm and implementation thereof. The use of different classifiers such as linear and tree-based classifiers concerning the original technique was studied by running multiple datasets on a self-written alternative implementation of IREOS using the Julia Programming Language. In addition to the direct comparison to the performance of IREOS, the primary focus of this work was to improve already existing evaluations of outlier detection while retaining the quality of the algorithm's output by ensuring the resulting outlier scores remain within the range of those obtained by IREOS.

Under empirical evidence, some of those tested predictors showed statistically similar results to the original implementation. The fastest predictor was able to speed up a calculation IREOS needed over 50 minutes to around 1 second. In our experiments, especially tree-based predictors and nonlinear support vector machines deliver highly similar and consistent results with only a fraction of the execution time of IREOS. Results show the potential of lightweight predictors being used internally for anomaly detection and the limitations in regard to their stability over different parameter settings.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Background	1
1.2 Motivation	3
1.3 Research Questions, Goals and Methodologies	5
1.4 Structure	6
2 Background Knowledge	7
2.1 Outlier Detection	7
2.1.1 Parametric models	8
2.1.2 Non-parametric models	8
2.2 Validation in Outlier Detection	13
2.2.1 External Validation	15
2.2.2 Internal Validation	18
2.2.3 Adjustment for Chance	20
2.3 Internal Relative Evaluation of Outlier Solutions: IREOS	22
3 Methodology and Experiments	29
3.1 Datasets	29
3.1.1 MDCGen Datasets	29
3.2 Data Preprocessing	32
3.2.1 Training Data	32
3.2.2 Outlier Scores	33
3.3 Classification Algorithms and Models	36
3.3.1 Kernel Logistic Regression	37
3.3.2 Support Vector Machines	38
3.3.3 Tree-Based Approaches	41
3.3.4 LibLINEAR	44
3.4 Sampling Methods and Subsetting	44
	xv

3.4.1 Sliding Windows	45
3.4.2 Coresets	46
3.4.3 Microclustering	47
3.5 FIREOS	48
3.5.1 General Characteristics	48
3.5.2 Normalization Interface	51
3.5.3 FIREOS Implementation	55
3.5.4 Caveats and Limitations	59
3.6 Description of Experiments	61
3.6.1 (S1) First Series of Experiments: FIREOS	62
3.6.2 (S2) Second Series of Experiments: IREOS	62
3.6.3 (S3) Third Series of Experiments: External Validation	62
3.6.4 Evaluation of the Experiments	63
3.7 Comparableness with IREOS	64
4 Results and Discussion	67
4.1 Evaluation Performance	67
4.2 Execution Time and Speedup	80
4.3 Discussion	86
5 Conclusions	89
A Appendix	93
A.1 FIREOS Results	93
List of Figures	111
List of Tables	117
Bibliography	119

Introduction

This chapter presents Fast Internal Relative Evaluation of Outlier Solutions (FIREOS), an alternative approach for the internal validation of outlier detection performances. We introduce the related technical background, explain the motivation of our research, and outline the followed methodology and the targeted goals. The chapter finishes with a depiction of the section structure used in this work as well as an outcome for possible extensions in future works.

1.1 Background

The goal of outlier detection is to find data points in data that deviate from expected behavior. It marks a crucial step during the preprocessing phase of the data science process and is needed to handle noisy and faulty samples. Outliers can be classified into different subtypes such as global outliers, contextual outliers, and local outliers [Figure 1.1](#). Global outliers or point anomalies are instances that diverge significantly from the totality of the data [\[Mod16\]](#). Different from global ones, local outliers only show an uncommon deviation to their surrounding points or local group. If multiple outlier instances group together into a collection where each individual sample is no longer anomalous, but the accumulated cluster is, the literature [\[Agg17\]](#) [\[SU12\]](#) terms them as collective outliers.

Detecting local outliers tends to be more difficult than global ones since the optimal number of clusters for the underlying model of data partitioning is difficult to calculate for an arbitrary dataset [\[NCZW18\]](#) [\[DK22\]](#). Besides outlier detection methods that seek anomalies by comparing only data attributes and therefore being unconditional, the literature differentiates between another type of detection that is conditional [\[Hon20\]](#). Such methods rely on certain input conditions which provide further insight on how different partitions sharing a common context behave. Conditional or contextual outliers may be even harder to detect since an optimal weighting between unconditional and

conditional must be known a priori in order to prevent misclassified inliers from being contextual outliers and vice versa.

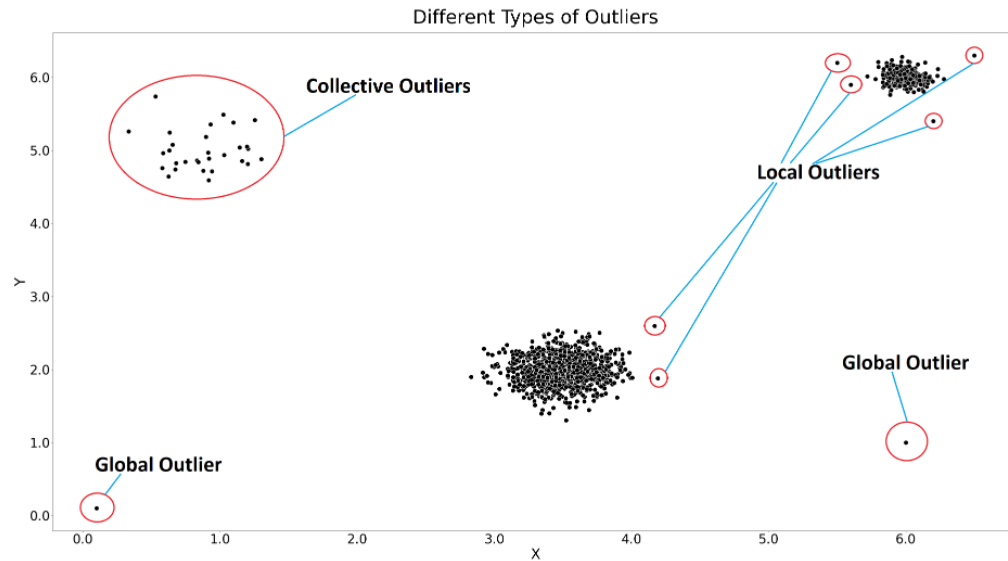


Figure 1.1: Different types of outliers. Contextual outliers are not represented in the figure. Drawn based on [Kha21, Figure 2]

But why is the classification of anomalies that important? Besides their influence in statistical models, anomalies transfer significant and often critical information in a wide range of application domains that can be processed further [CBK09]. Anomaly detection is not only a necessity for cleaning input data but rather the key appliance for many big data applications in the modern world. Monitoring and fraud detection systems mostly rely on the deviation of given instances in relation to common and known cases to detect anomalies. According to PwC's Global Economic Crime and Fraud Survey in 2020 over 42 billion dollars of losses over the time of 2 years were caused by fraudulent incidents [PwC20]. The application area of fraud detection software spans from banking and financial services (credit card fraud, money laundering), IT and telecommunication (phone fraud, network monitoring) to E-Commerce and Retail (delivery fraud, payment fraud, fake reviews) [Kan21]. It is estimated that the area of fraud detection and prevention keeps expanding by an annual growth of 23.2% reaching 190 billion dollars by 2030, according to Straights Research [Res22].

These numbers on their own show the opportunities on how to use anomaly detection directly for real-life use cases – not to mention that nearly every data cleaning process of software utilizing machine learning should feature outlier handling at some point before deployment. Fraud detection is only one of multiple applications where anomaly detection is required. These aspects of handling outliers open various interpretation possibilities. In order to take full advantage of projecting the data onto a model as effectively as

possible, some domain context must be understood by the user. As stated in the previous section the pure discovery of outliers might be the entire problem frame – once they are classified the problem is solved. However, anomalies are often not desired, unintended and lead to methods having less power - hence unreliable results [Val21]. If the outlier is a faulty measurement and just adds disproportional noise, it is better dropped. Does the outlier affect the assumption or just the results? How important is the preservation of a statistical model based on certain assumptions like normality [OST]? Might a gathering of deviating samples result in a new cluster and therefore being a novelty rather than erroneous? In the end, the influence and consequences of anomalies are dependent on the use case and cannot be decided by the method itself. Ruff et al. [RKV⁺20] address this difficult task of differentiating between the terms "Anomaly", "Outlier" and "Novelty" by distinguishing the objective or consequence in an application rather than the detection method itself: Anomalies act as data points of special interest (e.g., long-term survivors of a disease), outliers are treated as "measurement error" or "noise" that is recommended to be removed during preprocessing as it does not serve any beneficial purpose by keeping it, and novelties which act as the "new normal" existing models need to adapt.

Each outlier detection algorithm is constructed on a set of base assumptions that lead to different pictures of what is classified as an anomaly and what is not. This work is aimed to study and make feasible a measurement to evaluate outlier solutions that do not rely on prelabelled samples and is solely based on intrinsic properties of data. Thus, the evaluation is decoupled from application-dependent assumptions. Internal evaluation is not intended to replace external evaluation (only if missing or not available), but to complete it, enrich the interpretation and allow the generalization of the knowledge obtained.

1.2 Motivation

Outlier detection has been a hot topic in the literature for some years and new methods based on various outlier definitions were proposed and further refined over the course of time. Nonetheless, an anomaly detection algorithm that is universally superior to the others does not exist leading to the question of how to measure the "quality" of an outlier solution.

A common validation method in the literature is external or supervised validation which relies on labeled data emerging from empiric observations, that are evaluated against different target models typically favoring the least deviating one. Despite being an effective method for fair evaluation [FGK⁺10] this dependence on the gold standard raises the following question:

Where do these labels come from, and can those be trusted?

Large amounts of labeled data are not available in all domains, which results in high demand for methods that do not rely on them. Unsupervised or internal evaluation focuses merely on the topological and spatial characteristics of the data to implement

a translucent measure of “outlierness”, and thus does not need any external labeling. While unsupervised learning has been widely adopted for clustering and classification, it remains comparatively unnoticed in the area of anomaly detection, which results in a non-negligible gap in the landscape of different evaluation metrics. Internal validation methods such as IREOS should be paid more attention to and developed further to strengthen the trust towards unsupervised learning and unlabelled data.

The second drawback of external validation is potential label bias originating from certain assumptions of the underlying data that favor a specific algorithm. This risk is especially present when the data is labeled by a specific algorithm. Although it remains clear that the total exclusion of any assumption of a given dataset is not possible, a reduction to a neutral baseline should be desired.

But why can eliminating external influence be beneficial in the first place?

Most events do not occur uniformly by nature. An empiric number of observations or unequal probabilities for each outcome creates an uneven distribution of samples for each event. Especially the field of outlier detection suffers from that phenomenon since anomalies tend to be less frequent than inliers by definition. One major pitfall for many machine learning applications is the problem of class representation bias [DHP⁺11] [JALK16] [BDH⁺19]. Improperly treated, severe class imbalances lead to bad-performing models and incorrect conclusions due to the distortions in the data. Both supervised and unsupervised learning approaches are affected by class representation, but differently. A supervised learning algorithm may not identify anomalies, noise or novelties as something different to what has already been learned. On the other hand, unsupervised learning also faces these issues when dealing with clusters of different density and cardinality.

Besides technical aspects, the implementation of unsupervised learning models is also beneficial to economic factors, such as monetary costs. Labeled data often has to be acquired manually by asking a domain expert. Domain experts of important knowledge tend to be expensive, not always available and moreover still transfer personal bias into the data. In a supervised learning scenario, a possible method to mitigate the latter is the involvement of multiple experts resulting in even more costs. Unsupervised learning serves as a good addition to labeled data since it minimizes personal bias about the data [Li17]. Furthermore, an unsupervised model can be helpful by creating ground truth and an additional backbone to possible labels. In conclusion, internal validation provides a vital contrast and complementary knowledge to the most commonly used external validation and, therefore deserves particular attention.

Lastly, one of the main aims of this work is to speed up an internal evaluation algorithm that is computationally expensive. As a computer scientist, I aim to simplify complex problems and mitigate worst-case scenarios. Machine learning algorithms should be optimized where possible to reduce environmental costs, such as energy consumption and computational hardware, needed for operating big data applications as well as increasing the incentive for people to participate and enrich the data science community by making

it possible to run algorithms on consumer machines. Since the “free lunch is over” and major processing manufacturers lack the space for significantly boosting single-core performance [Sut05], it should be in everyone’s interest to utilize multicore devices as much as possible.

One groundbreaking reason why unsupervised learning is not adopted more frequently in real-life applications is that it often results in a lack of control when evaluating procedures. The absence of a ground truth related to the underlying input data often makes it seem unreliable or prone to failure. In addition to the mere possibility of malfunctioning, we may not even realize when and why errors happen, which casts an even more problematic light on the situation. But all of this makes it so crucial to invest resources into improving internal evaluation to make it more robust and expressive to pave the way for better integration in real-world systems.

1.3 Research Questions, Goals and Methodologies

This thesis deals with setting up a more lightweight implementation by extending the already established internal evaluation method IREOS and measuring its performance. The main focus of this work is twofold.

The first goal that is aimed to be achieved is to reduce the computational complexity of the algorithm and reach a **faster implementation of the original IREOS (G1)** implementation by Marquez et al. [Mar20]. We want to provide an answer to our first research question:

Which components of IREOS can be replaced by equivalent methods to obtain faster performances and minimally affecting accuracy? (RQ1)

To reach this goal we: (1) change the programming language into a faster one and provide both a sequential and parallel implementation; Furthermore, (2) we conduct structural adjustments on the algorithm to accelerate the calculation with the main focus on replacements of the internal predictor; Especially (3) less complex predictors such as Support Vector Machines, which are particularly suggested in the original paper due to their nonlinear and soft margin properties, tree-based approaches and linear classifiers are tested during the experiments; Besides replacing the algorithm (4), we also leave out adaptive quadrature for each element and reduce the input space by sliding windows, which both promise a significant speedup because each separation relies on at least three trained predictors; Finally, (5) we run both implementations on the same datasets in the same environment and statistically compare their runtimes.

Furthermore, in order to ensure the quality of the new implementation, it should remain consistent when compared to other evaluation metrics. The second goal is to **evaluate to which degree IREOS and FIREOS are consistent with existing external validation metrics. (G2)** In this respect, one of the main challenges is if different predictors that do not feature nonlinear and soft margin properties are able to retain the quality of assessing different solutions when compared to both the original IREOS

and state-of-the-art external evaluation metrics. These external evaluation metrics are mentioned in Section 2.2 and further explained in Section 2.2.1. All this amounts to answering the following second research question:

How does IREOS/FIREOS align with external evaluation metrics and, consequently, which type of outlieriness definition is favored by IREOS based on empirical evidence? (RQ2)

We reach this goal by: (1) a selection of classifiers that promise interesting results, due to their ability of being fast, non-linear and stable when sampled into forests or boosted; Apart from that (2), we do further finetuning by performing tests of different hyperparameter settings that are (i) inspired by those from the original implementation of (ii) recommended defaults by the underlying library itself; Ultimately, (3) we contrast results coming from FIREOS against (i) the solutions of the original IREOS of the Java implementation and (ii) existing external state-of-the-art metrics, to prove if there is a certain amount of consensus between them.

Additionally, this work comes with a new software module called FIREOS (Fast-IREOS) that contains all modifications and possible options tested in the experiments. To point out the advantages and disadvantages of several estimators in terms of parallelization in detail, a multithreaded implementation containing all features is delivered as well. All parts of the program are targeted to be reproducible and clearly arranged, which is the reason to include another layer of abstraction for possible usage in the form of a command line interface.

In the end, FIREOS is aimed to be a public library that is simple, intuitive, and able to be used out-of-the-box.

1.4 Structure

This work is organized into the following chapters.

- Chapter 1: Gives a brief introduction to the reader
- Chapter 2: Focuses on providing more in-depth knowledge to perform later experiments. Furthermore, an introduction to outlier detection and state-of-the-art algorithms and techniques are presented.
- Chapter 3: Serves as one of the main chapters describing the methodology and composition of the experiments in more detail. Besides the setup of the overall project, a detailed explanation of the new implementation is provided.
- Chapter 4: Summarizes the results of the experiments that were presented previously.
- Chapter 5: Draws a conclusion from the results and discusses possible further work in the future.

Background Knowledge

In this chapter, we want to make the reader familiar with the main outlier detection algorithms that are later covered in the experiments as well as some other recent approaches. We present different validation techniques in unsupervised learning and the current state-of-the-art of outlier detection to prepare the reader for the experiments later on. We finish this chapter with a more detailed elaboration of IREOS, which serves as the foundation for all adjustments that ultimately lead to the development of FIREOS.

2.1 Outlier Detection

Outlier detection is a composition of different aspects spanning from anomaly detection, novelty detection, noise detection, deviation detection and exception mining [HA04]. Since their usage is tightly interconnected to the context of the problem definition and most of them differ only by their conclusions and post-processing, this work uses exclusively the term “outlier detection” for simplification. According to Grubbs [Gru69] “an outlying observation, or “outlier”, is one that appears to deviate markedly from other members of the sample in which it occurs.”

In a machine learning setting, outlier detection can be explained as a classification problem, where an anomaly detection algorithm creates a model from input data to cast predictions on new samples if they are an outlier or not. Since the categorization of predictions consists of two possible outcomes, outlier detection is a binary classification problem and its methods are procedures that are able to classify outlying observations from a population. More complex models do not only predict those labels but also provide the probability of being an outlier. Outlier detection algorithms can be broadly divided between parametric (statistical) and non-parametric (model-free) methods that are following different paradigms [WBH⁺02] [BG05].

2.1.1 Parametric models

Parametric models consist of a finite set of parameters that are used to create a model from a given population [Ayu21][Ani12]. They either assume the underlying distribution of given observations or are based on statistical estimates of unknown distribution parameters [BG05]. Since many distribution functions are univariate and only have a few degrees of freedom they tend to be unsuited for multivariate as well as high-dimensional datasets [PKG03]. One of the most common parametric anomaly detection methods is the usage of Z-scores. Z-scores are using the property that if X is distributed as $X \sim \mathcal{N}(\mu, \sigma^2)$, then $Z = \frac{X-\mu}{\sigma}$ is distributed as $N(0, 1)$ [Hoa13]. The literature often refers to 3 as the outlier threshold for Z-Scores. In other words, each value exceeding a Z-Score of this threshold should be classified as an outlier. Although the general intuition of Z-scores is effective, using the standard deviation as an outlier detection method results in many false positives and should be replaced by the Median Absolute Deviation (MAD) [Ley13]. As an estimator, the sample median provides high robustness since it does not react as sensitive to data changes as the arithmetic mean. This adjusted technique of using MAD as a modified Z-score recommends a threshold of 3.5 for outliers [Hoa13]. Figure 2.1 shows the differences of Z-scores and modified Z-scores of the same underlying data.

2.1.2 Non-parametric models

Nonparametric models on the other hand are not specified a priori but instead directly determined from the observed data [Sal17].

For example, Local Outlier Factor (LOF) [BKNS00] uses the local density of each point to identify uncommon observations that diverge from the majority of the data. These densities yield into the LOF of each point which is computed by their nearest neighbors. Points having the lowest local densities are considered to be outliers. Since the calculation of local densities for n points requires $n \cdot k$ -nearest neighbor searches its computation is exceptionally expensive [YCR17][Tok22]. Figure 2.2 visualizes LOF for a two-dimensional problem and provides example scores on some of the data points.

Differently from LOF, other outlier detection methods in the literature use measures of distance [Figure 2.3] to differentiate between similar and dissimilar. These techniques are also referred to as distance-based methods and are also nonparametric [AP02]. As Knorr et al. introduced them in 1998 [KN98], distance-based outliers are defined as follows:

“An object O in a dataset is a $DB(p, D)$ -outlier if at least a fraction p of the other objects in the dataset lies greater than distance D from O ”.

Although being one of the more intuitive approaches, distance-based measures suffer from finding a fitting distance metric for a given problem [RRS00]. Furthermore, it is particularly strongly affected by “the curse of dimensionality” since the algorithm is exponential in the number of dimensions [AP02].

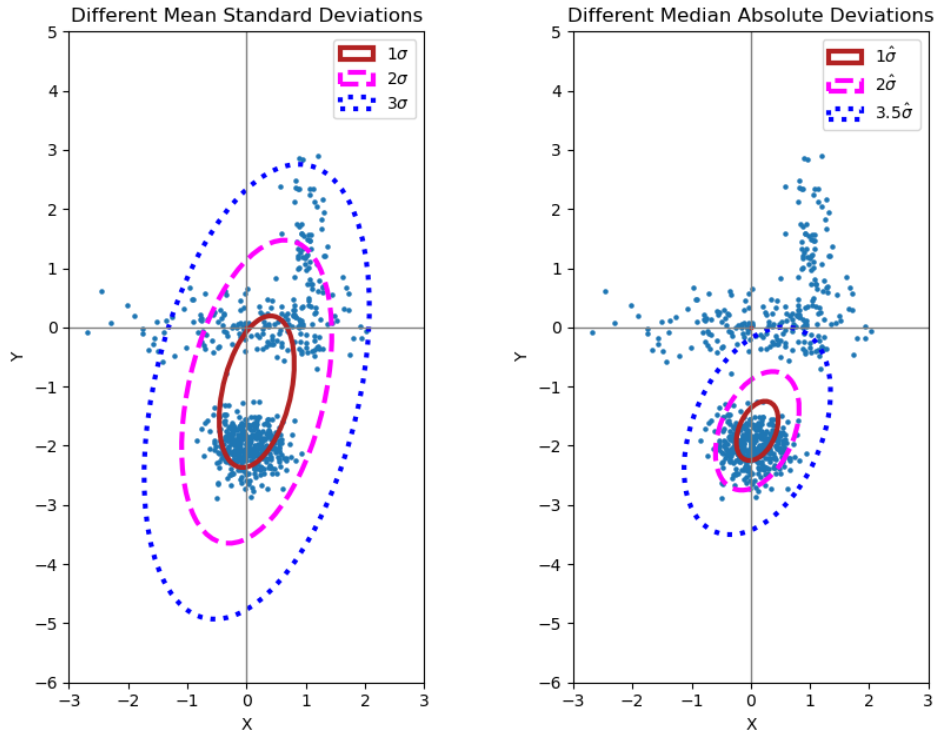


Figure 2.1: Outlier detection by Z-Scores and modified Z-Scores. Each data point outside the outermost blue ellipsis is considered an outlier. The underlying code is inspired by [\[Mis23f\]](#)

Nearest-neighbor-based techniques combine both principles from above by comparing the distances of the KNN. Ramaswamy et al. [\[RRS00\]](#) propose to compare the sums of all nearest neighbor distances for each point and sort them descending to their distance. The first N points are considered to be outliers. [\[HPN11\]](#)

An alternative approach has been made by introducing angle-based outlier detection [Figure 2.4](#) or ABOD respectively. Instead of using local densities or distances, the main assumption is that the variance of angles between different pairs of points tends to be less for outliers than for inliers [\[KSZ08\]](#) [\[PP12\]](#).

Although the general concept of ABOD has a time-complexity of $O(n^3)$, which is considerably worse than $O(n^2 + k)$ of LOF the original paper by Kriegel et al. [\[KSZ08\]](#) proposes a faster approximation (FastABOD) that reduces its complexity to $O(n^2 + n \cdot k^2)$. The robustness of ABOD in datasets that consist of a high number of features is especially advantageous for high-dimensional data.

Although many of the previous algorithms were further enhanced for certain use cases: Local outlier probabilities (LoOP) [\[KKSZ09\]](#), Local Density Factor (LDF) [\[LLP07\]](#),

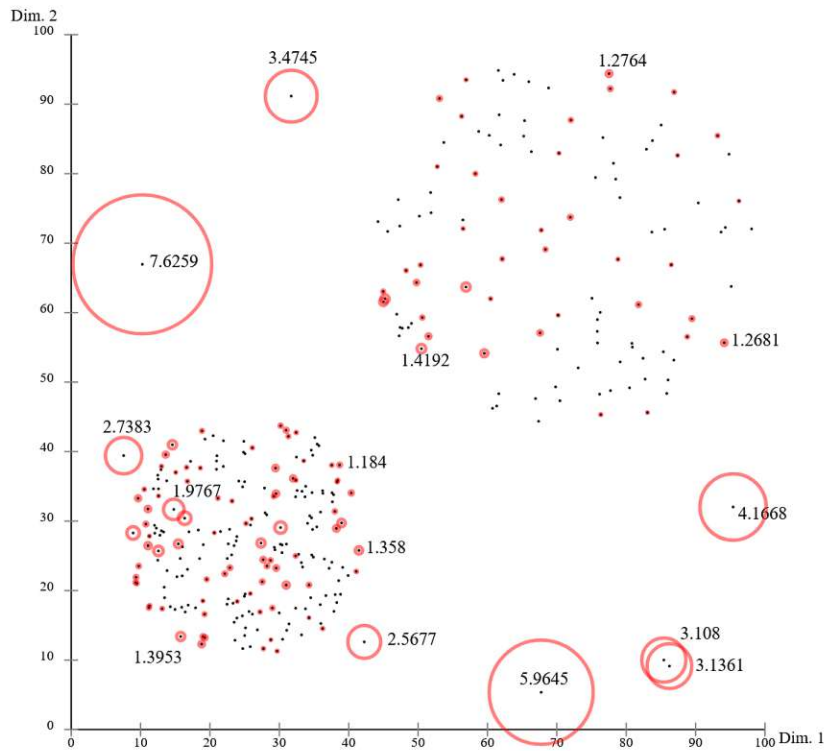


Figure 2.2: Visualization of LOF outlier scores using ELKI. The dataset is artificially generated to highlight LOF strengths. Some interesting LOF scores are printed for $k=5$. [\[WC10\]](#)

Local Distance-based Outlier Factor (LDOF) [\[ZHJ09\]](#), Top-N local outliers (TOLF) [\[YCR17\]](#), Connectivity Outlier Factor (COF) [\[TCFC02\]](#), Cluster-Based Local Outlier Factor (CBLOF) [\[HXD03\]](#), Local Density Cluster-Based Outlier Factor (LDCOF) [\[AG12\]](#), Local Correlation Integral (LOCI) [\[PKG03\]](#), Influenced Outlierness (INFLO) [\[JTHW06\]](#), GRIDLOF [\[CF03\]](#), Outlier Detection using Indegree Number (ODIN) [\[HK04\]](#), Dynamic-Window Outlier Factor (DWOFF) [\[Mom13\]](#), Distributes LOF computing (DLC) [\[BWX15\]](#) and more mentioned in [\[WBH19\]](#) they particularly have a weakness in big data and stream data applications since they are a lazy learner and computational costs are usually quite expensive when new objects are evaluated. In contrast to them Iglesias Vázquez et al. [\[IV18\]](#) propose the concept of Sparse Data Observers (SDO) which utilizes eager learning to minimize computational costs after initially training the model. The main concept is to randomly select a subset of points that are located in medium-to-high density zones as observers, which are later used as anchor points to evaluate instances based on proximity measures [\[MIVZ22\]](#). The algorithm is particularly well suited for large datasets and stream data environments due to its linear $O(n)$ time complexity and roots in principles of sampling theory [\[IV18\]](#). Another unsupervised approach features

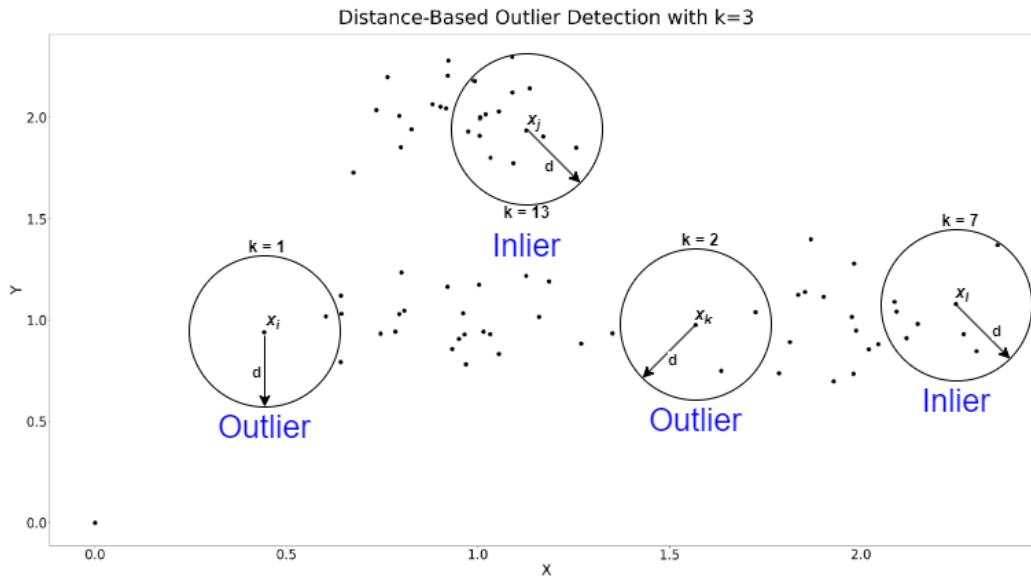


Figure 2.3: Principle of distance-based outlier detection. A datapoint x is classified as an outlier if less than k points are within distance d from x . Drawn based on [KN22, Slide 4].

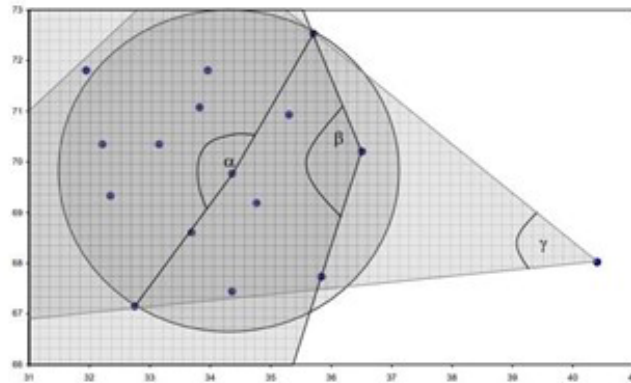


Figure 2.4: The intuition of angle-based outlier detection. Samples that are further away tend to construct smaller angles. [KSZ08, Figure 1].

the usage of histograms that have either static or dynamic bin widths [GD12]. The Histogram-based Outlier Score (HBOS) is calculated by creating a univariate histogram for each feature in the dataset and therefore assumes feature independence [PB19]. Higher bins represent value ranges of denser areas and are weighted higher than bins covering a larger interval [GD12]. The primary intuition behind HBOS can be observed in Figure 2.5. Global outliers usually populate sparse areas so even when the histogram consists of a

small number of bins these instances belong to a bin with low height that is separated from most of the data points. As can be observed in the figure, those low bin heights tend to be mostly on the edge of the histogram.

HBOS is significantly faster than other density-based outlier detection methods offering a linear time complexity of $O(n)$ when using fixed bins and $O(n \cdot \log(n))$ for dynamically sized bins and offering comparable reliability when it comes to detecting global outliers. However, local outliers are often misclassified since histograms are not able to model them properly.

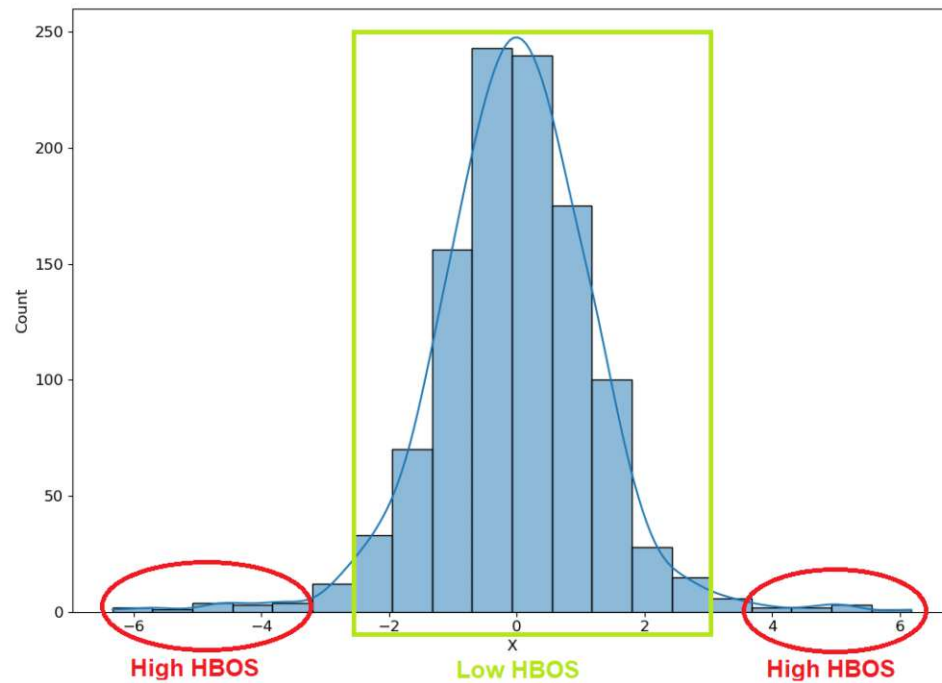


Figure 2.5: The intuition of histogram-based outlier scores. High bin heights in the histogram indicate low HBOS scores and inliers. Low bin heights are interpreted as outliers.

With the increasing popularity of machine learning, classification and regression algorithms were applied to anomaly detection problems. Artificial Neural Networks (ANNs) [MS03] [WBH⁺02] [MJS02] were considered powerful tools to learn complicated non-linear properties from underlying data. However, these approaches were not widespread in the outlier detection community since computational costs became too expensive for large datasets and small datasets were prone to overfitting [CSAT17]. Support Vector Machines (SVMs) on the other hand [SSB18] [WYT07] [HLV03] are often applied on datasets with a relatively small number of variables [BL10]. Although the majority of these learning techniques rely on labeled data and are therefore supervised, the One-Class Support Vec-

tor Machine [Figure 2.6](#) (OC-SVM or 1SVM) is a variant of the regular SVM that utilizes unsupervised learning [\[AGA13\]](#) [\[ERKL16\]](#). In contrast to learning a decision boundary to segregate between different labels, OC-SVMs achieve the maximum separation between points and the origin [\[SWS+99\]](#).

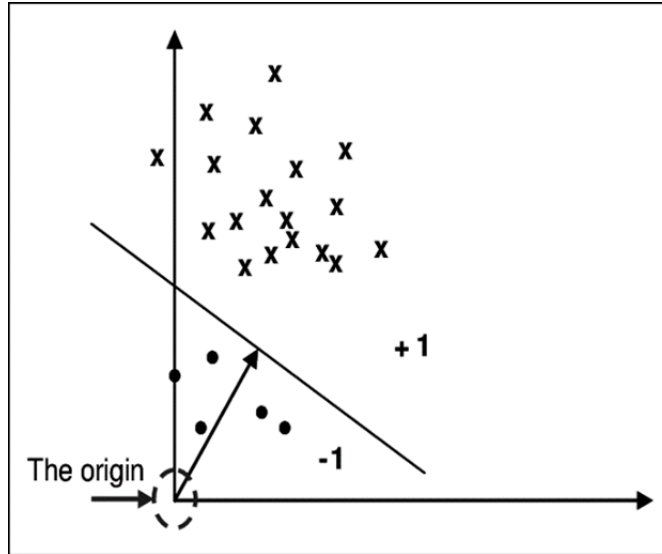


Figure 2.6: Outlier detection intuition in one-class SVMs. The data is separated from the origin with a maximum margin by applying a function f that returns +1 in "small" regions capturing most of the data points and -1 elsewhere. [\[AbdO06\]](#), Figure 1]

A completely different approach was taken by Liu et al. [\[Liu08\]](#). Since anomalies are scarce and mostly deviate from the majority of data by definition, they are easier to separate and more susceptible to isolation. Isolation forests [Figure 2.7](#) are based on building random binary decision trees to divide ("isolate") the current sample from the rest [\[Mah20\]](#) [\[Mis20a\]](#). An individual forest uses an ensemble of different isolation trees to increase stability and performance [\[Die00\]](#) [\[Sch90\]](#). Each isolation tree is completed when a leaf node exclusively contains the corresponding sample. If the datapoint tends to be separated earlier ("short path") on average, the probability of being an outlier is higher than samples with a longer path [\[GZSL19\]](#).

2.2 Validation in Outlier Detection

Machine learning algorithms are powerful methods to generate knowledge from data and can be broadly classified into supervised and unsupervised learning. In supervised learning, models are trained on labeled data to make predictions on unknown and unlabeled data [\[Dri21\]](#) [\[SB98\]](#). These labels can range from discrete binary yes/no labels to continuous variables such as the price for a certain good and must be known beforehand.

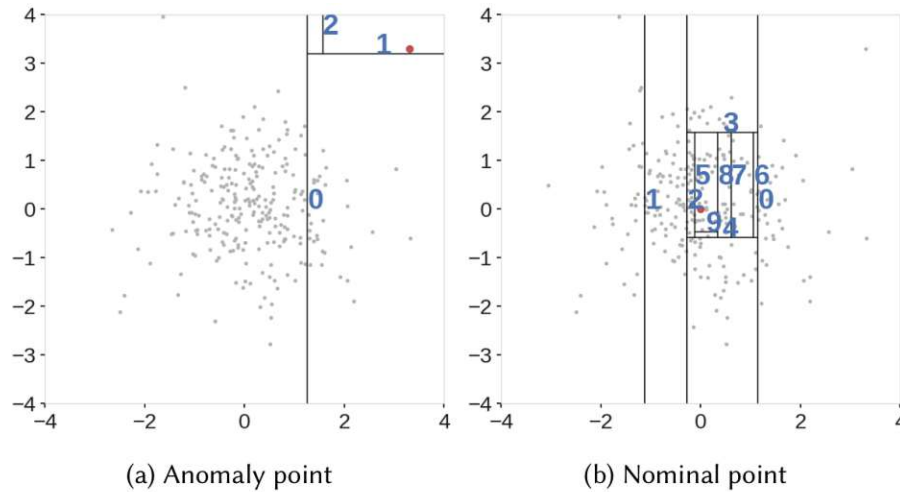


Figure 2.7: Isolation trees for different data samples. Inliers tend to be harder separable than outliers. [HCB18, Figure 5]

Different from supervised learning, unsupervised learning does not need this predefined labeling for performing that task [Bar89]. It utilizes certain properties of the data to identify and extract patterns automatically without the need for any external guidance [Dri21]. The literature often refers to three main tasks where unsupervised learning can be effectively applied on [IBM23]. These are clustering, association and dimensionality reduction [Smo22]. Clustering is used to find specific hidden patterns in the data by properties of similarity and dissimilarity and is the most common approach. Clustering algorithms operate on unclassified data points in order to categorize and group them into subsets referred to as clusters [IBM23]. The second procedure is the learning of association rules [PS91]. Association rule learning uses data mining techniques to uncover different relations between instances of a population that are frequently used together [KC14]. Lastly, unsupervised learning may also be used for dimensionality reduction. Although taking more data into account generally yields more accurate results, especially high dimensional datasets might lead to overfitting models. Dimensionality reduction promises to mitigate these concerns since it transforms data to a lower dimensional subspace while trying to retain the essence of the data as much as possible [Mur] [Bro20] [VdMPPVDH09].

Regardless of the context and use case method mentioned, model validation remains a crucial step in the process of finding the optimal algorithm. Machine learning validation is needed to assess the quality of the learned model and its interpretability and can be broadly categorized into internal and external validation [HPWG20].

Internal validation methods do not rely on any external information to evaluate the clustering structure [PN19] since they focus on two properties where labeling is not needed. First, there is cohesion, which is the proximity of different objects within the same cluster. Higher cluster cohesion indicates better compactness and mostly a more

accurate clustering. Other than cohesion, the separation between objects describes the level of detachment between different clusters and provides information about how well they are partitioned [LLX⁺10]. Objects having a high separation from each other should be in different clusters since high separation leads to large cluster diameters, which opposes the principle of cohesion. An optimal clustering solution usually consists of different tight clusters that are far away from each other.

External validation on the other hand uses a solution known in advance, which is empirical evidence and known as "Ground Truth". A solution is considered to be better the higher the statistical similarity against these true labels.

2.2.1 External Validation

This work deals with commonly used external evaluation algorithms such as:

- Area Under the ROC Curve
- Precision@n
- Average Precision
- Adjusted Average Precision
- Maximum F1 Score
- Adjusted Maximum F1 Score

following a similar procedure like Campos et al. [CZS⁺16] [IV18]. All experiments are conducted exclusively by indices that are adjusted for chance as well as AUC and Adjusted Mutual Information. More details about the exact experimental setup and used external evaluation metrics are provided in Section 3.6.

Before starting to look into these methods in more detail, it is crucial to define the basic terminologies of binary classification. Binary classification tasks result in two possible prediction outcomes on two different axes. The first axis is the classification outcome from the model to be tested, different from the second which consists of the actual result. Both outcomes and axes shape the so-called confusion matrix [Figure 2.8] that differentiates between four different cases for binary classification.

When referencing the confusion matrix these four different outcomes become important in regard to its definition. First, there is the true positive (TP) rate that counts the number of real positives that were identified by the model as such. Differently from TP, a false positive (FP) is an outcome where the prediction is positive but the sample is negative - therefore, a misclassification. Then there are true negatives (TN) which are correctly classified negatives and lastly, false negatives (FN) which are incorrectly classified positives. Depending on the use case and the desired outcome the model tries to

solve, different outcomes in the confusion matrix become more or less important for the evaluation. In the field of outlier detection TP and FP refer to the correctly classified outliers and inliers where FP and FN are wrongly classified outliers and inliers.

		Predicted Condition	
		Outlier	Normal
Actual Condition	Outlier	True positive (TP)	False negative (FN)
	Normal	False positive (FP)	True negative (TN)

Figure 2.8: Confusion Matrix for Outlier Detection Problems. [WT20, Table 2]

Receiver Operating Characteristics (ROC) is a popular method to evaluate the discrimination ability of model predictions independently to their exact statistical characteristics [HM82]. In information retrieval and applied machine learning for binary classification, these curves are popular to measure the performance of different models. In ROC Curves [Figure 2.9], the rate of true positives FPR , sensitivity or recall

$$TPR = \frac{TP}{TP + FN} \quad (2.1)$$

is plotted against the false positives rate FPR or specificity

$$FPR = \frac{FP}{FP + TN} \quad (2.2)$$

at different classification thresholds [PDR16, Unk22]. Each threshold represents the probability that predictions are separated into each target class and hold its own TPR and FPR . Perfect classification is achieved when the curve goes through point $(0, 1)$ since the predictor provides a true-positive of 100% and a false-positive rate of 0% [Faw06, HCl8]. A diagonal line ($y = x$) on the other hand indicates a random guessing model with no additional predictive power [Ste20]. In other words, the point $(0.5, 0.5)$ in ROC space represents a correct guessing of 50% for both positives and negatives. When looking at a second point $(0.9, 0.9)$ of the diagonal line although it can correctly guess 90% of the positive class, its FPR also rises to 90%, which is still not better than random guessing.

Although the ROC is a two-dimensional representation of a predictor's performance, it is mainly used for visualization as any fixed point on the curve can be used for the

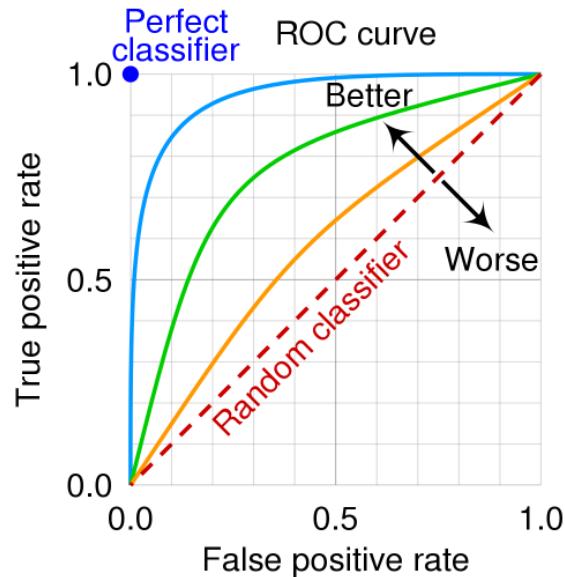


Figure 2.9: Different ROC Curves and their Interpretation. [cmg18](#)

evaluation resulting only in a snapshot of the entire curve [\[Bra96\]](#). A popular choice of a scalar value that estimates the whole curve is the area under it, commonly referred to as Area Under the Curve (AUC) [\[Faw06\]](#). Since the ROC space spans between zero and one, its squared value remains in that interval as well. Furthermore, the ROC AUC represents the probability that a randomly chosen positive example is correctly rated with greater suspicion than a randomly chosen negative example and is equivalent to the Wilcoxon test of ranks [\[PDR16\]](#) [\[Bra96\]](#) [\[HM82\]](#).

Besides ROC curves, using the precision of a classification model is another common evaluation metric.

$$\textit{Precision} = \frac{TP}{TP + FP} \quad (2.3)$$

Differently to recall, which is the number of correct predictions on real positives precision is the proportion of positive classifications.

In the field of information retrieval, precision describes the number of relevant documents divided by the number of retrieved ones [\[Cra09\]](#).

If r relevant documents have been retrieved at rank n , then:

$$\textit{Precision}@n = r/n \quad (2.4)$$

In $Precision@n$ or $P@n$ top n predictions are retrieved and evaluated against the true relevant ones. In outlier evaluation, a higher $P@n$ indicates that the algorithm is better at choosing the n most “outlierish” samples. However, this technique is highly sensitive to the selection of n since correct predictions at later ranks still influence the precision a lot at respective n [CZS⁺16] [PDR16]. To mitigate inconsistent results for different n during the evaluation, it is usually set to the true number of outliers which further means that the user requires some knowledge about the ground truth.

Average Precision (AP) adds the characteristic of recall into the calculation to weight true positives more when they are ranked higher for retrieved results [Zha09].

$$\text{Average Precision} = \frac{\sum_r P@r}{R} \quad (2.5)$$

where r is the rank of each retrieved sample and R is the total number of relevant samples.

Another widespread metric that takes the precision and recall of the confusion matrix into account is the F1 score [CJ20] [Pow08]. It is calculated by taking the harmonic mean of both measures and is defined as follows:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.6)$$

When indicating perfect precision and recall, the maximum possible value of the $F1$ score is 1, where on the other hand it becomes 0 when one of the values is 0. The $F1$ metric is a particular instance of the more generic F_β score that considers an additional factor β , which acts as a weight parameter for recall against precision [Sas07] [SJS06]. The higher the parameter β the more important precision becomes. In the $F1$ measure both precision and recall are equally balanced. However, there are some critical opinions about the $F1$ score in the literature since their equal weights for precision and recall as well as completely ignoring the true negatives may result in overoptimistic inflated results, especially on datasets with large class imbalances [CJ20] [Pow08].

2.2.2 Internal Validation

All discussed measures from before are external validation techniques that require some ground truth. When it comes to anomaly detection validation, there are only a few internal methods discussed in the literature.

Goix et al. [Goi16] introduced an alternative criteria to ROC and Precision-Recall curves on how to compare the performance of unsupervised anomaly detection algorithms by using Excessive-Mass (EM) and Mass-Volume (MV) curves. In their conclusion, they state that the EM and MV criteria reach similar conclusions on finding the best outlier solution in 80% of the cases by (almost) using no labeling at all. However, this method

requires those curves to be estimated by Monte-Carlo simulation, so they do not perform well in high-dimensional space. The authors propose feature bagging to improve the scaling capabilities of EM and MV.

Although there are some unsupervised approaches and metrics available in the field of clustering that can in theory be applied to outlier evaluation as well, these additional validity measurements are only used and useful for evaluating clustering. One of the first metrics for evaluating clustering algorithms is the Dunn Index (DI) which was introduced by J.C. Dunn [Dun74] in 1974. It uses the ratio of the minimum inter-cluster distance to the maximum cluster diameter to evaluate the goodness of the number of clusters chosen on the dataset [Sta17]. In the version proposed by Dunn, the diameter of a cluster A is defined by the maximum distance of two data points x and y :

$$\text{diam } A \triangleq \max_{x,y \in A} d(x,y) \quad (2.7)$$

Under consideration of a dataset having m clusters the Dunn index is expressed by:

$$D = \min_{1 \leq i \leq K} \left(\min_{1 \leq j \leq K, i \neq j} \left(\frac{d(C_i, C_j)}{\max_{1 \leq k \leq K} (\delta(C_k))} \right) \right) \quad (2.8)$$

where $d(C_i, C_j)$ denotes the smallest distance between two points belonging to different clusters and $\delta(C_k)$ is the diameter. In an optimal setting of dense and well-separated clusters, the maximum distance to the neighboring cluster tends to be high whereas on the other hand, the highest distance within one cluster is low, which results in a higher DI. Bad separation is indicated by smaller values. In the literature, maximizing the Dunn Index is often used to obtain the optimal number of clusters for given data [Sta17].

Another prominent metric is the Silhouette coefficient that was originally proposed by Rousseeuw [Rou87] in 1987. The main principle is that each cluster is based on the comparison of its tightness and separation which forms a so-called silhouette. The silhouette $s(o)$ can be computed for any object o belonging to cluster A to which it has been assigned. All objects O must satisfy the properties of metric spaces having some kind of similarity function (Rousseeuw uses Euclidean distances in his work but in theory, any function can be used). First, all average dissimilarities of an object o to all other objects in each cluster C are computed.

$$\text{Average Dissimilarity (AD)} = \frac{1}{n} \sum_{o=1}^n d(o, m(o)) \quad (2.9)$$

where $d(o, m(o))$ is the dissimilarity of an object o to the nearest datapoint to o , denoted by $m(o)$.

Let's assume that object o is assigned to cluster A . When considering a distance metric, the average dissimilarity o to all other objects in A is denoted by $a(o)$.

$a(o) = AD(o, A)$ where A is the assigns cluster of o . Under the assumption that there is at least one cluster besides A , $b(o)$ denotes the smallest average dissimilarity of a different cluster C to object o .

$b(o) = \min_{C \neq A} \{AD(o, C)\}$ where C is in X and X are all clusters except A .

Now the silhouette $s(o)$ can be obtained by the combination of $a(o)$ and $b(o)$ as follows:

$$s(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}} \quad (2.10)$$

Taking that formula into account, it is clear that the silhouette ranges between -1 and 1 . Rousseeuw differentiates between well-clustered cases where $s(o)$ is close to 1 , misclassifications that are close to -1 and intermediate cases around 0 . In general terms, the silhouette tends to move away from 0 when the assignment to a specific cluster becomes clearer where positive numbers indicate that the current clustering is correct where negative numbers point to erroneous classification. Besides being one of the first validation techniques that do not require external labeling and therefore only depend on the actual partition of the data, Rousseeuw presents silhouette- moreover, as a visually appealing graphical aid that improves the interpretability of cluster analysis results. Silhouette plots are line plots where the silhouette value for each observation is presented in descending order and grouped by each cluster. The structure's (=one cluster) height coincides with the number of objects where large and sharp structures indicate that the clustering algorithm found distinct clustering structures. [Figure 2.10](#) shows these structural differences. In the scatterplot, the first (blue) cluster is the best separated, which means that the distance to points belonging to other clusters is comparably high. Hence the silhouettes are closer to 1 and in the plot more delimiting. Another finding is the relative fluent course of cluster 0 (black) cluster that even turns negative for some observations. As already stated, the higher number of objects results in a higher structure in the diagram. However, since some of the instances that are assigned to cluster 2 are negative, they spatially better fit to cluster 0 .

Since silhouette only depends on the spatial properties of the data, it is not sensitive to the underlying clustering algorithm and its parameters. That means for example that a higher k for K-Means does not automatically lead to better silhouette scores but can be used to effectively detect poor choices for k since many very small scores may hint at an inadequate number of clusters.

2.2.3 Adjustment for Chance

While all the mentioned evaluation metrics deliver interpretable results for one dataset, they are difficult to compare between different experiments. For instance, precision is

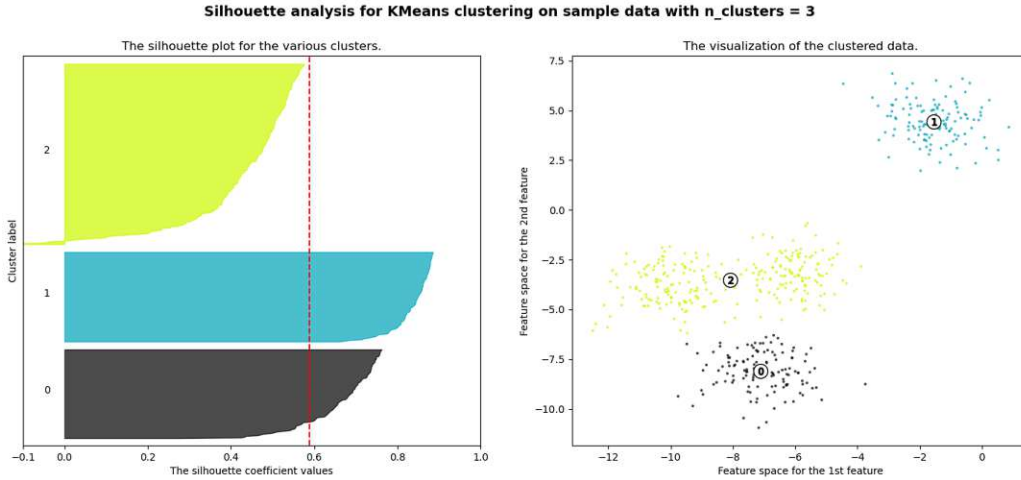


Figure 2.10: Silhouette Plot with corresponding data and clustering [Lea23].

dependent on the ratio of outliers to the number of instances since a smaller number of true positives always leads to a lower score, which is not informative as such [CZS⁺16]. However, due to the fact that the proportion of outliers cannot be generally defined for each use case, these inaccuracies might lead to misleading conclusions when looking at direct relationships. To increase the comparability nevertheless, any index can be adjusted for chance. Originally discussed by Hubert and Arabie [HAS5], adjustment for chance can also be applied on cluster validation to enable score comparison between various experimental settings and is defined as follows:

$$\text{Adjusted Index} = \frac{\text{Index} - \text{Expected Index}}{\text{Maximum Index} - \text{Expected Index}} \quad (2.11)$$

In the literature, when applied to the Rand index, the adjusted index is referred to as Adjusted Rand Index or ARI for short [VEB09] [RBtvV14] [LB21]. All experiments in this work include scores of adjusted indices for Precision@n, Average Precision and Max F1 Score similar to definitions established in Campos et al. [CZS⁺16].

$$\text{Adjusted } P@n = \frac{P@n - \frac{|O|}{N}}{1 - \frac{|O|}{N}} \quad (2.12)$$

$$\text{Adjusted } AP = \frac{AP - \frac{|O|}{N}}{1 - \frac{|O|}{N}} \quad (2.13)$$

$$\text{Adjusted Max F1} = \frac{\text{MaxF1} - \frac{|O|}{N}}{1 - \frac{|O|}{N}} \quad (2.14)$$

The last external evaluation metric that is considered for the experiments is Mutual Information (MI). Mutual Information (MI) is a symmetrical, non-negative measure to quantify the mutual dependence between two random variables [CT06] [VEB09] [RBtvV14] [SG02]. In other words, MI describes the amount of information both variables are sharing. The more of this information is known in one variable the more it reduces the uncertainty in the other one [RLF+06]. Independent variables on the other hand share no common information by definition and are therefore equal to zero [CT06]. The mutual information between two random variables X and Y is defined by:

$$I(Y, X) = \sum_{x \in X} \sum_{y \in Y} p(y, x) \log \frac{p(y, x)}{p(x)p(y)} \quad \text{[VEB09]} \quad (2.15)$$

Since the comparison of all metrics used needs to be ensured during all experiments the adjustment for chance is applied on the Mutual Information Index as well. Vinh et al. define the resulting Adjusted Mutual Information (AMI) as:

$$\text{AMI}(U, V) = \frac{I(U, V) - E\{I(M)|a, b\}}{\sqrt{H(U)H(V)} - E\{I(M)|a, b\}} \quad (2.16)$$

Where $\sqrt{H(U)H(V)}$ is a valid upper bound and $E\{I(M)\}$ is the expected value of the Mutual Information $I(M)$. A more detailed mathematical derivation is provided in [VEB09].

2.3 Internal Relative Evaluation of Outlier Solutions: IREOS

Internal Relative Evaluation of Outlier Solutions (IREOS) is an unsupervised evaluation technique and the foundation of this work. Preliminarily proposed at the *27th International Conference on Scientific and Statistical Database Management* in 2015 by Marques et al. [MCZS15] and further extended in 2020 by an additional publication [MCSZ20], IREOS is the first internal validation index for unsupervised outlier detection [CZS+16] [MCZS15].

Its main intuition is to measure the quality of individual outlier solutions by calculating the added value against the goodness of a random solution. Furthermore, evaluated

scorings on a set of different outlier solutions are comparable in relative terms to each other to elaborate the best candidates on the one hand as well as to establish a ranking [MCSZ20].

In more formal terms, an outlier detection solution consists of a subset $S \subset X$ where X is the input dataset having N samples. S includes all objects that are labelled as outliers resulting in $|S| = n$ where $n \ll N$ in most cases.

$X = \{x_1, \dots, x_N\}$: X is an unlabelled dataset containing N objects.

$S \subset X$, $|S| = n$: S is a top- n outlier detection solution.

Similar to the principle of various other outlier detection algorithms, IREOS evaluates an object's separability [Figure 2.12] to determine the amount of divergence, stating that individual objects that are farther off tend to be better separable than instances in tight clusters. The basic intuition is letting a trained classifier assess the separability of each data instance. In the original work, the use of maximum soft margin classifiers as estimators is encouraged since they possess three advantageous properties:

1. Margins of the decision boundary between instances belonging to different classes are maximized
2. Distances of samples to the decision boundary are quantified into a measure of separability
3. More control in the handling of contextual outliers

Since global and local outliers are clearly separated from the rest of the data, linear separation is mostly applicable. However, complex structures of local outliers as well as inliers are often not linearly separable and therefore don't have a linear decision boundary. In order to mitigate this problem and be more versatile, the paper suggests the use of nonlinear predictors named Nonlinear Support Vector Machines or Kernel Logistic Regression. Both methods take advantage of the "Kernel Trick" [Figure 2.11] a technique that transforms nonlinear data into a higher dimensional space where linear separation is possible [Sch00]. The original authors use Kernel Logistic Regression (KLR) for all their experiments and conclusions as well as for the implementation of IREOS in Java since it brings additional properties to the table that are beneficial for getting a deeper insight into (interim) results:

- Each logistic regression output $p(x_j)$ of a sample x_j can be mathematically explained as probability being an outlier
- Kernel Logistic Regression is known to be robust even in the presence of imbalanced classes and small amounts of training data [MCZS15]

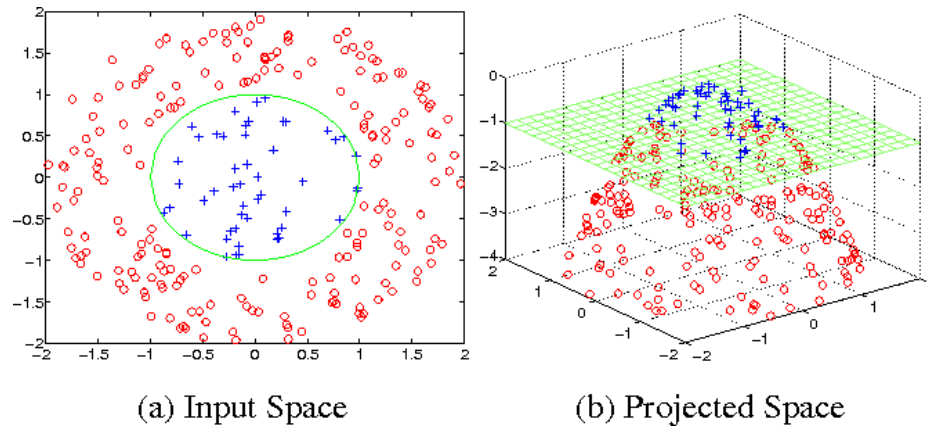


Figure 2.11: Visualization of the Kernel Trick. A kernel function projects the input space into a higher dimensional space where both classes are linearly separable [GCP05, Figure 1].

To narrow down the infinite number of learnable non-linear functions, several kernel functions are introduced. Kernel functions are measures of similarity over all pairs of data points computed by inner products. One of the most popular kernel functions that are also used by IREOS is the Gaussian radial basis kernel (RBF) which is defined by:

$$K(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}} \quad (2.17)$$

where the value of the kernel $K(x, x')$ of two samples x and x' is defined by the squared euclidean distance $\|x - x'\|^2$ and a free parameter σ .

In the domain of machine learning [SSB⁺97] [AL14] [Mis20b] the RBF kernel is often associated with an equivalent definition that involves a parameter $\gamma = \frac{1}{2\sigma^2}$:

$$K(x, x') = e^{-\gamma\|x_i-x_j\|^2} \quad (2.18)$$

As can be observed in the first notation of the formula, the substituted γ parameter is the inverse of the standard deviation of the Gaussian function. An increase of the γ parameter (decrease of σ) defines a Gaussian function with a smaller standard deviation, resulting in the kernel function being more sensitive to distances. Higher values for γ only consider points that are very close to each other as similar because the range of influence of individual samples is decreased, resulting in a more curved decision boundary since it is only dependent on its closest points [Mis20b] [VTS04]. A small value for γ on the other hand leads to a more linear and smoother decision boundary since objects are not discriminated as strictly.

Different from classification tasks, IREOS does not train the predictor on unseen data but moreover uses it to quantify the degree of discrimination leading to the question of how to obtain the optimal value for γ . In classic machine learning tasks, a grid search measures the algorithm's performance on a set of predefined parameters usually selecting the best performing in the end. However, this is not applicable in unsupervised settings since nothing exists to be evaluated against. To overcome the disadvantage of choosing a random value for γ , the original paper suggests calculating the AUC from 0 to a specified γ_{max} for which all instances are labeled as outliers.

$$\int_{\gamma=0}^{\gamma_{max}} p(x_j, \gamma). \text{[MCZS15]} \quad (2.19)$$

Since IREOS evaluates separabilities for each instance of a dataset rather than a single point, the average curve considering all data objects of a given solution S is computed.

$$\bar{p}(\gamma) = \frac{1}{n} \sum_{x_j \in S} p(x_j, \gamma) \quad (2.20)$$

To normalize this value into the interval $[0, 1]$ the result must be divided by γ_{max} . Since the amount of discrimination rises with gamma the AUC is a strictly monotonic function reaching 1 at some point. The normalized index of the separability of a solution S in relation to a collection of data objects is defined by:

$$I(S) = \frac{1}{\gamma_{max}} \int_{\gamma=0}^{\gamma_{max}} \bar{p}(\gamma) \quad (2.21)$$

Since separability curves can only be computed for a finite set of gammas the exact AUC value coming from a continuous range can only be approximated. To avoid fixed calculations of arbitrary numbers for γ to approximate the separability curve, the paper [MCSZ20] suggests adaptive quadrature (or adaptive numerical integration). Adaptive quadrature splits the separability curve into n subintervals recursively until some approximation criterion is met. In the original paper, each interval is subdivided into two equal parts with the motive to prune those recursive trees of accurate sections early. For the error estimate that approximates the divergence of the exact curve I_{exact} and the summation of all subintervals, the original paper proposes the application of Simpson's rule.

$$\mathcal{E}_{2n_\gamma}(I) := \left| I_{exact} - \left(I\left(0, \frac{\gamma_{max}}{2}\right) + I\left(\frac{\gamma_{max}}{2}, \gamma_{max}\right) \right) \right| \quad (2.22)$$

$$\mathcal{E}_{2n_\gamma}(I) \approx \frac{1}{15} \left| \left(I\left(0, \frac{\gamma_{max}}{2}\right) + I\left(\frac{\gamma_{max}}{2}, \gamma_{max}\right) \right) - I(0, \gamma_{max}) \right| \quad (2.23)$$

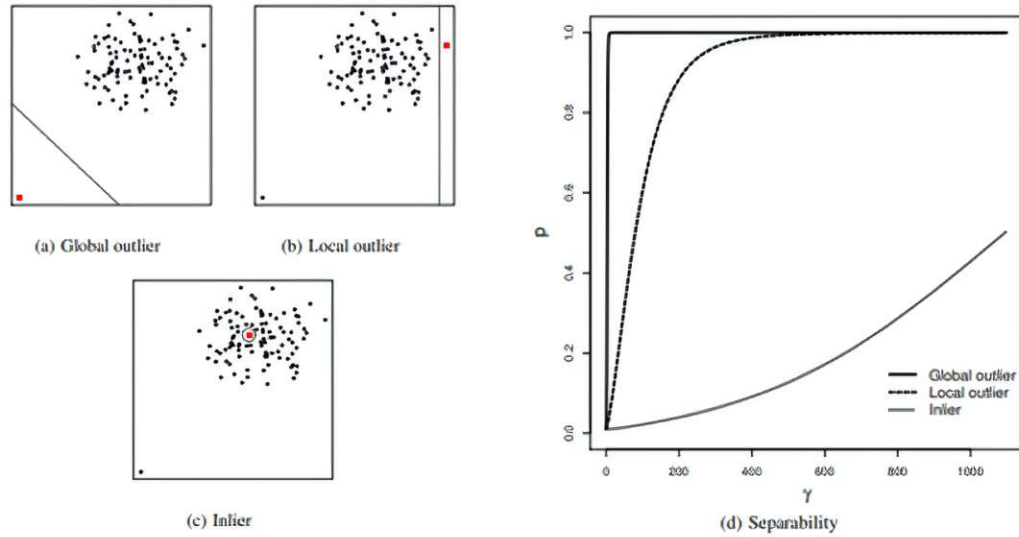


Figure 2.12: Separability curves for different points labeled as outliers. [MCZS15, Figure 1]

As the definition 2.23 shows, the amount of error is approximated by the fraction of the absolute difference between the AUC of the entire interval and after the split into two subintervals (bisection). Simpson's rule uses the fraction of $\frac{1}{15}$ in contrast to the midpoint rule for numerical integration that is $\frac{1}{3}$, resulting in a much faster decay of error after multiple splits.

Although the previously introduced baseline algorithm serves as the main foundation of IREOS, it is still missing some crucial intuitions that are important for various outlier detection applications. One of them covers a user-defined parameter that adds further control on how to interpret unexpected tiny gatherings of points – namely clumps. As already mentioned in Section 2.1 at least basic domain knowledge about the data must be available to correctly decide if contextual outliers are more likely to be noise or equally valid observations. Despite additional parameters being rather uncommon for unsupervised evaluation, the authors included the maximum clump size parameter m_{cl} , where the soft margin classifier allows a certain amount of misclassification for building an optimal decision boundary. This parameter is directly intertwined with the penalty term P_t for both SVMs and KLR and controls separation. Usually, that error term is defined by:

$$P_t = C \sum_{j=1}^N \xi(x_j) \quad (2.24)$$

where $\xi(x_j)$ is an error component associated with x_j and C is a constant. To implement the flexibility of allowing some misclassification for an increased margin m_{cl} is part of this term:

$$C(x_j) = C \cdot \beta \quad (2.25)$$

$$\beta = \frac{1}{m_{cl}} \quad (2.26)$$

Since higher values for C increase the values from the penalty term and tell the predictor to avoid misclassification when building a decision boundary, m_{cl} acts as the inverse of beta to indirectly adjust C . The higher m_{cl} , the higher the fraction of the penalty term, resulting in smaller costs for possible misclassifications when optimizing the hyperplane. Furthermore, the application of soft margins solves a second, related intuition since it reduces the influence of nearby objects of the same clump that otherwise severely influences the decision boundary and margin as of hard margin scenarios.

The computational complexity of IREOS is dependent on the underlying classifier which is executed for each parameter n_γ . Considering the complexity $O(f(N, d))$ as the complexity of the classifier where N is the overall number of samples and d the dimensionality, the resulting complexity becomes $O(n \cdot n_\gamma \cdot f(N, d))$ [MCZS15] [Mah20]. Since KLR is computed in $O(d \cdot N^3)$, the total complexity is $O(n \cdot n_\gamma \cdot d \cdot N^3)$. However, since calculating IREOS for each sample is not dependent on each other, the total work can be split into several subtasks that can be executed on distributed machines, which makes the algorithm highly parallelizable.

In 2022 the authors of IREOS published a follow-up paper named Similarity-Based Unsupervised Evaluation of Outlier Detection (SIREOS) [MZCS22] where the original separability measure is replaced by a similarity metric. Similarity metrics tend to be computationally less demanding in comparison to training classifiers for separability measures. Although SIREOS does not show significantly better results when contrasted to Mass-Volume curves or IREOS, it is faster to compute than the others. According to the authors, this speedup reaches significance not only in regard to the number of data points but also dimensionality. Despite both IREOS and SIREOS being closely related, they do not always agree on the same solutions during the experiments, which makes the existence of either of them justified as they are adding variation to the evaluation portfolio.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Methodology and Experiments

Now that the reader is equipped with the recent state of the art, this chapter continues by explaining the structure of the experiments in detail. We start this section with a brief overview of data gathering and datasets used as well as all the preprocessing that is performed. Every classifier that is used in the experiments is introduced separately to the reader and analyzed for its strengths and weaknesses followed by a brief excursion about data sampling. After establishing all the new concepts that are considered for the new algorithm in theory, we provide a complete technical description of the implementation of the new FIREOS module in Section 3.5. We finish this chapter by going more into detail on the execution of the experiments concerning runtime performance as it lays additional focus on the original IREOS implementation and crucial adjustments that keep experiments comparable.

3.1 Datasets

Since this work deals with the evaluation of outlier solutions, a thorough testing suite of diverse data arrangements is needed to notice the benefits and drawbacks of each technique and reach a proper conclusion. To have more control over the layout of the data, all datasets are exclusively synthetic and generated by a tool named Multidimensional Dataset Generator for Clustering (MDCGen) [LVZFZ19].

3.1.1 MDCGen Datasets

In the paper by Iglesias Vázquez et al. MDCGen is described as a tool that enables the generation of multidimensional and synthetic data that can be used for testing, evaluating and benchmarking unsupervised classification algorithms. It features several mechanisms to control the number of clusters, their distribution, inter-cluster distances and overlap as well as the addition of noise and outliers. In this thesis, a variety of 80

datasets generated by MDCGen are used for experiments featuring different properties such as size and arrangement of the data. The input data consisting of n instances and d features can be split into $d - 1$ columns of predictor variables and one target variable which can be positive or negative. A positive class label represents the associated cluster whereas a negative flags an outlier. These 80 datasets can be further divided into four broad subcategories. Each category emphasizes certain structural characteristics and phenomena that occur on a global or local scale.

Those categories are:

- complex
- high-noise
- low-noise
- basic2d

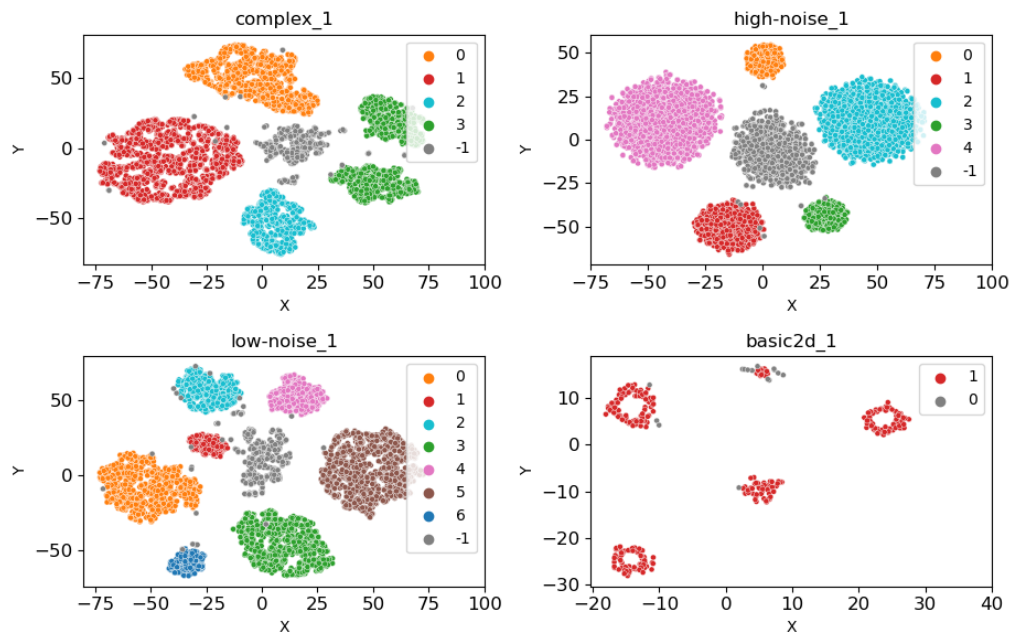


Figure 3.1: 2D representation of different problem instances for each category. Each dataset was dimensionality reduced via T-SNE, which preserves the local structure and clustering of the data. Instances in grey or '-1' correspond to outliers and from 0 to X to different clusters of inliers. (but for the last figure, in which '0' means outliers)

[Figure 3.1](#) and [Figure 3.2](#) show different 2D representations of the structure of one problem instance for each category. The former plot focuses on the preservation of

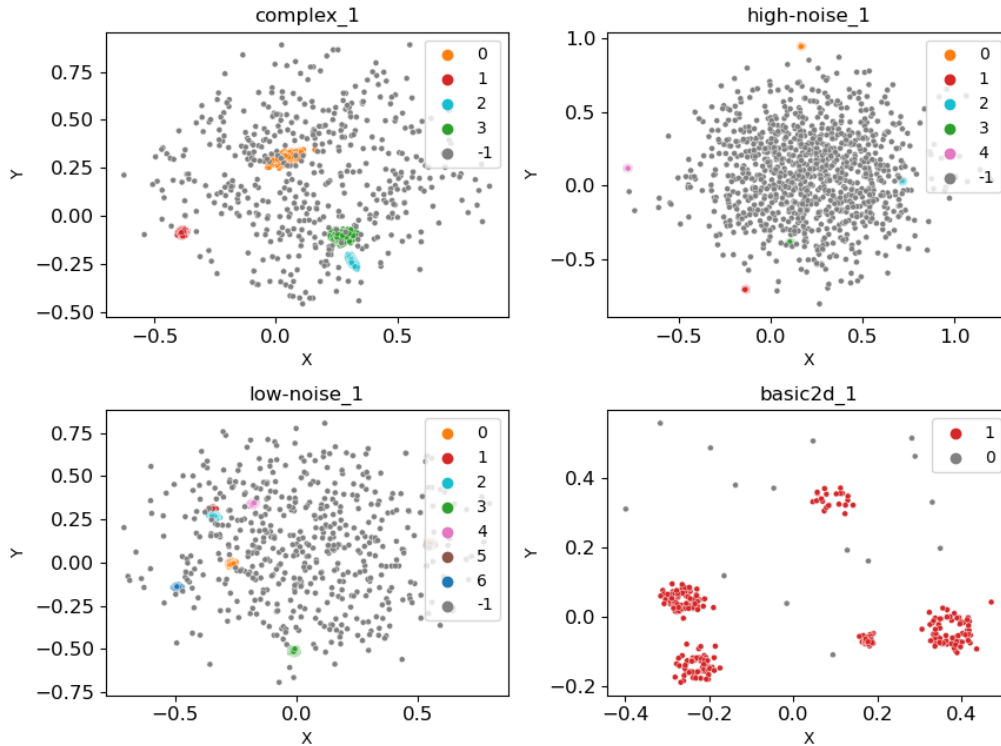


Figure 3.2: 2D representation by using PCA of the same problem instances shown in [Figure 3.1](#). Different from T-SNE, PCA only preserves the global structure rather than local similarities and is highly affected by outliers. Alike [Figure 3.1](#) instances in grey correspond to outliers.

clusters by applying T-SNE on each dataset. T-SNE is a non-linear dimensionality reduction technique that embeds each data point into a lower dimension by preserving local neighborhoods. In other words data points that share a low distance are still close to each other after T-SNE is applied. In contrast to those local structures, the second plot uses Principal Component Analysis (PCA) to highlight the global structure of the same data. Differently to T-SNE, PCA retains the global structure of the data by reducing the dimensionality of variables that have a high correlation. Moreover, PCA is a linear dimensionality reduction method.

As both figures show, datasets belonging to the complex category consist of different cluster-like structures that surround a cluster of outliers. When looking at the high-noise plot, the central cloud of grouped outliers expresses that there are many outliers and they show geometric characteristics that make them very similar to each other when compared to inliers. In low-noise datasets, those outliers are less scattered and basic2d sets feature only two dimensions of input data. In addition, they are far smaller than the

other three categories featuring only ~ 250 samples in comparison to $\sim 5500 - 7000$ for the others.

Although datasets within the same category represent instances trying to cover similar problem areas and align on certain macrocosmic properties, they are still rather diverse and chaotic. Each dataset is associated with a prefix that indicates the category and a number that defines the input instance (i.e.: `complex_1`). All experiments concerning runtime as well as similarity measurements are conducted on these 80 datasets and use this particular naming.

3.2 Data Preprocessing

Data preprocessing is one of the most crucial steps in the data mining process and seriously affects the performance of the outcoming model. In its developer's guide, IBM describes data processing as an important control mechanism to ensure the accuracy, completeness and consistency of the data [Mus19]. Generally, it can be split into four stages:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

As stated in the previous section, all datasets used are considered synthetic data, which makes most of the data preprocessing process obsolete. Procedures such as dealing with duplicates or missing data were already performed by MDCGen.

3.2.1 Training Data

Internal evaluation uses intrinsic patterns and partitions of the input data (training data) to learn a model that is later evaluated against different candidate solutions (outlier scores). Since IREOS and FIREOS are set up like every other machine learning problem, they rely on consistent training data. As already mentioned in the previous Section 3.1 all datasets that are used in the experiments are synthetic and generated via MDCGen. The preparation usually involves cleaning the data first but one major advantage of MDCGen is that all generated datasets do not contain any missing or inconsistent information that needs to be dealt with. That even includes the application of feature scaling, which is not necessary anymore because every feature is already normalized.

Since FIREOS features some predictors that are sensitive towards feature scaling such as SVMs, it works more reliably for scaled training data. However, in theory, tree-based settings such as decision trees or random forests might not lead to much degradation

when they are trained on unscaled data [AMS⁺21]. More detailed information about each predictor is provided in Section 3.3. In summary, each experiment is conducted on normalized data already coming from MDCGen.

3.2.2 Outlier Scores

After training the algorithm on the input data, the second phase consists of the evaluation against a set of candidate solutions - in this work referred to as "outlier scorings". In theory, these candidate solutions can be any outlier detection algorithm that provides a scoring for a given input. A well-performing evaluation metric is distinguished by two properties.

Citing the original paper by Marques et al., IREOS is designed to "*evaluate and compare different candidate outlier detection solutions.*" [MCSZ20]. Since the main purpose of validation metrics is to be able to make a judgment on the performance of one or various models, it is preferred to have a clear separation between solutions of different "goodness".

Secondly, the evaluation algorithm should be able to rate solutions having related "goodness" similar to each other and provide some kind of pattern that is comprehensible. The term "goodness" in this context is a bit critical and hard to grasp since the goodness of a solution can not be postulated as something logical or easily definable especially in the case of internal evaluation and its nature to be exclusively dependent on the underlying data high dimensional structures goes beyond comprehension for us humans. In order to assess the quality of an evaluation metric more accurately and get the big picture on its performance, various scores should be evaluated by it. FIREOS as well as IREOS is evaluated by the same seven different outlier detection algorithms namely:

- Sparse Data Observers (SDO) [LV18]
- Angle-Based Outlier Detection (ABOD) [KSZ08]
- Histogram-Based Outlier Score (HBOS) [GD12]
- Isolation Forest (IForest) [Liu08]
- K-Nearest Neighbor (KNN) [RRS00]
- Local Outlier Factor (LOF) [BKNS00]
- One-Class Support Vector Machine (OCSVM) [SWS⁺99]

More in-depth information about each outlier score can be found in Section 2.1. To evaluate each solution equally, the outlier scoring requires to be inside a certain range as well as similarly interpret inliers and outliers. IREOS expects outlier scorings within the interval $[0, 1]$ where smaller values represent inliers and values towards 1 are more

outlierish. In other words, anomaly scores that IREOS as well as FIREOS expect can be interpreted as outlier probabilities.

Many outlier detection algorithms, however, do not output their solutions as probabilities, which makes some kind of transformation necessary to get them usable as proper inputs for FIREOS. Similar to the original paper, outlier solutions are transformed into probabilities by the normalization framework by Kriegel et al. [KKSZ11] which provides methods to transform different outlier scorings into probabilities. The second intention of the authors is to unify different outlier detection scores of different scales into one uniform scale that sufficiently contrasts an outlier from an inlier. Furthermore, the paper mentions that this normalization framework and the goal of a unification of outlier scores is inspired by Hawkins: "*a sample containing outliers would show up such characteristics as large gaps between 'outlying' and 'inlying' observations and the deviation between outliers and the group of inliers, as measured on some suitably standardized scale*". Depending on the underlying score, the unification is performed in up to two steps.

First, the score is regularized, which means that all values have to fit into the interval $[0, \infty]$. A regularized value $Reg(x) \approx 0$ translates as an inlier and $Reg(x) \gg 0$ as an outlier. The regularization is followed by a normalization method that finally transforms a score into the interval $[0, 1]$. Multiple normalization methods such as min-max scaling, Gaussian Scaling or Gamma Scaling is suggested in Kriegel's work. All experiments conducted in this thesis use min-max normalization for outlier scorings, although FIREOS also features standardization (more details Section 3.5).

Table 3.1 shows inlier and outlier values before treatment as well as the proposed regularization method for each outlier detection algorithm that is implemented in FIREOS. As an important addition only ABOD, KNN, LOF and LDOF are transformed by Kriegel. The other scores are adjusted following his scheme but were not directly mentioned in the paper.

Table 3.1: Outlier scores used for the experiments and their scales and normalization into probabilities

Outlier Score	Inlier Value	Outlier Value	Regularization
SDO	0	$\gg 0$	None
ABOD	80000	0	Logarithmic Inversion
HBOS	0	$\gg 0$	None
IForest	0	1	None
KNN	0	$\gg 0$	None
LOF	1	$\gg 1$	Baseline Regularization
LDOF	$\frac{1}{2}$	$\gg \frac{1}{2}$	Baseline Regularization
OCSVM	> 0	< 0	None

Every value for each score in the table is set after the definition from their original paper. When looking at the scores used for the experiments, however, some scores show

a different scale due to the output of PyOD [ZNL19], which was used to generate each solution. PyOD uses the same interpretation for outliers as the original paper by Liu et al. [Liu08] where an anomaly score very close to 1 points to an outlier. It uses the inverse scores of the score function by SkLearn which defines them as "the lower, the more abnormal. Negative scores represent outliers, positive scores represent inliers." [SkL23]

For the sake of completeness, differently to their description in [KKSZ11] outlier values from ABOD are negative by PyOD. However, since the Angle-Based Outlier Factor, which is then sorted and used by the ABOD algorithm, is defined as "the variance over the angles between the difference vectors to all pairs of points, weighted by the distance of the points" a positive value makes more sense. Therefore, the inverse logarithmic regularization takes care of it by treating negative values as positive. Figure 3.3 shows the distribution of raw ABOD scores over all datasets as well as after the normalization.

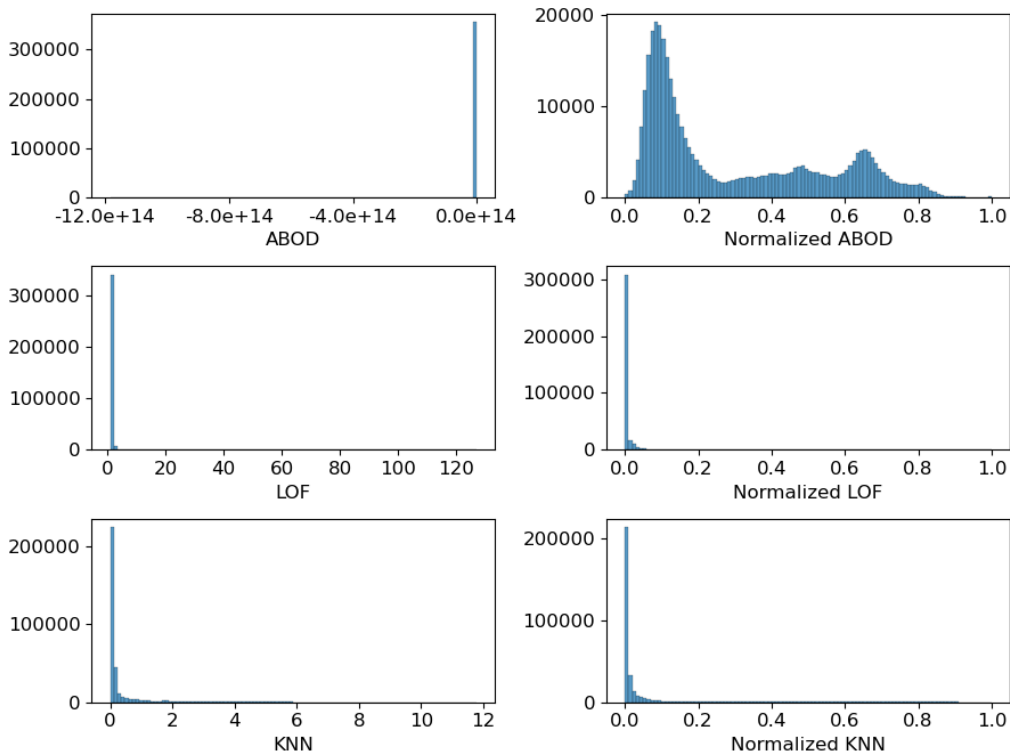


Figure 3.3: Histograms of outlier scores for data points across all 80 datasets before and after the normalization by Kriegel. All normalized scores are transformed into the interval $[0, 1]$ where values close to 0 represent inliers and 1 outliers. Each histogram consists of 100 bins.

Contrary to ones expectation, OCSVM and IForest are both adjusted by normal min-max normalization. Both measures consist of only a few samples having a very large

outlier score which would distort many samples toward being an outlier when inverted. As the majority of scores would be close to 1 after normalizing inverted scores and therefore problematic for a fair evaluation OCSVM and IForest are both adjusted by normal min-max normalization instead of inverting. [Figure 3.10](#) visualizes this process of normalization for all algorithms that are not explicitly mentioned in [\[KKSZ11\]](#).

3.3 Classification Algorithms and Models

The internal classification algorithm is the centerpiece of both IREOS and FIREOS. It is responsible for calculating the separability of each instance versus the entirety of the data, which correlates with the main intuition of the IREOS baseline index, i.e., that outliers tend to be discriminated more easily than other observations [\[MCSZ20\]](#). Furthermore, apart from being the main model for subsequent evaluations, this internal predictor is part of the algorithm with the most computational complexity behind it.

The original paper suggests the use of nonlinear maximum margin classifiers to be able to separate objects that are clear inliers or in other words which are not separable by a linear decision boundary. Both in the paper and in the Java code this predictor is Kernel Logistic Regression (KLR), which is already further explained in [Section 3.3.1](#).

The original work also mentions Nonlinear Support Vector Machines (SVMs) being able to fulfill these requirements. Both procedures are related since they both utilize kernel functions to transform a non-linear problem into a linear separable one. SVMs are explained in more detail in [Section 3.3.2](#).

However, Marques et al. conclude in their final chapter that "*different types of classifiers could lead to indexes with different biases, which nevertheless would still follow the same fundamental intuition behind IREOS*" [\[MCSZ20\]](#), which is the status quo from where FIREOS continues. Different from the main implementation, FIREOS is designed to provide multiple classifiers that still work the same way. The key principle of having a wide selection of possible algorithms is to cover different biases and behaviors, which are later analyzed and compared during the experiments. Although KLR might be replaced by any classification algorithm in theory, predictors returning output probabilities are clearly advantageous due to the behavior of weighted scores by (F)IREOS.

Given these requirements, the following classifiers are featured by FIREOS:

- Logistic Regression
- Support Vector Machines
- Decision Trees
- Random Forest
- Boosted Trees

Tree-based predictors are particularly focused in this work, since they are nonlinear and promise fast and stable results. The following sections aim to provide additional information about each predictor that is used as well as the theoretical advantages and disadvantages of the methods mentioned. Ultimately, important hyperparameter settings as well as their origins and references for each predictor are addressed.

3.3.1 Kernel Logistic Regression

KLR is an extension to classical Logistic Regression, which is a statistical model, that differently from other linear models such as basic Linear Regression, estimates the probability of certain events occurring based on one or more independent variables.

Logistic Regression can be divided into three subtypes:

- Binary Logistic Regression
- Multinomial Logistic Regression
- Ordinal Logistic Regression

Binary Logistic Regression focuses on the categorical response between two possible classes. IREOS by its KLR predictor as well as the ordinary Logistic Regression in FIREOS utilize binary outputs to determine the probability of an arbitrary sample being an inlier or outlier. Besides binary responses, it is also possible to include more than two output classes, which is then referred to as multinomial Logistic Regression. Ordinal Logistic Regression is applied if those categories indicate any kind of order.

Logistic regression belongs to the category of discriminative models, which stand out by setting up decision boundaries through observed data. Different from generative models, that learn the joint probability distribution, only the posterior probability of each output class is taken into account by them [Men96]. Logistic Regression uses the sigmoid function to map any input variable into the interval $[0, 1]$. Values > 0.5 are then labeled as positive class and predictions ≤ 0.5 become negative.

However, classical Logistic regression is not able to separate nonlinear data accurately [EZ19], which is the reason to include a similar technique on what nonlinear SVMs do, that is to use a kernel function. Kernel functions are used so that linear classifiers are able to solve problems that are otherwise linearly inseparable by mapping the feature space into a higher dimensional space. The most commonly used kernel functions are linear, radial basis function and polynomial kernel.

Classical logistic regression that uses a kernel function becomes KLR. KLR is often compared to nonlinear SVMs, since they have a similar classification performance [Has03] and are both powerful discriminative methods [KDSP02]. Different from SVMs, KLR is suited for Bayesian design and features direct probabilistic interpretation, which is especially beneficial as an internal predictor for IREOS, since output separabilities are also

based on probabilities. Furthermore, by design KLR features optimal margin properties during data separation [Has03].

However, KLR has an algorithmic complexity of $O(n^3)$, which is computationally more expensive than SVM which has $O(n^2m)$ where m is the number of support vectors. Since one main goal of FIREOS is to be more lightweight than its predecessor, this $O(n^3)$ remains a very crucial key figure for algorithmic choices. In other words, significant speedup for FIREOS can only be achieved when

$$O(X) < O(n^3) \tag{3.1}$$

for each algorithm X , which is the reason why the new implementation does not feature KLR but classical LR, which has a time complexity of $O(nd)$ where d is the number of dimensions [Ban20].

3.3.2 Support Vector Machines

Another widespread linear discriminative model is the SVM. Originally invented by Vapnik and Chervonenkis in 1963 [V.N64], SVMs are popular predictors for classification and regression. SVMs tend to be quite robust when it comes to new observations since they calculate an optimal decision boundary between two classes.

But how do SVMs decide on an ideal solution when infinite solutions for them exist in theory?

Same as other discriminative classifiers, SVMs try to find a hyperplane that separates both classes as well as possible. Their main principle is to maximize the margin between the two points closest to the decision boundary, which are named support vectors. Figure 3.4 shows the SVM decision boundary of a linearly separable problem. Samples that are on the margin are called "Support Vectors".

Since the resulting learner is an optimization algorithm that tries to maximize the margins of the decision boundary, SVMs are called maximum margin classifiers.

Margins in the context of classification can be subdivided into two subcategories.

First, there is hard margin classification, where the predictor does not allow any misclassification when building the hyperplane, which only results in a unique solution when the data is linearly separable. Furthermore, hard margins are generally very sensitive to outliers and tend to be prone to overfitting.

On the other hand, there is soft margin classification, which allows some separation violations for the decision boundary. Soft margins do not suffer from overfitting like hard margins and further have the great advantage that linear kernels might still work for not fully linearly separable data. The concept of different margins was transferred into the model by the declaration of the cost hyperparameter C , which controls the amount

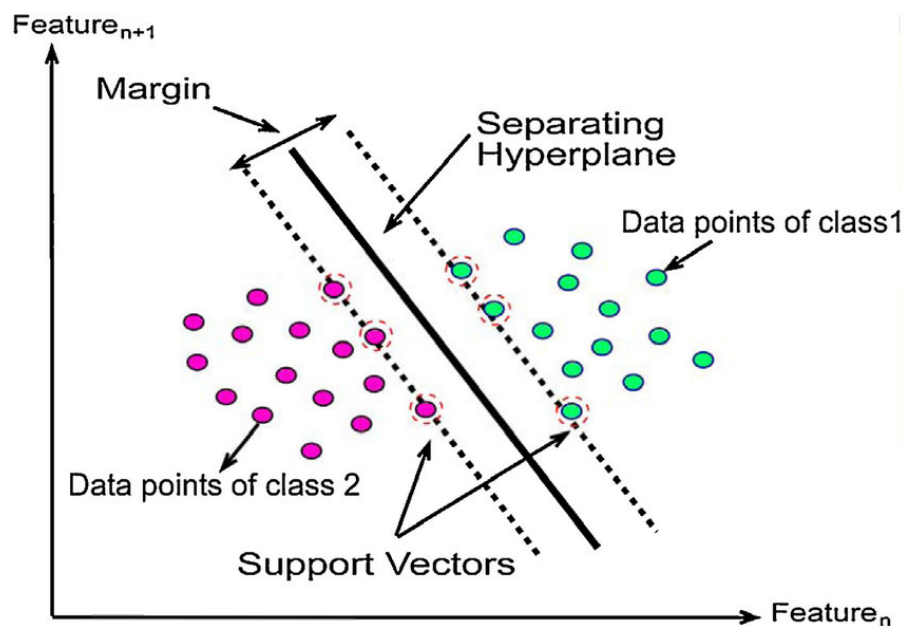


Figure 3.4: The main principle of SVMs. The predictor tries to find an optimal hyperplane by maximizing the margin between the support vectors. [SO17 Figure 2]

of overlap [HRTZ04]. It acts as a penalization factor for erroneous classification, where higher values force the predictor to favor smaller margins but fewer misclassifications over larger margins however, some violations might occur. For the sake of completeness, the parameter λ should be also mentioned as it is often mentioned in the literature [EPP00, Pla00, HRTZ04] as a "regularization parameter" and directly related to C :

$$\lambda = \frac{1}{C} \quad (3.2)$$

Although soft margins enable the solution of linear decision boundaries even when the data is not linearly separable, there are problem instances where nonlinear decision boundaries are clearly desired. Analogous to KLR in the last section, SVMs also utilize kernel functions in order to separate data that is otherwise linear inseparable. One prominent kernel function that is also implemented in FIREOS is the Gaussian Radial Basis Function (RBF).

Kernel functions are continuous, symmetric functions that can be used as measures of similarity between data objects [Sre07]. Kernels usually describe complex properties of high-dimensional data that are hard to interpret or unintuitive. The RBF kernel on two samples $x \in R^k$ and x' is defined as:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \quad (3.3)$$

The hyperparameter σ can be interpreted as the amount of influence each instance has on its surroundings. A higher value for σ results in data points still influencing the classification of new points that are farther away. In other words, the decision boundaries radius from each data point is higher for an increased σ , which leads to an overall less curved line since instances additional to the support vectors are influencing the optimal hyperplane. Furthermore, chances of overfitting become higher for lower values since the model strictly disallows misclassification and starts building up local islands of decision boundaries [SAA20].

The parameter σ is often replaced by a different control parameter γ :

$$\gamma = \frac{1}{2\sigma^2} \quad (3.4)$$

which leads to the notion that is also mentioned in the original IREOS paper:

$$K(x, x') = \exp(-\gamma\|x - x'\|^2) \quad (3.5)$$

In conclusion, γ acts as the inverse of the radius of samples and is indirectly proportional to σ .

Nonlinear SVMs using the radial basis function kernel are also featured in FIREOS by the Scikit-Learn and LibSVM [CL11] libraries. Both implementations, however, have slight differences when it comes to the assignment of parameters.

The SVM predictor that is delegated to the Scikit-Learn library is based on the original paper and implementation where cost parameter C is fixed to 100, class weights are balanced and unbounded iterations are used. Furthermore, a tolerance of 0.0095 was adopted from the `tol` parameter of the Java implementation [Mar20]. Besides the explicit setting of probabilistic output, LibSVM always runs default parameter settings. Both predictors still accept an arbitrary parameter for γ , primarily for the numerical integration that is used. Further details are provided in Section 3.5 where the implementation is explained.

There is still one question left unanswered and that is the calculation of probabilities. Since SVMs do not directly provide class probabilities, their prediction cannot be directly used as separability for instances in FIREOS. However, there are methods to transform non-probabilistic outputs into $[0, 1]$ such as Platt scaling, where the SVM trains an

additional parameter of a sigmoid function, that maps each output into a probability [Pla00].

Since outlier detection evaluation problems only consider the probability for outliers (or inliers respectively), they are equivalent to binary classification. Both Scikit-Learn and LibSVM state in their official documentation that all probabilities of binary class problems are calculated by Platt scaling [Lea23]. For the sake of completeness, it should also be mentioned that multi-class probabilities are calculated by extensions of Platt scaling. Scikit-Learn mentions [WLW04] in its documentation and LibSVM cites [LLW07].

3.3.3 Tree-Based Approaches

As already mentioned in Section 3.3 tree-based models are particularly interesting for FIREOS. But what exactly are the characteristics of tree-based models and how are they beneficial for the new software?

Tree-based algorithms are the collective term for a set of predictors that use a decision tree as input data representation to predict target variables [Mis23h]. Decision trees are with an algorithmic time complexity of $O(nd \log(n))$ [SLN18] less expensive than KLR and show high accuracy and good separation even for categorical data. Furthermore, they are comparatively easily interpreted and visualized and perform well on nonlinear relationships as well, which makes them particularly well suited as a KLR replacement predictor. Decision trees can both be used for classification (Classification Tree) and regression (Regression Tree) problems.

In general decision trees are built by determining the feature to split on, which is also called Attribute Selection Measure (ASM) and the optimal value division. Although there are several goodness metrics on how to decide on the splitting, the more common approaches are Gini Impurity, Information Gain and Log-Loss Scoring which are also implemented by the Scikit-Learn library [Mis23g].

The Gini Index or Impurity represents the probability that a randomly chosen sample gets misclassified. It is defined by:

$$Gini(X) = 1 - \sum_{i=1}^k p_i^2 \quad (3.6)$$

[DT14]

where i is the number of different class variables and $p(i)$ the probability of an class i being correctly classified. After calculating the Gini Impurity for each attribute, the algorithm chooses the split with the lowest value. These two steps are repeated until a termination criterion is reached. Decision tree algorithms that use the Gini Impurity split each attribute into two sub-partitions resulting in a binary decision tree.

Although the construction of such a tree is relatively fast, a major drawback is a strong tendency to overfit especially for fully-grown models. Even slight changes in the input

data may influence the sequence of splits and impact the decision boundaries of the final model severely.

One effective measure to reduce the high variance of a fully trained predictor is pruning the tree which prevents the model from overfitting. Esposito et al. [EMS97] published a profound analysis as well as empirical studies of several top-down and bottom-up pruning methods that aim for a simplification of decision trees.

Nevertheless, decision trees show the same challenges as SVMs as their results are non-probabilistic. They are not intended to act as probabilistic models and lack well-founded and uniform recommendations from publications. There are, however, some important contributions that address this issue [ZE01, KJS17, MD03].

The FIREOS implementation features two different decision tree approaches. One is using the common Scikit-Learn interface and the other is a native implementation in Julia. The former uses the relative frequency of classes in the leaf nodes as a probability. However, keeping the preset parameter settings for the classifier would result in a major pitfall, since Scikit-Learn does not apply any pruning by default, which would always lead to $P = 1$ since leaf nodes are grown until they are pure.

To mitigate this problem, the maximum tree depth parameter is set to 2 and therefore heavily pruned, which furthermore leads to faster execution due to fewer splits. The native predictor calculates probabilities differently since the library is shipped with a method that *"calculates $P(L = label|X)$ for each row in features. It returns a $N_row \times n_labels$ matrix of probabilities, each row summing up to 1."* [Mis23a]

To still cushion the problems of overfitting, FIREOS constructs 1000 independent trees and takes the mean of their probabilities. As various independent trees can be composed into a different classifier named random forest, the naming of this procedure remains debatable. Due to the reason that we want to keep the naming of each classifier in FIREOS as close as possible to those of the internal libraries we decided to retain the naming as is.

Nonetheless, random forests are also included by FIREOS and similar to decision trees two implementations are provided. As before FIREOS features a native implementation and the classifier from Scikit-Learn.

But what is the difference between a random forest and a decision tree?

Random forest is an ensemble learning method that uses a combination of different tree predictors at training time to make predictions by unifying their outputs for instance by majority vote [Bre01]. The intention of using randomized simple trees is to make the predictor more robust by reducing the noise of an individual model. Breiman further states that the generalization error converges against a certain value when the number of independent trees is high enough.

Similar to decision trees, finding the optimal splits and sampling techniques was the subject of much discussion in the literature varying from bagging [Bre96] as an early

approach to random splits [Die00], randomized training sets [Bre99] and random selection of features [Ho98]. Both implementations use the default parameter setting of the native library, which applies random splits on all features, a sample size of 70% of the input instances and 10 trees.

Furthermore, like for normal decision trees random forest outputs need to be transformed into probabilities. Since both libraries provide their implementation for it, these functions are used.

Besides decision trees and random forests, FIREOS features an additional tree-based classifier that is XgBoost [CG16]. Boosting which is similar to Bagging in a random forest is an ensemble learning method of weak predictors but instead of training them in parallel, boosting learns from the mistakes of the previous model.

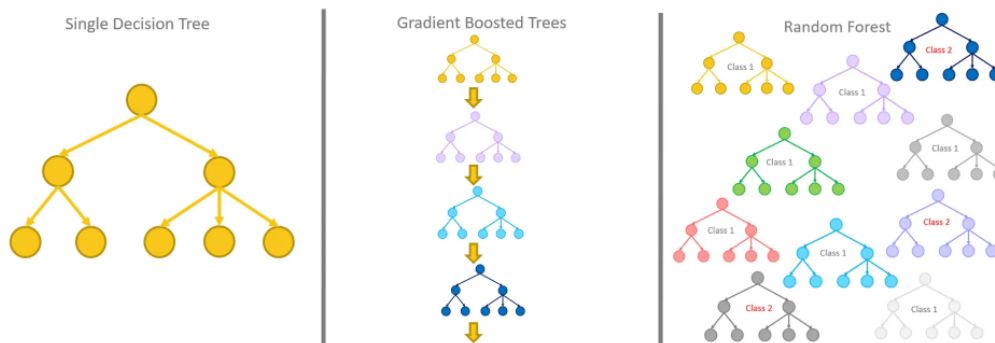


Figure 3.5: Principles of decision trees and ensemble models. Random forests use parallel bagging to improve the predictive performance of decision trees. In gradient-boosted trees, multiple weak learners are sequentially combined into a stronger model. [Sil20, Figure 1]

After training a base learner, another weak predictor is trained on the residuals which are calculated from the loss function (i.e.: log-loss, MAE) of the first tree [Sim22]. These steps are repeated until there is no predictive gain or a maximum number of trees are fit. [Figure 3.5] visualizes the principle of bagging and boosting in comparison to a singular decision tree.

One prominent technique is the principle of Gradient Boosting which uses gradients to minimize the loss function. XgBoost uses the principle of gradient boosting with some slight modifications discussed in [CG16]. In summary, these adjustments make the base Gradient Boosting more flexible by adding an objective function instead of a loss function as well as second-order gradients and penalty terms.

FIREOS uses three different XgBoost models consisting of different boosters:

- Tree Booster

- DART Booster
- Linear Booster

Although two of them use tree-based boosters (Tree and DART) and one a linear model, all three models use logistic regression for binary classification as a learning task, since that is the only way to produce output probabilities.

The setting for the Tree Booster is inspired by the docs [Mis22] and acts as a standalone random forest with boosting. The other two classifiers use hardly any different parameter settings than default, except similar to before the maximum depth for each tree is limited by 2 for each case.

3.3.4 LibLINEAR

The last classifier that is added to the new implementation belongs to "LibLINEAR - A Library for Large Linear Classification" [FCH⁺08]. LibLINEAR provides several lightweight linear predictors that promise fast results even for large datasets. Although the original IREOS paper explicitly recommends nonlinear classifiers, best yet with a soft margin, the intuition for adding a predictor from LibLINEAR is primarily to test the tradeoffs between very fast results and stability. When looking further, LibLINEAR has some similarities with LibSVM which also provides SVM implementations.

The main differences between those two packages lie in the details of each implementation. The SVM solver in LibSVM is intended to work both for linear and kernelized operations, which comes at a cost. Bottou et al. [BL07] estimate in their work about SVM solvers, that the asymptotic number of support vectors grows linearly alongside the number of instances leading to the computational costs of an algorithm capable of solving an SVM problem with an arbitrary kernel between $O(n^2)$ and $O(n^3)$. The latter for models with a high cost-parameter C .

Since LibSVM tends to be a more versatile library with kernelized SVM implementations, it is a decent choice for the predictor in Section 3.3.2. Nevertheless, for linear SVMs, LibLINEAR should be preferred due to its near-linear training time [Joa06]. Unfortunately, LibLINEAR currently does not feature probability estimates for its SVM implementations [Mis23a] which makes them unusable for FIREOS. However, there is also an implementation for L2-regularized logistic regression which features a parameter for probability outputs. This algorithm setting with a dual-based solver and all its default parameters is the predictor that is finally used in the FIREOS implementation.

3.4 Sampling Methods and Subsetting

Besides the reduction of computational expense by using a more lightweight predictor, another dimension of adjustments is the reduction of input data. The core intention with respect to FIREOS is obvious.

Since FIREOS trains at least n predictors of an input space of $n \times d$ samples, a significant reduction would result in a major speedup. This section aims to provide different sampling and subsetting techniques that reduce the number of observations for training the classifier. Although there are feature selection algorithms for dimensionality reduction that also decrease the input space, this chapter is dedicated to horizontal downscaling. Apart from the idea of sliding windows, which is characterized in the next subsection and integrated into FIREOS, other approaches that were not included in the final implementation are covered in theory.

3.4.1 Sliding Windows

The concept of sliding windows is one of the simplest but most effective measures to control the number of instances used for the predictors. While it is easily integrated as an additional tuning parameter for FIREOS and provides a decent amount of control, sliding windows enable the processing of large datasets that would otherwise be infeasible for internal evaluation.

Initially originated from streaming data environments where huge volumes of data need to be processed in a reasonable time, the use of a window keeps a managed subset of instances in memory [KGP⁺11]. All samples within the sliding window are termed as active objects and relevant for the calculation, differently from the rest that is ignored. If a new data point enters a saturated window, the oldest object becomes inactive and is therefore no longer managed. Although the extension of stream data analysis is possible for FIREOS in theory, its main focus, for now, is to operate on normal outlier solution evaluation. Therefore, the concept of a time-based window is changed to a count-based approach where *top-n* samples from the current sample that is calculated are kept.

The main advantage of using a sliding window of size $W \ll n$ is to reduce the complexity of predictors from $n \cdot n \cdot d$ to $n \cdot W \cdot d$. Furthermore, since the separabilities from IREOS and FIREOS are calculated sample after sample, the sliding window changes only by one instance in each iteration. After the separability of sample $x_i \in N$ in a sliding window scenario of $W \ll n$ is calculated, x_i is removed from the active collection of samples, x_{i+W} is set to active and calculated. The last W samples always use the instances $x_{(n-W), (n-W+1), \dots, n}$ to prevent underpopulated windows. Figure 3.6 shows the intuition of sliding windows in FIREOS. An additional optional parameter that can be configured for FIREOS is the size of the sliding window. A lower value leads to faster training time but inaccurate and fluctuating results.

One major advantage of this approach is the absence of any data shuffling or random sampling step, which saves execution time. However, the lack of shuffling might also lead to unrepresentative subsets and an unfair number of occurrences especially when the arrangement of data points is in order.

Particularly the last W data samples are over-represented since they are included in W predictors where the first instance is only featured once and that is when the separability of itself is calculated. Still, it is arguable if random sampling results in a significantly

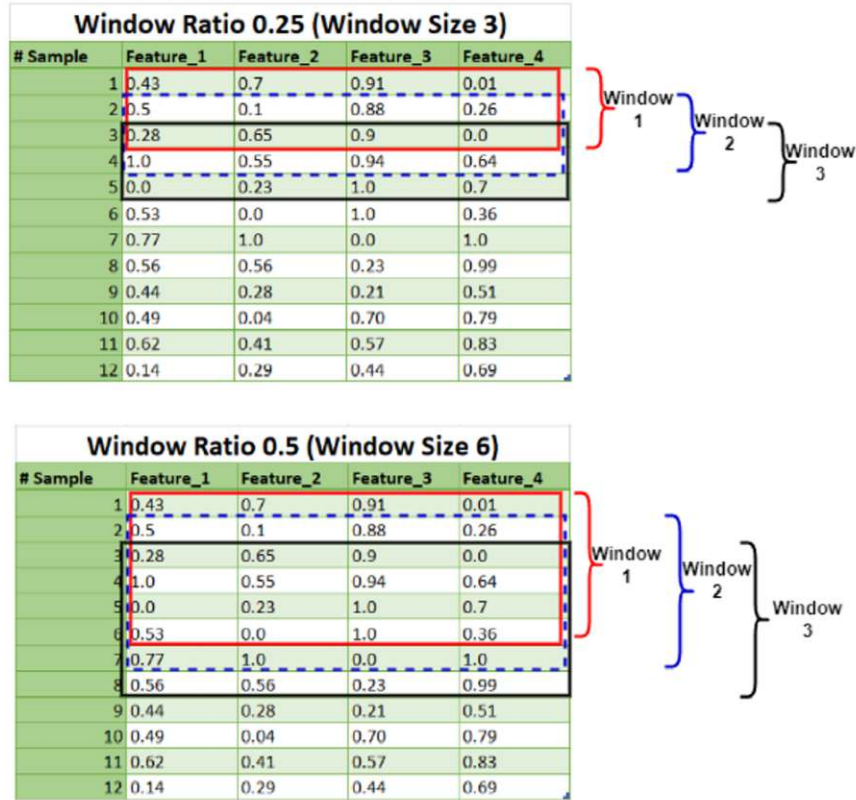


Figure 3.6: Sliding windows in FIREOS. The smaller the ratio the smaller the window frame. A window ratio of 1.0 equals the full dataset.

better solution here, since as long as the arbitrary picking is not stratified, samples might suffer from under-representation nonetheless. Although theoretical considerations of other approaches were pursued and presented in subsequent chapters the application of sliding windows following the example of [Figure 3.6](#) is the only subsampling feature we included in the final version of FIREOS of this thesis.

3.4.2 Coresets

A statistically and in the context of outlier detection more sophisticated method is the usage of coresets instead of sliding windows. Coresets or originally Core-Sets [\[BHP102\]](#) have their origins in the field of clustering, where they are defined as smaller subsets of points from a dataset that still preserve the structural properties and clusters of the data. Bădoiu et al. prove that an $(1 + \epsilon)$ approximation exists for k -centering and median-centering in Euclidean space when applied on the coreset.

According to Ding et al., [\[DYW19\]](#) the main flaw of the original idea is a high complexity

due to the large number of possible coresets. So a different approach is introduced by using a greedy algorithm inspired by the k-means clustering algorithm from Gonzales [Gon]. This greedy approach is aimed to be a "*quality guaranteed algorithm with low complexity*" [DYW19] that further works reliably with outliers in the data.

Besides an alternative approach for k -centering, this greedy algorithm is able to construct coresets for clustering problems with outliers. Since executing algorithms on a small coreset significantly reduces time complexity in comparison to the whole dataset, they are very well suited for large, high-dimensional datasets. According to Bădoiu et al., the size and dimensionality of the underlying data are not decisive for the coreset.

Although coresets were initially perceived as a technique for clustering, their concept might be applicable in different areas as well. Especially their property to preserve the structural occurrences and layout of outliers makes coresets an interesting choice for FIREOS, which relies on the existence of outliers. However, the number of libraries featuring a fast implementation of coresets is quite limited.

A particularly promising implementation named "Minicore: Fast Generic Coresets" [Bak23] was field tested prior in combination with PyIREOS [Tob22], an early prototype and predecessor of FIREOS in Python also written by the author of this work. Minicore features many concepts that were discussed before such as Gonzales algorithm or greedy coreset construction, which resulted in fast subsets of the data. However, a stable integration into the Julia code of the current FIREOS implementation became problematic due to limitations, which ultimately led to the discarding of this idea. Especially technical limitations in regard to multithreading and Pycall which will be further addressed in Section 3.5.4 in more detail are largely to blame for this.

Nevertheless, coresets are still a promising extension to further reduce the computational costs of internal evaluation indices like (F)IREOS and should be considered for large datasets.

3.4.3 Microclustering

Another method that should not be left unmentioned is the concept of micro-clusters. Similar to coresets, it is often associated in the literature as a clustering technique, since it primarily acts nowadays as a structure for stream clustering [MNPT18]. The reason nowadays is that the concept of micro-clusters originates from Cluster Feature (CF), which was introduced in the BIRCH algorithm [ZRL96] but deals with static data [AWS14, MNPT18]. Aggarwal et al. [AHW⁺03] extended this original approach by the addition of a time window that tracks and maintains important information, since revisiting former objects in data streams is not possible.

Different from the classic sliding window approach, statistical information about data locality is kept by micro-clustering. This is achieved by storing it in a specific data structure called pyramidal time frame. Its main intuition is assigning incoming data samples in a stream to the closest existing clusters. If no cluster is suited well enough for

a merge (since the maximum boundary is lower than the distance from the new data point to the corresponding cluster), a new cluster must be created for that sample. However, since the initial number of micro-clusters must not be exceeded, an existing cluster must be removed or two clusters combined into one. In case of removal, the algorithm favors deleting clusters containing outliers with an old timestamp. More details about the exact calculation and storage can be found in [AHW⁺03].

Continuing this procedure over large data streams results in clusters that span over regions that contain exclusively inliers [KGP⁺11]. Since micro-clusters can be constructed in a fast fashion even for large volumes of data, their output can be used as a structurally similar approximation of the input dataset as well as for data streams. This problem simplification was also considered as a possible speedup opportunity for FIREOS but lack of adoption in official implementations leaves this extension to future work.

3.5 FIREOS

Fast Internal Relative Evaluation of Outlier Solutions or FIREOS for short is a software implementation and extension of IREOS. It is the centerpiece of this work as it is used for most of the experiments later on and embeds most of the theoretical concepts introduced in Section 3.3 and Section 3.4. This chapter is dedicated to giving detailed information about the implementation of FIREOS and its features such as different predictors, sampling techniques and other settings and parameters.

The first subsection provides some non-technical general information about the implementation by explaining core decisions on the architecture. Moreover, a high-level view of the software interface of FIREOS is given. Followed by Section 3.5.2 which is the first chapter that explains the actual code of the new implementation by discussing the part of FIREOS that deals with the normalization framework introduced in Section 3.2.2. Section 3.5.3 continues with an in-depth discussion about technical features and implementation details of the main FIREOS functions. The following subsection aims to provide some closing words and thoughts about various milestones and events that occurred during the engineering. Furthermore, it provides explanations about difficulties that lead to the omission of several features mentioned in previous sections as well as different technical challenges that lead to various refactorings and changes throughout the development. Ultimately, the last section provides insights into how benchmarks against the original IREOS implementation are performed.

3.5.1 General Characteristics

The main intuition of FIREOS is to provide a fast library for internal validation of outlier solutions which starts with the decision of a suited programming language. The key requirements for this language are high performance, flexibility and good multi-threading capabilities.

The Julia Programming Language is a relatively new, fast and dynamically typed programming language that "*is designed for parallelism, and provides built-in primitives for parallel computing at every level*". [Mis23b] Furthermore, it provides various packages for machine learning models and data visualization tools. Promising the performance near low-level languages, the decision fell on Julia as the programming language for FIREOS [Joh17].

Figure 3.7 shows benchmarks of different programming languages performing different operations. As can be observed, Julia is not only fast but also quite consistent with various problems.

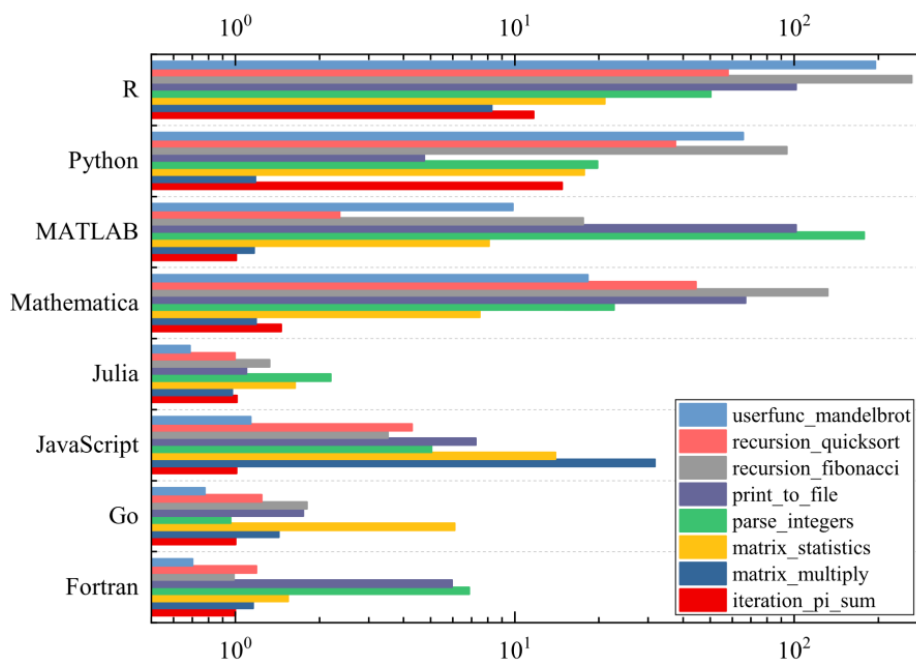


Figure 3.7: Performance comparison of different programming languages for different computations. [GMP+20, Figure 2]

FIREOS is organized as a software module that provides three different services:

- fireos
- normalize_solutions!
- evaluate_solutions

which can be called separately and on-demand from outside. They do not have any dependencies on each other but represent the three main steps that needed to be performed to conduct the experiments.

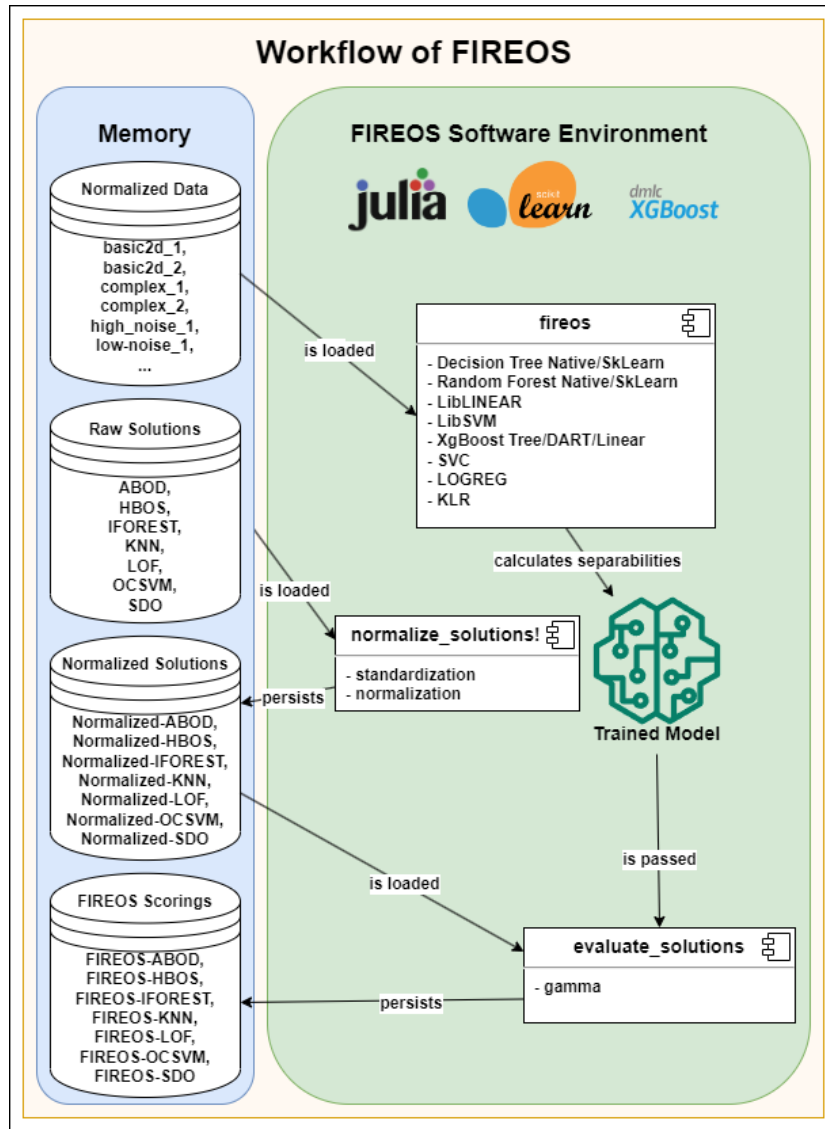


Figure 3.8: Service diagram and workflow of the three main functions of the FIREOS module. 1) **fireos**: Main implementation of the algorithm that calculates the separabilities of a given problem instance and defined predictor. 2) **normalize_solutions!**: Loads raw solution scores from disk and transforms them into normalized solutions that can be interpreted by FIREOS. 3) **evaluate_solutions**: Calculates and persists the final FIREOS scorings from given solutions and a trained model.

Figure 3.8 shows the interrelationships between those three functions as well as a conceptualization of the process from input data and raw solutions to final FIREOS

scorings.

The function "fireos" is pretty self-explanatory as it trains the data with the FIREOS algorithm and parameters and returns the object's separabilities. To prevent misunderstandings "FIREOS" refers to the name of the technique and the main software module "fireos" is the training function of FIREOS. "normalize_solutions!" is called to transform an array of solutions into interval $[0, 1]$.

Different from other functions that will be explained later, this function ends with an exclamation mark (!). This notation is adopted as a property of the Julia Programming Language, where by convention functions with an exclamation mark indicate in-place alteration. In other words, instead of creating a new tuple and returning it, the scaled values are stored inside the tuple that is passed [Mis23d]. This leads to a reduction of memory allocations which is crucial for good performance [Mis23c].

Lastly, the function "evaluate_solutions" assesses specified solutions against a trained FIREOS model.

This interface with all its mentioned functions is implemented by two different libraries called "fireos_lib" and "fireos_lib_par". As the name already implies the FIREOS module consists of a sequential and parallel implementation. Both types possess the exact same feature set and workflow, except the parallel uses multithreaded constructs such as parallel loops and synchronization. Similar to this naming strategy, each function that has a "_par" suffix is part of the parallel implementation which further means that the functions "evaluate_solutions_par" and "normalize_solutions_par" exist.

Common business logic that is executed in both implementations equally is separated into a third file called "fireos_common". A graphical overview of the FIREOS ecosystem and the relationship between its components is provided in [Figure 3.9](#).

The FIREOS module imports several Julia helper packages such as distance functions, libraries for threading and machine-learning algorithm implementations. Besides those native packages, FIREOS features further additions from the ScikitLearn.jl library which provides an interface to the popular Python library Scikit-Learn for Logistic Regression and SVMs. Furthermore, most of the global constants and parameters used during the calculation are defined here. These variables include values for in and outlier classes, maximum recursion depth and some more that are explained in more detail during the following chapters. All experiments for FIREOS were conducted under version 1.0.3.

3.5.2 Normalization Interface

This section is dedicated to the normalization subcomponent of the FIREOS interface: the `normalize_solutions!` or `normalize_solutions_par!` function respectively. In theory, this normalization framework is not directly associated with the original algorithm, since FIREOS and therefore FIREOS assume solutions being probabilistic beforehand. The reason why this relatively small feature is discussed separately is its importance concerning the experiments.

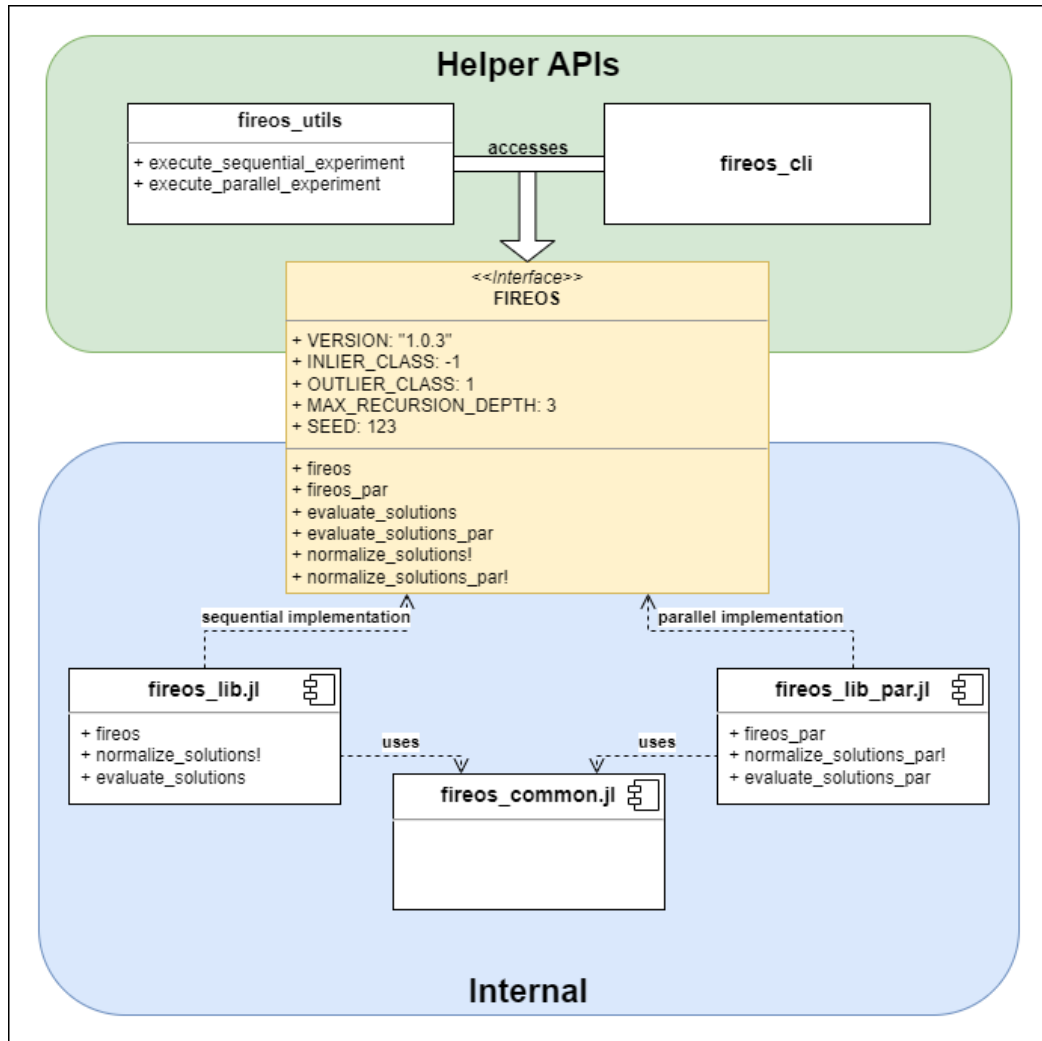


Figure 3.9: Technical conceptualization and relationships between components of FIREOS. Both helper APIs that access the FIREOS module interface can be called from outside. "fireos_utils" provides additional functions to call fireos "out-of-the-box" and "fireos_cli" provides a basic command line interface. All components below the interface definition can be summed up as the backend of the module.

When looking back at the discussion from Section 3.2.1 and Section 3.2.2, the initial situation of the experiments are normalized training data but outlier scores at different scales that need to be unified first. As already mentioned, the initial approach of ensuring that is the unification of outlier scores by a normalization framework [KKSZ11]. However, not every score that is used for the experiments is described in that work. When looking at the original paper, LOF, LDOF, ABOD and KNN are integrated into the

normalization framework. Therefore, for the experiments, those scores are transformed with the corresponding regularisation metrics mentioned in the paper.

SDO, OCSVM, HBOS and IForest, however, are outlier scoring techniques that provide non-probabilistic scorings but are not covered by the initial approach. In order to solve this problem nonetheless, the original framework is extended by the missing predictors under consideration of initial principles which are:

1. Transforming the score into the interval $[0, 1]$
2. Normalized scores can be interpreted as the probability of being an outlier
3. The distribution of scores should favor inliers rather than outliers

Since there is no specific treatment mentioned in the original paper as well as no scientific evidence directing to a logarithmic scaling when looking at the histograms of each score, all measures are transformed by basic linear regularization.

As Section 3.10 shows, only HBOS possesses this specific characteristic that a large number of mediocre scores exist. On the other hand, for all the other scores the histograms look similar. There is a rather high number of scores that are close to zero and therefore inliers where only a few values reach one.

As already explained in Section 3.2.2, the regularization step is followed by another scaling method that ultimately transforms the result into $[0, 1]$. The original framework suggests multiple scaling techniques for this second step. The FIREOS module can handle two of them:

- normalization
- standardization

Normalization or min-max normalization applies a linear transformation where the smallest value of the data is mapped to 0 and the highest to 1 [KKSZ11]. This technique does not assume any underlying distributions and therefore does not add any contrast.

Standardization or Gaussian Scaling on the other hand does exactly that. The underlying data is assumed to occur in a normal distribution having two degrees of freedom (the mean μ and the standard deviation σ). After fitting the input data into this statistic, it is transformed into $X \sim \mathcal{N}(0, 1)$. Different from normalization, standardized values are able to exceed 1 or be less than 0.

These properties, however, are problematic, since input scores for FIREOS should be interpretable as probabilities and therefore between 0 and 1. Although standardization is added as a feature to the FIREOS implementation, exclusively normalization is applied

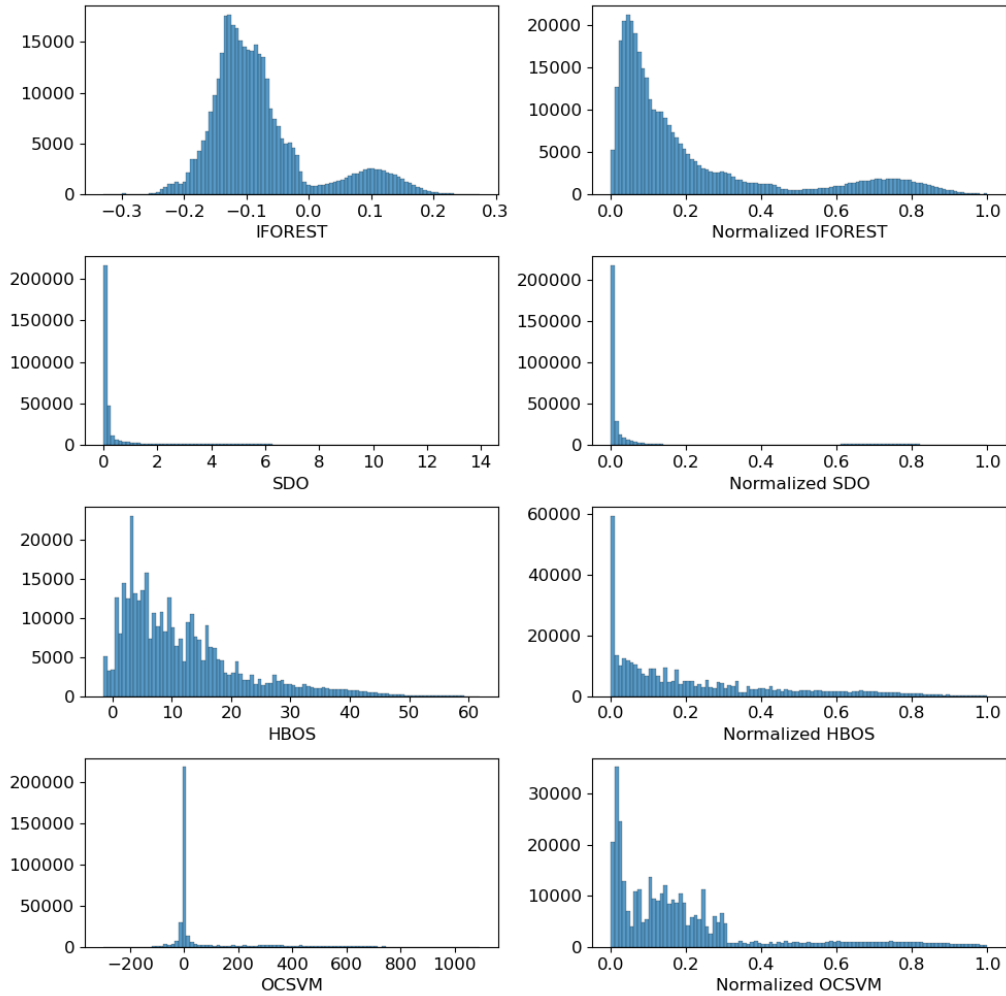


Figure 3.10: This figure serves as an extension to [Figure 3.3](#) as it shows all the remaining outlier scores that are not mentioned in [\[KKSZ11\]](#) but are treated in this work. The column on the left shows histograms of raw unprocessed scores differently from the right column which shows the same scores but normalized. Each histogram consists of 100 bins.

to the scores for each experiment. When looking from a more technical perspective, both the sequential and parallel implementations share a common function definition:

```
normalize_solutions!:
```

- **Inputs:**

- **solutions:** A tuple of the solution matrix and list of scoring algorithm names
- **norm_method:** A scaling method
- **Output:** A tuple of the scaled solution matrix and list of scoring algorithm names

The only differentiation can be observed as the parallel solution is able to normalize multiple algorithms at once. Since this work features seven different outlier scores, the optimal speedup would be achieved with seven cores.

3.5.3 FIREOS Implementation

As preliminaries about the FIREOS interface were provided in the previous section, this chapter concludes with an in-depth explanation of the technical context of the implementation. After normalizing all outlier scores, the FIREOS predictor is ready to be trained. To initiate the training phase, the second function "fireos" is called, which is also the most complex of them. "fireos" is defined by:

fireos:

- **Inputs:**
 - **X:** A matrix that represents input data
 - **clf:** A name of the internal predictor
 - **kwargs:** A dictionary of further hyperparameters
- **Output:** A numerical vector of separabilities

The signature of this function is inspired by the "fit" function from the popular Scikit-Learn library as it performs all calculations on the input data and returns the fitted model. In the case of internal evaluation, this fitted model is an array having the same size as the number of training samples. Intuitively, it stands for the separability of each instance in relation to the rest of the data and acts as weights for the outlier scores it is going to evaluate. Furthermore, it can be directly processed further as it also comes in the same format as the input of "evaluate_solution" which can be associated with the "transform" function in Scikit-Learn.

One major advantage of this architecture is that outlier scores can be saved as a comma-separated file on the disk and reloaded at any time. Moreover, a fully trained model can be evaluated on multiple solutions without any retraining. The downside of fitting the entire model regardless of the scores that are evaluated afterward is, that especially for outlier detection many scores might be zero. For these instances, training could be skipped in theory since they always remain zero and therefore, do not contribute to the

final score. However, since FIREOS is evaluated on a set of scoring algorithms that mostly return non-zero results in the first place, an on-demand training approach was discarded. Nonetheless, the original IREOS implementation in Java trains the algorithm only for non-zero instances as well as for each solution independently.

Besides passing the training data, the "fireos" function requires another mandatory parameter, which is the classifier. [Table 3.2](#) shows all possible classifiers that are available in both implementations as well as their underlying model and characteristics.

Table 3.2: Different classifier options in FIREOS and their characteristics

Classifier Parameter	Underlying Model	Characteristics
svc	SVM	SVM and RBF Kernel in SkLearn
libsvm	SVM	Linear SVM using LibSVM
logreg	Logistic Regression	LR in Scikit-Learn
klr	Logistic Regression	Same as above using RBF Kernel
liblinear	Logistic Regression	LR using LibLINEAR
decision_tree_native	Decision Tree	Native DT
decision_tree_sklearn	Decision Tree	DT using Scikit-Learn
random_forest_native	Random Forest	Native RF
random_forest_sklearn	Random Forest	RF using Scikit-Learn
xgboost_tree	XgBoost	XgBoost with Random Forest
xgboost_dart	XgBoost	XgBoost with DART Booster
xgboost_linear	XgBoost	XgBoost with Linear Booster

Besides these technical differences, each classifier has different hyperparameter settings and intuitions which are already discussed in prior sections. More information in regard to predictors based on Logistic Regression and SVMs can be obtained in [Section 3.3.1](#) and [Section 3.3.2](#) respectively. Tree-based classifiers were already discussed in [Section 3.3.3](#) and LibLINEAR in [Section 3.3.4](#).

Before the training starts, the specified classifier parameter is mapped into a function, which provides the predictor's implementation as well as the transformation into probabilities. Since each predictor has its own way of providing results (probabilistic vs. non-probabilistic, non-uniform output) this function must also contain corresponding methods to convert each output into a common form. Passing a function as a parameter that is dynamically executed is a key language feature of Julia. Using this extra level of abstraction enables simple maintenance of hyperparameters as well as the implementation of additional predictors in the future.

Besides the specification of the internal predictor and its hyperparameters, FIREOS features its own parameters as well. Some of them were already established for IREOS as some are additional tuning parameters for complexity reduction. Except for one value, all of these parameters are optional arguments, that in case of absence have default values as can be observed in [Table 3.3](#).

Table 3.3: Additional tuning parameters of FIREOS

Name of Parameter	Mandatory	Datatype	Default Value
<code>adaptive_quads_enabled</code>	Yes	Boolean	<i>nothing</i>
<code>window_size</code>	No	Integer	<i>n</i>
<code>gamma_min</code>	No	Float	0.0
<code>gamma_max</code>	No	Float	Varies by Predictor
<code>tol</code>	No	Float	0.005

"`adaptive_quads_enabled`" is mandatory since it acts as a crucial tuning parameter that makes a big impact on the training of FIREOS as it controls how separabilities are calculated.

The original IREOS paper uses adaptive quadrature as a numerical integration method to estimate the area under the curve of the separability over different gammas. One main advantage of this method is that the γ hyperparameter for the RBF Kernel of the KLR does not have to be randomly guessed or set beforehand, which leads to more precise calculations and expressive results.

However, the major downside is that the number of predictors that need to be trained rises exponentially for estimations that do not fulfill the tolerance criterion as the recursion of adaptive quadrature continues. In the worst case, the number of trained predictors rises:

$$\#ofPredictors = O(2^n) \quad (3.7)$$

where n is the depth of the recursive tree.

In order to prevent memory overflows as well as unknown runtimes for non-converging problems, the global constant `MAXIMUM_RECURSION_DEPTH = 3` is defined. Although some curves might not reach sufficient approximation goodness, the trade between a higher recursive depth and performance loss does not pay off. Even after further optimizing the algorithm so that separabilities of each γ are calculated exactly once, setting a static recursion limit to prevent theoretic endless calculations is necessary.

Besides performance reasons, the mechanism of numerical integration became problematic for a different reason too, which concerns predictors that do not consist of a hyperparameter γ . Closely considered, the γ parameter is a remainder of nonlinear predictors featuring an RBF kernel function and originates from the idea of KLR in the original work. Tree-based and linear predictors, however, do not accept such value and therefore do not benefit from this adaptive quadrature, which ultimately is the reason to include the FIREOS parameter "`adaptive_quads_enabled`". In order to prevent any unnecessary calculations, the user is able to enable or disable the mechanism of numeric integration by setting `adaptive_quads_enabled`. If this value is set to *true*, adaptive quadrature following the example of IREOS [MCSZ20] is performed. When

"adaptive_quads_enabled" is *false*, no numerical integration is made leading to the separability of only one predictor being calculated.

The meaning of the remaining parameters "gamma_min", "gamma_max" and "tol" which originated from IREOS, are therefore also quite intuitive as they enable the user to pass custom values for the boundaries for the adaptive quadrature. The original intention of retaining these parameters in FIREOS is to create an implementation that is as close as possible to the initial Java implementation to ensure equal conditions for the experiments.

Furthermore, "gamma_min" is set to 0.0 by default in the original implementation, which would lead to crashes in FIREOS when using the SVM predictor of Scikit-Learn [Lea23]. To prevent this problem, FIREOS intercepts these cases by changing the value from 0.0 to 0.0001.

"gamma_max" describes the maximum gamma of the RBF kernel and "tol" sets the tolerance for an estimated approximation error.

The last hyperparameter that can be passed is "window_size". Differently from those that were explained before, "window_size" works the same regardless of any other setting. It serves as an additional control parameter for featuring sliding windows, which was already introduced in Section 3.4.1. The main purpose of this parameter is to compare the influence between non-sophisticated subsets and the entire training data in:

- Execution time during training
- Quality of evaluated solutions

The passed argument represents the size of the sliding window. If training should be performed on the entire dataset nonetheless, the user can leave out the argument or pass the total number of samples manually. Finally, after fitting the FIREOS model, the second part of the interface is invoked:

evaluate_solutions:

- **Inputs:**

- **fireos:** A trained FIREOS model
- **solutions:** A tuple of the scaled solution matrix and list of scoring algorithm names
- **gamma_min:** "gamma_min" or 0.0
- **gamma_max:** "gamma_max" or 1.0

- **Output:** A dictionary of scoring algorithms and evaluation

"evaluate_solutions" calculates the goodness of each outlier score by multiplying each scoring vector against the vector of separabilities from FIREOS. Calling this function marks the last step of the FIREOS pipeline as the output consists of the evaluated scores normalized into $[0,1]$. From a technical point of view, this function is similarly built like the normalization function except it does not change values in place, which encourages the reuse of the separabilities or scoring vectors.

3.5.4 Caveats and Limitations

Although most of the important implementation details were already explained in Section 3.5.3 and Section 3.5.2, this section is purely dedicated to complications and technical limitations that occurred during the implementation. Since some of these problems are the cause of several unique results that are shown in Chapter 4, it is important to mention and explain those.

```
function get_svm_clf_par(X, y, outlier_index, gamma, T)
    #SVC cannot deal with gamma == 0
    if gamma == 0.0
        gamma = 0.0001
    end

    # lock(func, lock) did not work
    outlier_prob=0.0
    lock(1)
    try
        outlier_prob = sk_svm_par(X, y, reshape(X[outlier_index, :], (size(X)[2],1)), gamma)
    finally
        unlock(1)
    end
    return outlier_prob
end
```

Figure 3.11: Excerpt of the SVM predictor function in parallelized FIREOS. To prevent PyCall from crashing all Scikit-Learn functions are outsourced into the function "sk_svm_par" and called sequentially.

As explained previously and indicated by the names of the internal predictors, FIREOS consists of native and foreign libraries. However, the Scikit-Learn implementations of Logistic Regression and SVM are imported by the custom macro "@sk_import" which loads the original Python version of the model directly. When the Julia wrapper of Scikit-Learn calls the underlying model implementation during training, the library uses a package called "PyCall" in the background which can "*directly call and fully interoperate with Python from the Julia Programming Language*" [Joh23].

During the experiments training the classifiers for SVM and Logistic Regression resulted in unstable behavior as well as multiple crashes due to memory access violations inside PyCall. Especially when called by multiple threads at the same time, PyCall crashes most of the time. After some research, it became clear that the PyCall package is not

threadsafe since it does not release the Global Interpreter Lock (GIL), which is a mutex that allows only one thread to hold control of the Python interpreter at a time. In other words, Python allows only one thread to be in a state of execution at any time [Aji23]. When the second thread enters the PyCall interface and tries to acquire the GIL, SIGSEGV is returned and Julia crashes.

This behavior is very bad for the performance of those predictors since multithreaded access is not possible. In order to still be able to run the algorithm in Julia, a lock is added around the critical section. Figure 3.11 shows the location of the critical section for the SVM classifier in the code.

This, on the other hand, creates a severe bottleneck since a function that takes up to 99% of the execution time needs to be accessed sequentially. It can be observed that most of the execution time of FIREOS takes place inside the PyCall package. The consequence of this measure is a massive reduction in performance for the multithreaded implementation possibly leading to zero speedup in comparison to the sequential solution.

Furthermore, although PyCall is strictly synchronized in version 1.0.3 of FIREOS, random crashes do still occur occasionally during the training of larger datasets. The sequential implementation, however, does not suffer from these failures and is clearly recommended for the classifiers concerned.

Another important fact that should not be left unmentioned in case of reproducing runtimes is the startup behavior of Julia. Since Julia is a dynamic language it is prone to the invalidation of external code [Hol21]. In order to mitigate this issue and stay as flexible as possible, every time a new Julia prompt is started, packages are recompiled (precompiled) before the actual code can be executed. This precompilation is far from trivial since packages and functions are precompiled on demand. Especially for small problems these precompilation tasks may take a large amount of time and distort the actual performance of the code.

Dataset	libsvm-0.1-sequential	libsvm-0.1-parallel	libsvm-0.5-sequential	libsvm-0.5-parallel	libsvm-1.0-sequential	libsvm-1.0-parallel	decision_tree_native-0.1-sequential	decision_tree_native-0.1-parallel
basic2d_1	0.963	0.642	0.769	0.16	1.71	0.314	0.461	0.638
basic2d_10	0.201	0.0427	0.785	0.159	1.78	0.313	0.0208	0.00237
basic2d_11	0.205	0.0448	0.798	0.168	1.67	0.319	0.0237	0.00221
basic2d_12	0.247	0.0456	0.794	0.161	1.8	0.322	0.0185	0.0019
basic2d_13	0.249	0.0431	0.749	0.156	1.63	0.321	0.0113	0.0033
basic2d_14	0.201	0.0922	0.788	0.16	1.68	0.319	0.0127	0.00322
basic2d_15	0.204	0.0447	0.802	0.164	1.54	0.315	0.0107	0.00297
basic2d_16	0.201	0.0431	0.784	0.155	1.69	0.321	0.0101	0.00309
basic2d_17	0.203	0.0452	0.843	0.174	1.68	0.326	0.00945	0.0021
basic2d_18	0.198	0.042	0.75	0.149	1.6	0.312	0.00943	0.00166
basic2d_19	0.204	0.0441	0.817	0.165	1.81	0.334	0.0163	0.00192
basic2d_2	0.2	0.0448	0.747	0.158	1.61	0.301	0.0154	0.00178
basic2d_20	0.201	0.0456	0.758	0.161	1.66	0.311	0.0173	0.00201
basic2d_3	0.251	0.0472	0.8	0.171	1.66	0.334	0.0176	0.00166
basic2d_4	0.244	0.0414	0.722	0.147	1.64	0.306	0.00962	0.00245

Figure 3.12: Excerpt of different runtimes in seconds for smaller datasets. Outliers in the first row of the table show the impact of precompilation tasks in Julia.

Figure 3.12 shows this phenomenon and its extent on recorded runtimes for small datasets. It can be clearly observed that both sequential and parallel runs for the 0.1 window of the first row show significantly higher runtimes than any other run. The column-wise IDE coloring visualizes this effect even stronger as it paints high runtimes in blue and smaller more reddish. This is due to the batch execution of all 20 basic2d datasets. The execution starts with each algorithm having a window size of 0.1 of all "basic2d_1" datasets. All of the following measures do not need any precompilation and make them appear more consistent than the first one. As shown in the table, the omission of these tasks results in a speedup of more than 300 in some cases. For fairness reasons, this precompilation time is kept for the first basic2d dataset. Experiments featuring datasets of the complex, high-noise or low-noise series are not affected as much since training the algorithm takes up most of the time.

3.6 Description of Experiments

Now that every step of the FIREOS evaluation pipeline is explained in theory and a more technical context, this section is dedicated to the specific setup of the experiments. As already mentioned, all experiments are run on 80 different datasets of various sizes and conducted on a series of methodology steps (S):

- (S1): FIREOS with nine (LibSVM, Decision Tree Native, Decision Tree SkLearn, LibLINEAR, Random Forest Native, Random Forest SkLearn, XgBoost Tree, XgBoost DART and XgBoost Linear) out of its twelve classifiers is **run on each dataset and evaluates the solutions of seven outlier detection algorithms** (ABOD, IFOREST, HBOS, KNN, LOF, OCSVM and SDO). The three remaining predictors (SVC, LOGREG and KLR) are not considered since their runtimes are infeasible for the problem sizes tested.
- (S2): The exact same solutions and datasets are evaluated with the **original IREOS implementation in Java** [Mar20].
- (S3): A third series of experiments is conducted by assessing the same outlier solutions by **five external evaluation metrics**. (Adjusted P@n, Adjusted Max F1 Score, Adjusted Average Precision, Area Under the ROC Curve and Adjusted Mutual Information)
- (S4): Finally the quality of results and runtimes from **FIREOS (S1)** are compared against:
 - a) **IREOS (S2)**
 - b) **External metrics from (S3)**

All experiments are executed on an AMD Ryzen 9 7950X, 16C/32T, 4.50-5.70GHz and 64Gb DDR5 RAM.

3.6.1 (S1) First Series of Experiments: FIREOS

The first series of experiments focuses on benchmarking the novel implementation. Each problem instance (dataset) comes with the corresponding solutions of seven outlier detection algorithms. (ABOD, IFOREST, HBOS, KNN, LOF, OCSVM and SDO) FIREOS is individually trained on each dataset and assesses those seven solutions by first creating a scoring between 0 and 1 as well as ranking of these scorings. A FIREOS score close to 1 indicates high agreement on the outlying data points by both FIREOS and the corresponding anomaly detection algorithm. A score close to 0 means that most of the detected outliers by the algorithm are considered inliers by FIREOS and therefore not a favorable solution. FIREOS uses all its underlying predictors except "svc", "logreg" and "klr" as they are too slow to be considered effective contenders.

Besides assessing the result's quality, FIREOS furthermore tracks the overall execution time of each run during training and evaluation. These numbers are aggregated for both the sequential and parallel implementation and exactly reproducible for most of the classifiers due to a fixed seed. Most of the classifiers since both LibSVM and LibLINEAR do not feature a mechanism that ensures entirely equal results when executed with the same seed. To still gather the exact runtimes, each experiment is conducted multiple times on the same machine.

Section [3.7](#) provides a more detailed explanation of how exactly this execution time is measured for each implementation.

3.6.2 (S2) Second Series of Experiments: IREOS

In the second series of experiments, the same 80 datasets from before are run by the original IREOS algorithm. The procedure is very similar to the previous setting. As before the identical outlier solutions are loaded and assessed but this time by the original IREOS implementation in Java [\[Mar20\]](#). Like before, all the runtimes are measured for each run. However, in order to keep both procedures comparable, some adjustments to the Java code had to be made and are documented and explained in Section [3.7](#).

Different from the FIREOS module, the IREOS code is optimized for parallel processing and does not feature a sequential implementation.

3.6.3 (S3) Third Series of Experiments: External Validation

After performing internal evaluation of 80 instances by FIREOS and IREOS this final series of experiments is conducted by the assessment of external validation indices. As already mentioned in Section [2.2.1](#) this work uses a common methodology for the evaluation of scores to [\[CZS⁺16\]](#). Differently to the previous two series of experiments, all seven outlier solutions are now evaluated by five state-of-the-art external evaluation metrics, which consist of:

- Adjusted P@n

- Adjusted Max F1 Score
- Adjusted Average Precision
- Area Under the ROC Curve
- Adjusted Mutual Information

Same as before each validation metric assesses each solution by an evaluation ranking and scoring.

3.6.4 Evaluation of the Experiments

After performing the experiments from before, all the gathered data can be summarized as three batches of results:

1. Internal validation results by FIREOS
2. Internal validation results by IREOS
3. External validation results by state-of-the-art algorithms

When going back to the original research questions, the main goal of this work is to show the **improvements of FIREOS in regard to the runtime as well as the goodness of solutions**. Now that all the important data to answer those questions are collected, the methodology for the evaluation of those experiments can be presented before the results are published in Chapter [4](#).

First, there is the assessment of quality. In order to gain a better understanding of the nature of each predictor in FIREOS, we have to compare it against a ground truth. If no ground truth is available for the datasets to be evaluated against we have to define it. In this work, we approach this by the creation of a metric originating from empirical evidence which is an averaged score from all five external evaluation algorithms and named "consensus" in further results. In other terms "consensus" is a metric that summarizes collective agreement from common algorithms. Since both FIREOS as well as IREOS provide assessments on the same seven outlier detection solutions of each dataset, we can finally evaluate these scores against each other as well as against this novel performance index.

Then there is the assessment on speed. Similar to the first evaluation criteria, the speed is also compared between different implementations. However, we do not have a consensus value for the runtime of each problem instance. Therefore, we compare the difference in runtime between the original IREOS implementation and FIREOS. Since IREOS is utilizing multithreading, the runtime of the parallel FIREOS implementation is a fair comparison. Furthermore, we examine the speedup of different internal predictors between the sequential and parallel FIREOS.

Since both, evaluation quality and speed are important benchmark criteria, an optimal predictor for FIREOS is found by taking the results from the correlation matrix as well as runtimes into account.

3.7 Comparableness with IREOS

This chapter deals with the technical differences between the IREOS and FIREOS implementation. Although both algorithms are closely related to each other both implementations were developed with certain intentions that also lead to fundamental differences.

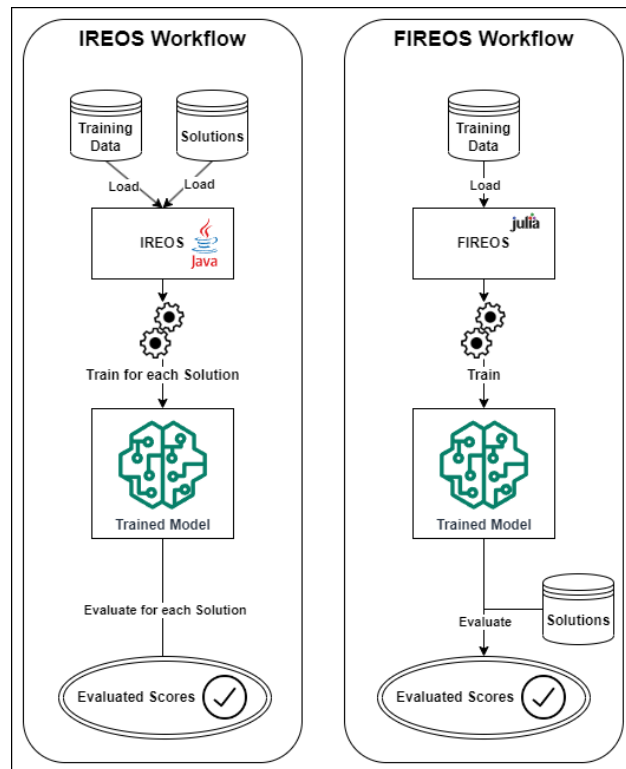


Figure 3.13: Workflows for IREOS and FIREOS. The main difference between them is that FIREOS trains the predictor independent from any solution. Since FIREOS always calculates the separability of all data samples, no solution has to be known during the training phase of the algorithms. After the training, an arbitrary number of solutions can be evaluated at the same time. IREOS on the other hand trains and evaluates once for each solution. In terms of performance, this is a disadvantage when evaluating a large set of different solutions. However, IREOS ignores the separability calculation for data points that are rated as 0 in the solutions, since they do not contribute to the final score. This accelerates the algorithm, especially for sparse solutions.

From the beginning, the architectural design of FIREOS was customized for the experiments, in other words, it is meant to be optimized for the evaluation of multiple large outlier scoring vectors at once. A FIREOS predictor is trained exactly once for all samples and is then able to evaluate an arbitrary number of different candidate solutions.

Differently to this approach, the original IREOS implementation is trained for each solution which would be infeasible here since the classifier would be retrained seven times for each dataset. Nonetheless, as already discussed earlier, this behavior is beneficial for on-demand evaluation. Figure 3.13 shows the similarities and differences between the workflows of both IREOS and FIREOS.

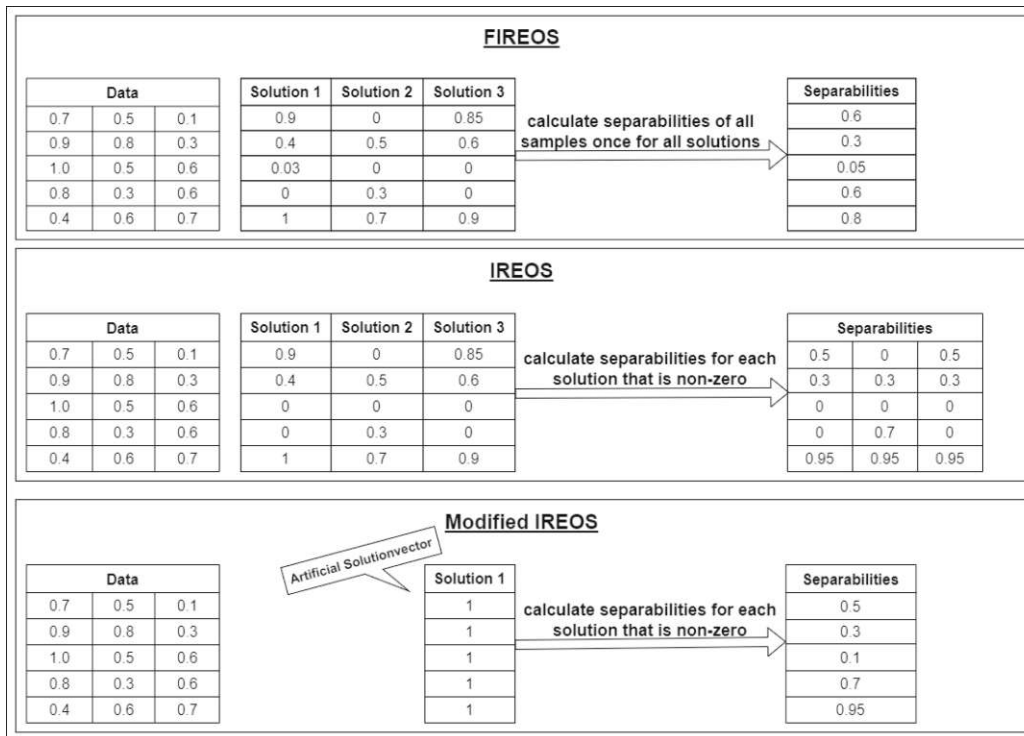


Figure 3.14: Examples of separability calculations between FIREOS and IREOS. Although the same data and solution matrix are used, FIREOS always outputs exactly one separability vector but IREOS one for each solution. Furthermore, the separability matrix of IREOS contains zero for any zero in the solution vector, since IREOS ignores those instances. In order to modify IREOS to behave similarly to FIREOS, we instead of passing the solution matrix, provide an artificial solution vector of ones to ensure the calculation of all samples and force IREOS to calculate a separability vector rather than a matrix.

Since these limitations exist, a direct comparison of both implementations would result in unfair conditions. To still solve this problem, the original implementation is extended by

a client wrapper (IREOScli), which changes the course of processing to fit the workflow of FIREOS. Instead of directly passing each solution beforehand, integrating it into the training phase and evaluating one solution at a time, a different approach is implemented. Rather than training and evaluating sequentially, which is the default behavior, we use an artificial solution vector of ones instead. Since IREOS ignores the separability calculation for samples rated as 0 in the solution vector, our artificial solution forces IREOS to train on all the data samples once, which is equivalent to the behavior of FIREOS. [Figure 3.14](#) visualizes the differences of FIREOS and IREOS calculations and shows the application of this artificial solution vector.

After the training phase, IREOS evaluates the calculated separabilities on an arbitrary number of solutions, just as FIREOS. The domestic multithreaded structure and synchronization as well as the integration of the internal KLR library remains unchanged from any adjustment.

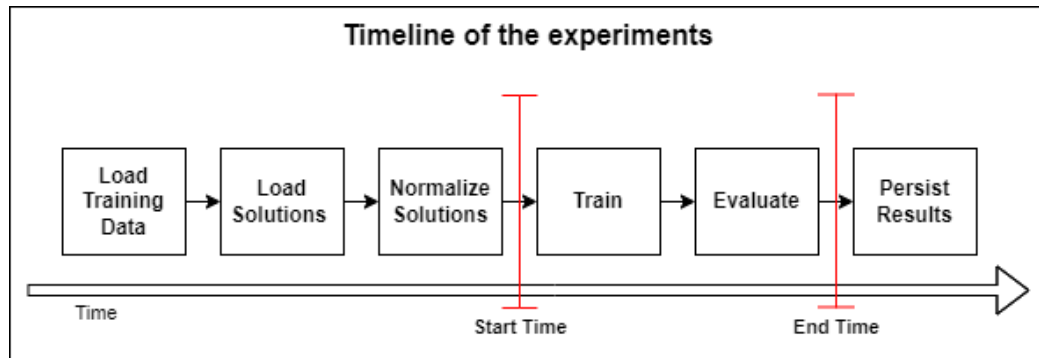


Figure 3.15: Outline of timeline and runtime measuring during the experiments.

Since the modus operandi of both methods are finally aligned now, a more precise explanation about how the timings are measured is provided. Although I/O costs might be noticeable, especially for larger datasets, the timings in [Chapter 4](#) do not include any load and save operations. This is due to the main intention of providing an accurate analysis of the algorithmic complexity in real-life applications as well as possible random variations for runtimes. In other words, all timing and speedup measurements are started right before the model is trained and stopped when all seven outlier scores are properly evaluated. [Figure 3.15](#) shows a simplification of the evaluation process and the start and end time of all runtime measures during the experiments.

Results and Discussion

In this chapter, we present and visualize all the results obtained from the experiments to the reader. Furthermore, we discuss and interpret the research questions defined in Section 1.3. The structure of the experiments is twofold. We lay our primary focus on the quality of evaluation and how well each FIREOS predictor performs when compared to other evaluation techniques. Then, we contrast these outcomes with their associated execution time and elaborate on what time each predictor is able to evaluate an arbitrary solution on a given problem of a certain size. Finally, we clarify the amount of speedup of each predictor when executed on a multicore machine.

4.1 Evaluation Performance

As already established in Chapter 3.6 we lay our focus on the evaluation of seven different outlier detection algorithms. (ABOD, IFOREST, HBOS, KNN, LOF, OCSVM and SDO) Each dataset presented in Section 3.1 comes with a total of seven outlier solutions - one for each algorithm. We now run our novel FIREOS implementation on those datasets one by one and evaluate the quality of each outlier solution. The resulting FIREOS scores range between 0 and 1 and describe the amount of consensus with the corresponding outlier detection solution. A higher score is an indication that FIREOS agrees more with the corresponding solution and a lower score means less agreement. In the end, we gather 9 different evaluations of 7 solutions for each instance by FIREOS since the former is the number of predictors considered in our experiments. (LibSVM, Decision Tree Native, Decision Tree SkLearn, LibLINEAR, Random Forest Native, Random Forest SkLearn, XgBoost Tree, XgBoost DART and XgBoost Linear)

Besides the scorings by FIREOS, we let the original IREOS algorithm evaluate each solution as well and collect the resulting IREOS scores. Ultimately since MDCGen provides labels for both inliers and outliers for each dataset we can utilize this labelling to calculate scores of outlier solutions by external validation methods. As already stated,

we consider 5 external evaluation methods as a comparison in our experiments. (Adjusted P@n, Adjusted Max F1 Score, Adjusted Average Precision, Area Under the ROC Curve and Adjusted Mutual Information)

Although we already unified the outlieriness scores of each algorithm by Kriegel's framework to ensure fair evaluation by (F)IREOS, we again face the problem that not all validation outputs are on the same scale which prevents us from comparing their similarity and distribution. Therefore, we require performance metrics that lie on a common scale, are comparable and preserve the order of evaluated scores.

In this work, we introduce two different performance indicators that are used to assess the quality of different evaluation methods:

- Z-Scores
- Ranks

Z-Scores are statistical measures that describe the relationship of a value to the mean of a series of data points. The value 0 indicates that the examined score is equivalent to the arithmetic mean, a value of 1 means one standard deviation above the mean. Z-Scores might be negative as well, representing scores below the mean. Furthermore, they have the property to be a comparable metric although the underlying accumulations of data points may consist of different sizes or scales. In our experiments, one series of data points consists of the 7 evaluated outlieriness scores by one evaluation method.

The second performance indicator that is used for the experiments is ranks. We create our ranking by using the Z-Scores from before and ordering their values. Higher scores result in higher ranks where equivalent scores share the same rank.

From both scores and ranks we compute pairwise distances of each series and summarize them as distance matrices. These distance matrices share the same properties as correlation matrices being symmetrical and positive semidefinite. Its diagonal equals 0 since the distance of a point to its identity is always 0 by definition. Distance matrices of Z-Scores can be compared with correlation matrices, however, their intuition is more or less the opposite. A higher correlation usually results in data points being linearly more similar, which is closely related to small distances. Higher distances, on the other hand, indicate high dissimilarity and point to disassociation. In summary, those distance matrices describe the mean absolute error between normalized validation results of 7 outlier detection algorithms.

[Figure 4.1](#) shows this calculation pipeline for a simplified problem. An example instance "Dataset 1" consists of two independent variables or features and one target variable which is the labeling. As already mentioned each data point that is assigned to a positive number belongs to a cluster and is therefore an inlier. Outliers on the other hand are outlined with the "-1" label. In order to obtain the final distance matrix of Z-scores a series of steps and calculations are performed:

1. We run several outlier detection algorithms on this dataset, which create multiple vectors of different solutions. (During the experiments we use seven algorithms but for the visualization due to lack of space and simplicity only three.)
2. We calculate the evaluation for each of the three solutions (seven for the experiments) with four evaluation methods (For the experiments we use 16) belonging to three different validation categories:
 - One FIREOS variant (9 for experiments)
 - One IREOS variant
 - Two external validations (6 for experiments)
3. We use Z -Score standardization for the three solutions (7 for experiments) so that they are in the same range for all evaluation methods. In the visualization, each evaluation method is bordered on the same color.
4. We then calculate the differences of the evaluation scores for each of the three solutions (7 for experiments). So for each evaluation method, you get three (7 for experiments) values and then calculate the difference of evaluations. e.g.
 - Difference between $P@n$ and ROC (for outlier detection method detection method ABOD) = $P@n(ABOD) - ROC(ABOD)$
 - Difference between $P@n$ and ROC (for outlier detection method detection method HBOS) = $P@n(HBOS) - ROC(HBOS)$
 - Difference between $P@n$ and ROC (for outlier detection method detection method KNN) = $P@n(KNN) - ROC(KNN)$
 - For the experiments differences are also calculated for IFOREST, LOF, OCSVM, and SDO.

With these calculations, we get three (7 for experiments) differences for each evaluation comparison pair (e.g. AUC, $P@n$, IREOS,..)

5. We then average those three (7 for experiments) differences to get the difference between evaluation methods. [Figure 4.1](#) shows each of the calculations and describes the average as "mean" function.
6. For the experiments, we take the mean of those average differences over the 80 data sets which is the value that is put in the matrix. (This step is not shown in the figure since we only consider one dataset for the example.) Nonetheless, for the corresponding dataset, each pair of average distances is presented in red.

The calculation of the rank matrices is very similar to those of the Z -Scores. Instead of calculating evaluation scorings in step 2, we rank the three (7 for experiments) values from the solutions. If we take $P@n$ from [Figure 4.1](#) we can observe the scores: 1.0 ABOD,

0.87 HBOS, 0.6 KNN. In other words, P@n finds that ABOD is the best solution (highest score) and therefore ranks it first. The second rank would be HBOS with a score of 0.87 and the third KNN with 0.6. In the experiments, we rank each solution from 1-7 since 7 outlier detection algorithms instead of three are considered. Those ranks are then used for the difference calculation in step 4 as we consider differences over ranks instead of differences over Z -Scores.

Although the general intuition of distance matrices is comprehensible, an in-depth interpretation and elaboration of such values remains delicate. First, there is the nonlinear nature of Z -Score distances. A distance of 0.2 is not twice as similar or "accurate" as a distance of 0.4. In general terms one can agree that lower scores are more similar to a certain degree, however, the expressiveness of each scoring is limited to unknown dynamics which depend on each instance. This phenomenon can also be projected on rank distances as perfect similarity between ranks does not automatically mean equivalent quality of scoring. [Figure 4.2](#) shows some examples where similar ranks and scorings might lead to wrong conclusions.

To interpret matrices of Z -scores, we need to consider that the value of a cell represents an average difference between two evaluation methods (e.g., IREOS and AUC) when validating the solution of an outlier detection algorithm (e.g., LOF). As already mentioned before, these evaluated scores are comparable due to normalization on variance (i.e., Z -weighted) which further means that cell-values must be directly interpreted as "times of standard deviations". (in their respective scales) Different from matrices of scores, cell-values of rank matrices show the average difference in positions of the same solution given by two different evaluation methods.

The interpretation of cell-values of rank matrices is considerably more intuitive. For instance, a cell-value equals 1 means that two evaluation methods (e.g., IREOS and AUC again) on average disagree in one position when evaluating the quality of a random solution among the compared algorithms. That is, if IREOS finds LOF as the second-best algorithm, AUC will tend to place in the first or third position. In contrast, the interpretation of Z -scores is more complicated since the original scores given by an evaluation method can be very unevenly distributed. For instance, it can happen that a given evaluation method scores algorithms like in case *A*: 0.100, 0.101, 0.102, 0.103, 0.104, 0.105, 1.000, case *B*: 0.100, 0.100, 0.100, 0.100, 0.100, 0.100, 0.200, case *C*: 0.100, 0.200, 0.300, 0.400, 0.500, 0.600, 0.700, etc. where each case entirely different distributed. This behavior leads to the assumption that one should exclusively consider matrices of ranks as an accurate benchmark. However, as shown in [Figure 4.2](#) rank matrices tend to be misleading and inaccurate in certain cases as well. Given the nature of the task, the scores of case *C* would seem rather unusual, since case *A* and *B* are quite close to each other in terms of scores. When considering ranks case *C* entirely agrees with case *A* and case *B* would be the stranger. On the other hand, strong discrepancies might also be quite possible (for example, perhaps the specific type of studied anomaly is only visible for one algorithm in the comparison, or the other way round and even more common, perhaps most algorithms solve properly the task, but one failed completely). Thus,

average distances about 1 standard deviation in matrices of Z -scores and/or average distances about 2 ranks in matrices of ranks are significant, but we might get an inflated perspective of the difference if we forget the context of the comparison and the type of data that we are analyzing.

Therefore, we always include both score and rank distances for each evaluation of the same 7 outlier detection algorithms in all experiments. Furthermore, as described in Section 3.6.4, the additional metric "consensus" describes the average external validation score. Besides absolute scorings, also the distance between the ranked results is calculated. Figure 4.3 and Figure 4.4 show these distances for basic2d and Figure 4.5 as well as Figure 4.6 for the remaining datasets. Figure 4.7 shows the variance of the distance matrix between complex, high-noise and low-noise datasets. Until now we only considered outlier evaluation with the whole data for our quality study. Figure 4.8 shows the effect of different sliding windows for the most important predictors. In this experiment, we consider window sizes of:

1. 10%
2. 50%
3. All samples

4. RESULTS AND DISCUSSION

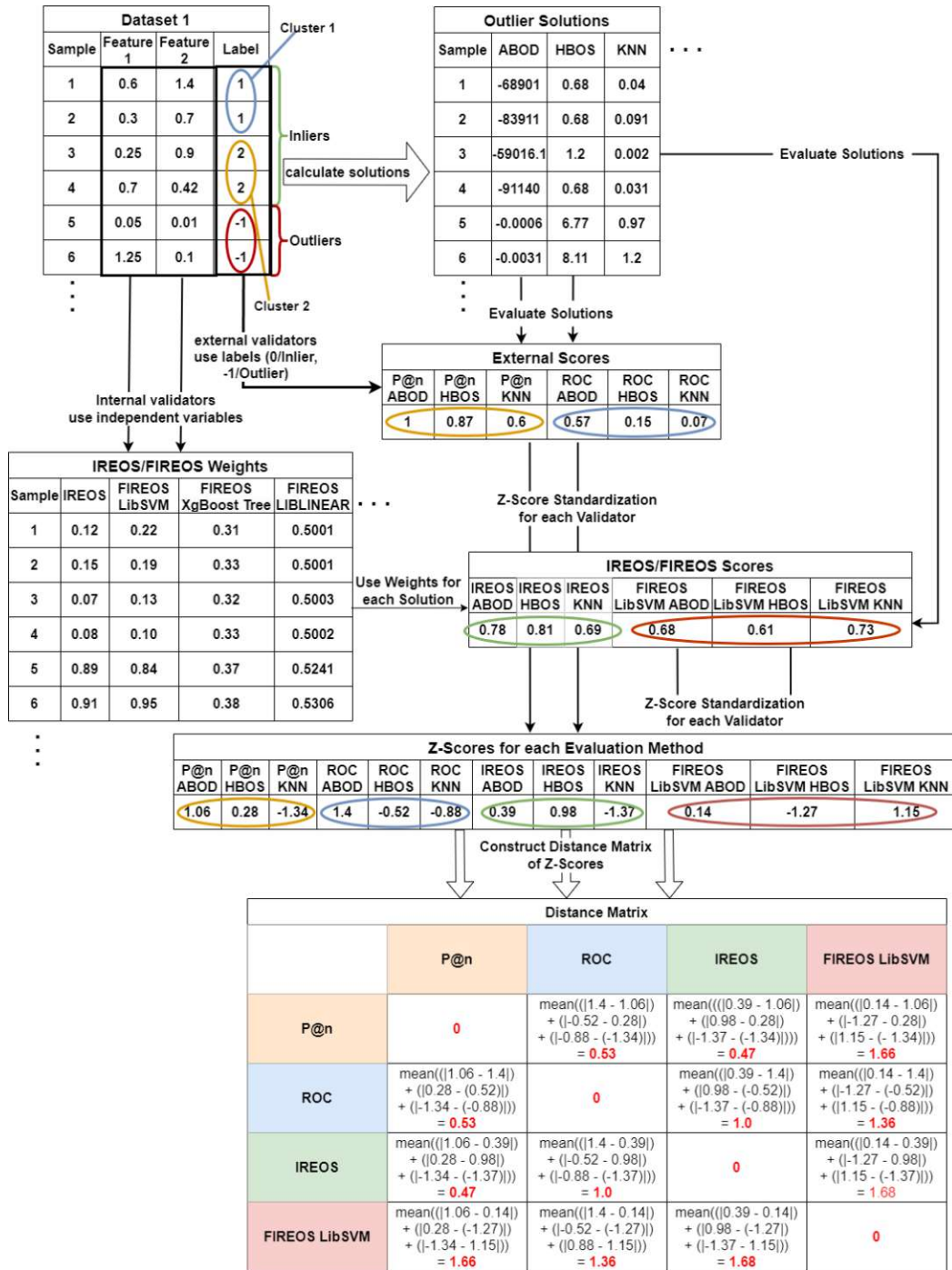


Figure 4.1: Visualization of all the steps we perform to obtain distance matrices of Z-Scores from an example dataset.

Scorings						
#	score1	score2	score3	score4	score5	score6
1	0	0.445	0.12	0	0.01	0.01
2	0.4	0.446	0.471	0.02	0.01	0.5
3	0.5	0.4461	0.47	0.99	0.51	0.51
4	0.55	0.4462	0.469	0.998	0.52	0.52
5	0.6	0.4463	0.468	0.999	0.53	1
6	1	0.447	0.46	1	1	1

Ranks						
#	score1	score2	score3	score4	score5	score6
1	6	6	6	6	5	5
2	5	5	1	5	5	4
3	4	4	2	4	4	3
4	3	3	3	3	3	2
5	2	2	4	2	2	1
6	1	1	5	1	1	1

Z-Scores						
#	score1	score2	score3	score4	score5	score6
1	-1.72	-1.87	-2.24	-1.44	-1.23	-1.71
2	-0.37	-0.17	0.47	-1.39	-1.23	-0.27
3	-0.03	0	0.47	0.69	0.23	-0.24
4	0.14	0.17	0.46	0.71	0.26	-0.21
5	0.31	0.34	0.45	0.71	0.29	1.21
6	1.67	1.53	0.39	0.71	1.67	1.21

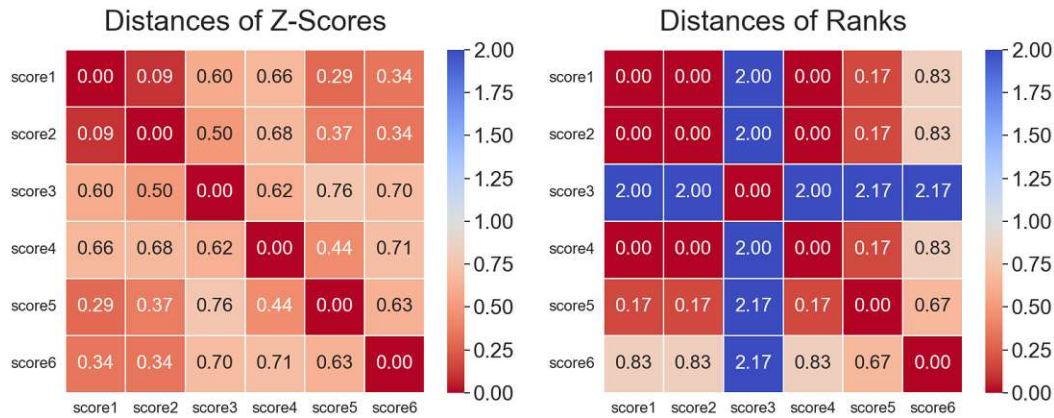


Figure 4.2: Examples of different validation scorings where the consideration of only one distance matrix (scores or ranks) would lead to an incomplete picture: Given are six scoring results of six fictional problem instances where we consider score1 as the gold standard. Score1 and score2 are very similar due to their distance matrices although their initial scorings are very different. Score3 looks very similar to score2 but shows strong deviations when looking at ranks. The distance between score4 and score1 is larger than to score3 and score1, however, the distance between the ranks is completely different. Lastly, score5 and score6 show another ranking problem. Although both predictors "misclassify" exactly one rank (score5 misclassifies row 1 and score6 misclassifies row 5) the outcome of the distance matrix is very different considering the distance of score6 the second highest. The misclassification of higher ranks is penalized more than lower ranks because the error is propagating further.

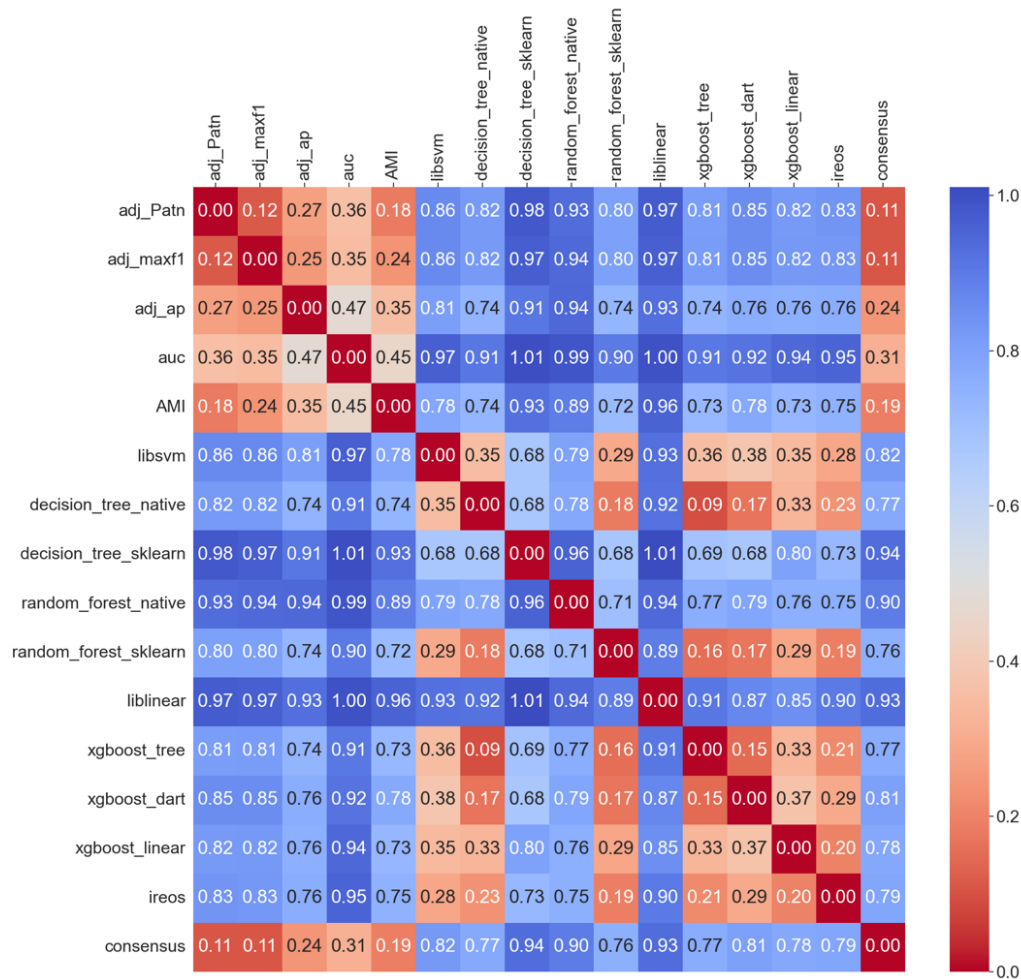


Figure 4.3: Distance matrix of different evaluation metrics that are executed on the seven outlier solutions for each basic2d dataset. Since FIREOS predictors return values on different scales, Z-Scores are used. Lower values in red represent more similar scorings and high values are more distinctive. It can be observed that most of the scorings performed by FIREOS predictors are closer to IREOS scorings than external validation. Obviously, external scorings show a high degree of similarity to the "consensus" metric. Lastly, LibLINEAR is shown to be the method that evaluates most differently among the compared predictors. Although many distances are colored in blue, there are no signs of strong deviations between any algorithm since the maximum number of standard deviations is 1.01.

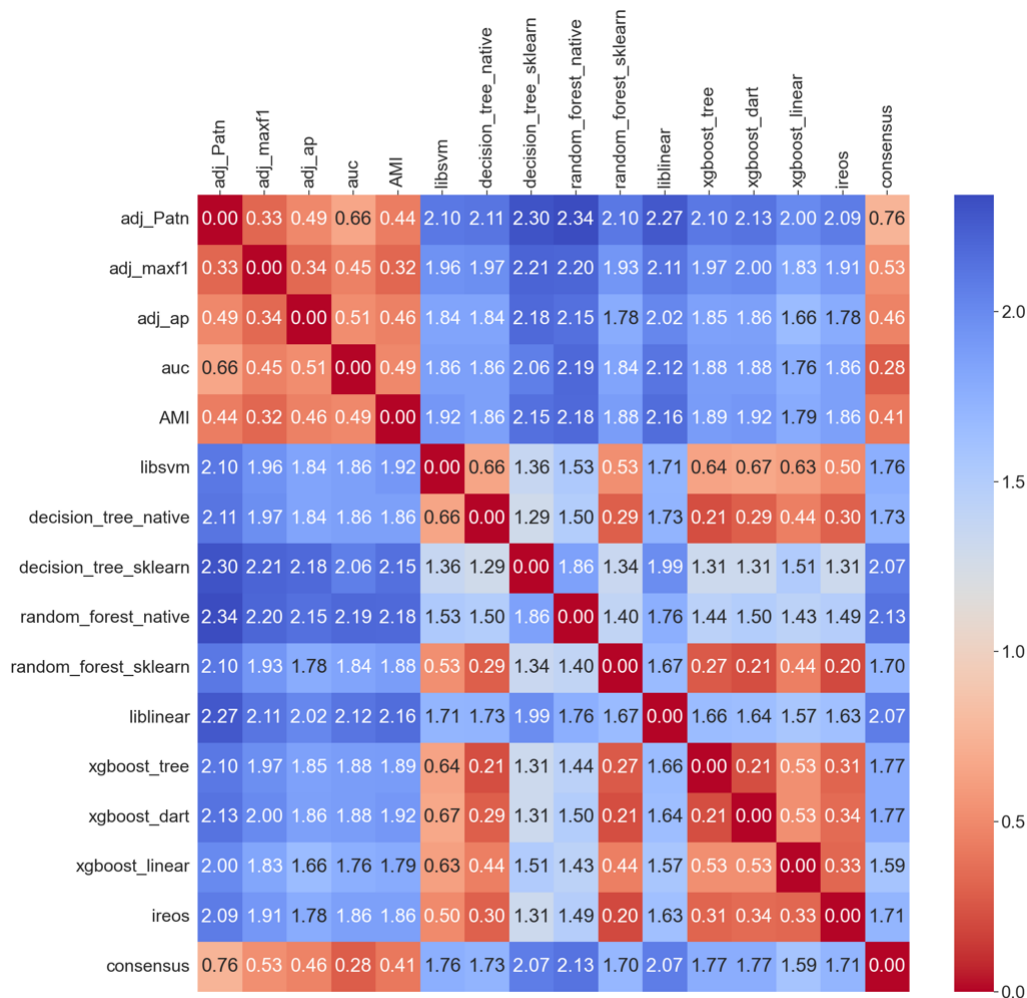


Figure 4.4: Distance matrix of different evaluation rankings executed on seven outlier solutions for each basic2d dataset. Lower values in red represent a lower distance between ranks, hence lower MAE . Most FIREOS predictors show similar distances to the consensus metric indicating common agreement of ranks. Only `decision_tree_sklearn`, `random_forest_native` and `liblinear` show strong deviations to consensus. When looking at other pairs they still show larger distances than any other FIREOS predictor leading to the assumption that one joint deviating prediction is the reason. Different from the distance matrix between Z -Scores, the MAE between predictors tends to be generally higher. However, as already stated before rank-similarity is especially sensitive to small misclassifications.

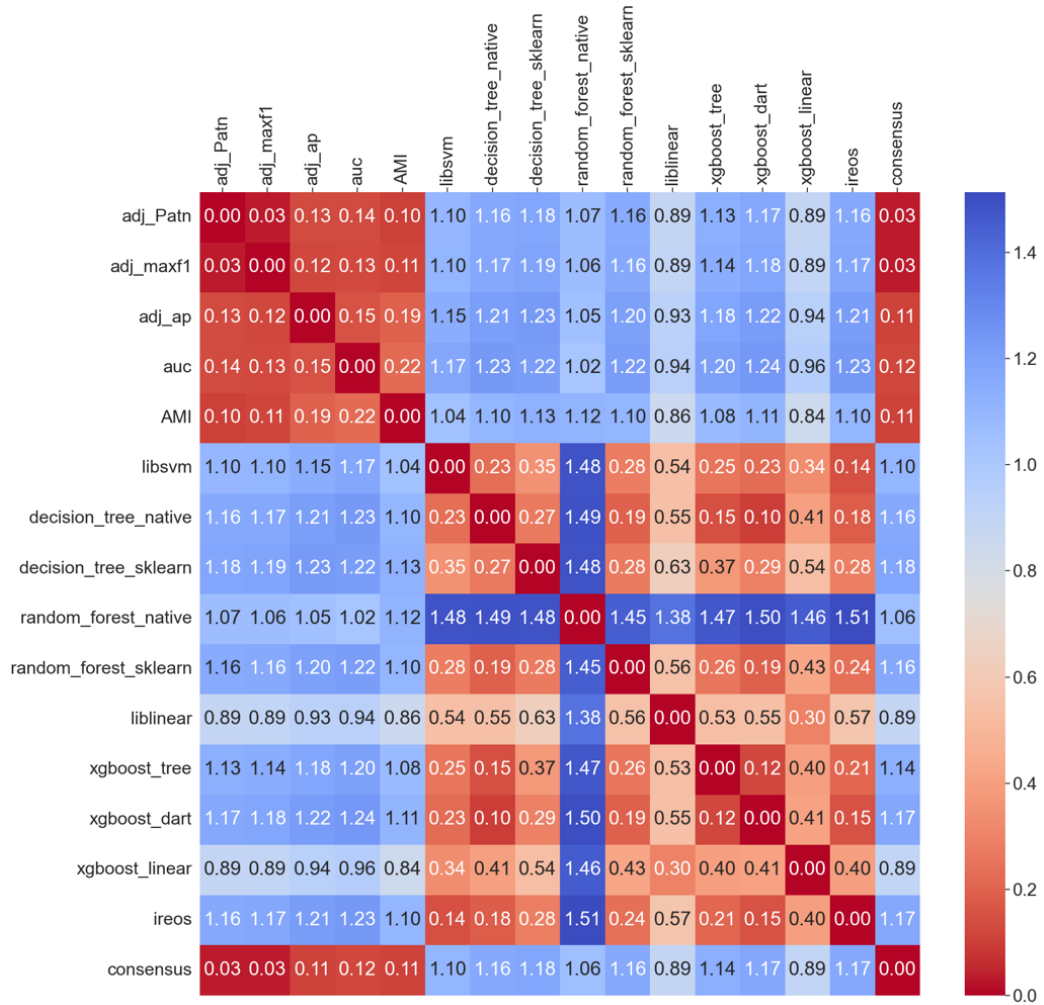


Figure 4.5: Z-Score distance matrix of different evaluation metrics executed on seven outlier solutions for each of the larger complex/high-noise/low-noise datasets. Lower values in red represent more similar scorings and high values are more distinctive. As shown most of the scorings performed by FIREOS predictors are closer to IREOS scorings than external validation. Although many distances are colored in blue, there are no signs of strong differences between any algorithm of FIREOS and the consensus metric since the maximum number of standard deviations is 1.17. However, random_forest_native clearly acts as a special case here since it shows large and consistent deviations from all other FIREOS predictors. The difference to any external validation method as well as consensus is lower but still aligned with the other FIREOS predictors which again shows the complex dynamics when comparing different outlieriness metrics.

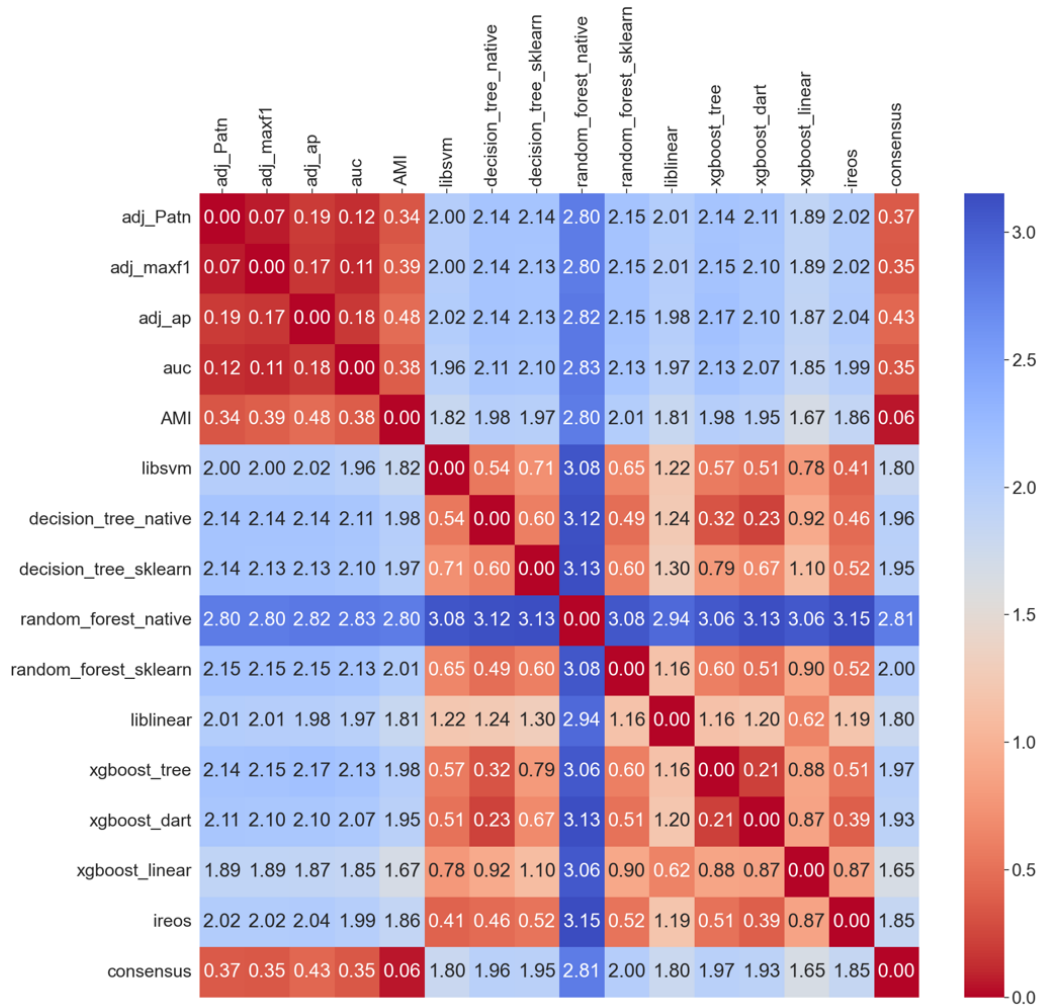


Figure 4.6: Distance matrix of different evaluation rankings executed on seven outlier solutions for each of the complex/high-noise/low-noise datasets. Lower values in red represent a lower distance between ranks, hence lower MAE . It can be observed that most of the scorings performed by FIREOS predictors are closer to IREOS scorings than external validation. The `random_forest_native` predictor provides rankings that are close to the opposite of any other method. This plot shows the extreme sensitivity of rank distances in comparison to Z -score distances which can be observed in [Figure 4.5](#).

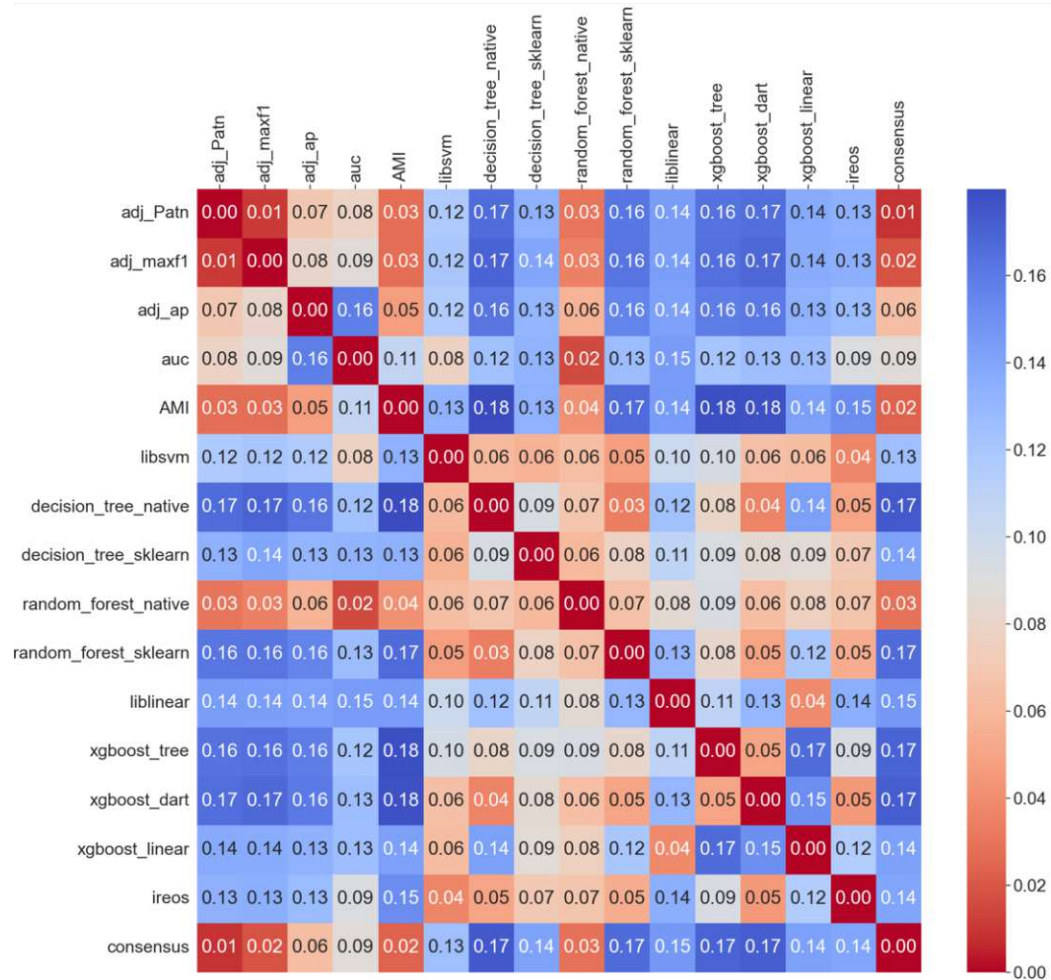


Figure 4.7: Standard deviations of Z-Score distance matrices between complex/high-noise/low-noise datasets. External measures tend to have a slightly higher variance toward internal validation. One explanation for that might be that external validation depends on predefined labels which are not known by internal indices. These labels, however, agree sometimes more and sometimes less with the topological properties of the data depending on the instance and therefore appear "less stable" in terms of variance.

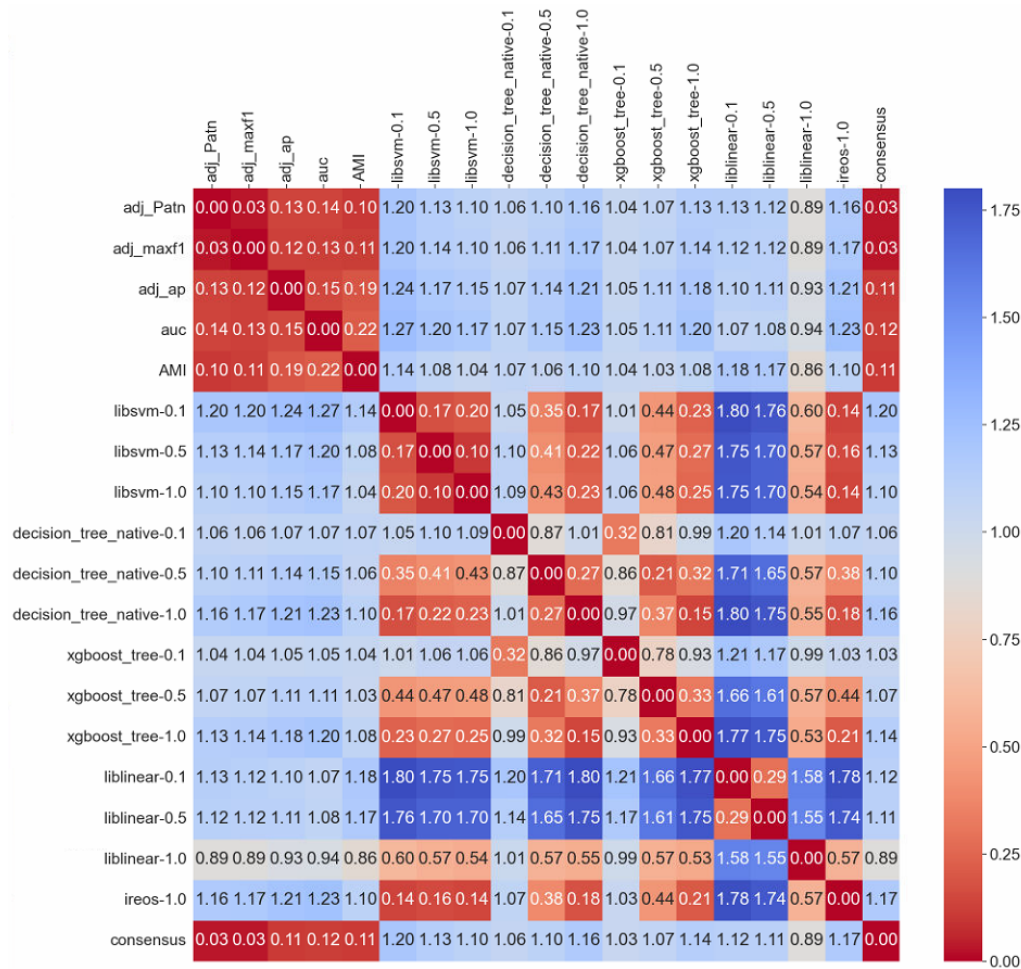


Figure 4.8: Distance matrix of different evaluation methods that are executed on seven outlier solutions for each of the complex/high-noise/low-noise datasets. This chart focuses on the quality of solutions when using sliding windows of different sizes. Suffixes of 0.1 mean 10%, 0.5 represents 50% and 1.0 means that all data samples are considered. Solutions from tree-based classifiers such as decision trees or XgBoost Tree show heavy degradation in quality when applied with sliding windows resulting in less agreement with IREOS as well as other FIREOS predictors for smaller window sizes. Results from linear predictors show a similar picture. Nonlinear support vector machines on the other hand seem to be not affected by this phenomenon as LibSVM results remain very stable over different settings.

4.2 Execution Time and Speedup

Besides the display of scores, each experiment also features the duration of the calculation. [Table 4.1](#) shows the detailed breakdown of runtimes for each dataset measured for each predictor. The abbreviations of the columns stand for:

- LS: LibSVM
- DT-N: Decision Tree Native
- DT-S: Decision Tree SkLearn
- RF-N: Random Forest Native
- RF-S: Random Forest SkLearn
- LL: LibLINEAR
- XTree: XgBoost Tree
- XDart: XgBoost DART
- XLin: XgBoost Linear
- Ireos: Original IREOS

The execution times of the original IREOS are on the farmost right column. [Table 4.2](#) focuses on the speedup potential of FIREOS predictors. [Figure 4.9](#) shows the runtime distribution across smaller problem instances of basic2d. [Figure 4.10](#) and [Table 4.2](#) show the potential of speedup for each FIREOS predictor on larger problems. In [Figure 4.11](#) and [Figure 4.12](#) the impact of sliding windows in or without the context of IREOS runtimes are visualized. Finally [Figure 4.13](#) shows the runtime behavior of the best performing FIREOS predictors LibSVM, Decision Tree Native and XgBoost Tree in contrast to the original IREOS implementation for different problem instance dimensions.

High-Noise Datasets										
#	LS	DT-N	DT-S	RF-N	RF-S	LL	XTree	XDart	XLin	Ireos
1	108.7	4.7	1.7	32.7	49.7	2.7	10.4	26.0	10.6	3098.3
2	224.8	8.2	2.1	72.2	66.1	4.0	24.1	41.9	15.3	3074.1
3	283.5	6.8	2.8	26.0	30.0	2.8	17.2	31.5	16.0	4902.6
4	140.5	4.8	1.0	64.4	79.5	3.6	8.2	16.2	5.4	1739.7
5	554.5	7.9	2.9	27.5	47.3	3.6	35.8	88.7	14.4	8678.9
6	103.2	5.1	2.0	28.0	36.6	2.6	11.9	21.6	7.9	2710.3
7	114.0	4.5	1.2	47.7	66.0	3.2	15.1	42.4	6.8	2539.6
8	795.4	7.5	3.2	24.8	37.2	3.9	42.9	105.7	16.5	9394.9
9	150.5	6.7	2.0	57.7	80.1	5.3	16.2	27.6	6.8	2816.8

#	LS	DT-N	DT-S	RF-N	RF-S	LL	XTree	XDart	XLin	Ireos
10	115.7	5.5	1.1	42.0	54.7	3.1	14.2	32.2	8.3	2609.8
11	122.2	6.1	2.3	40.1	56.1	3.5	16.9	38.2	7.9	3727.1
12	634.8	6.4	2.6	19.0	27.2	2.7	29.9	84.4	8.7	5767.3
13	258.3	7.9	2.4	45.1	65.4	4.6	25.7	53.9	14.2	6740.5
14	262.0	8.9	3.4	48.0	62.8	3.7	24.6	49.8	23.8	7628.7
15	166.5	6.0	2.1	46.8	58.3	3.8	18.0	37.8	10.6	3102.9
16	119.1	6.3	1.8	29.7	33.0	2.7	14.2	29.6	14.3	2562.5
17	117.3	5.4	2.0	41.9	62.7	3.3	18.4	24.1	5.8	3814.0
18	526.7	6.6	2.4	19.4	30.6	2.8	36.9	101.2	12.0	6763.0
19	139.0	6.1	1.1	60.9	86.5	5.6	11.2	36.3	16.7	2806.7
20	508.9	6.5	2.6	19.7	29.6	2.9	29.2	68.1	10.4	5919.1
Low-Noise Datasets										
1	99.5	4.2	0.8	31.3	36.6	2.3	11.3	19.2	7.1	1529.5
2	484.3	5.3	2.0	18.7	20.4	2.3	8.8	14.5	7.7	2380.0
3	90.9	4.4	1.0	25.0	27.5	2.0	10.9	19.5	9.8	1868.2
4	92.3	4.5	0.8	37.6	46.6	2.4	7.0	14.4	4.1	1539.1
5	446.7	5.4	2.2	23.5	29.3	2.1	10.1	13.9	8.1	3196.8
6	244.3	5.2	1.9	20.1	26.2	2.0	9.8	12.6	7.7	2337.7
7	236.7	5.0	1.8	23.7	30.8	2.2	11.6	20.6	12.0	2113.0
8	127.8	3.7	1.8	16.8	19.3	1.8	7.0	11.7	5.6	1626.2
9	95.4	3.7	1.3	26.7	41.4	2.0	6.8	13.6	12.3	1830.3
10	93.0	4.3	1.1	21.2	22.6	1.8	10.4	18.0	9.0	2152.4
11	308.2	4.3	2.0	19.3	28.7	2.0	7.6	12.8	4.0	1808.4
12	117.9	3.5	1.4	19.9	24.2	1.9	6.8	9.7	4.8	1720.2
13	105.6	5.0	0.8	46.9	60.3	2.6	13.0	22.5	12.4	1723.0
14	97.6	4.9	2.0	34.5	40.9	2.7	8.0	15.5	4.0	1714.4
15	87.8	4.0	1.5	26.2	36.1	1.9	9.3	20.8	8.6	1879.8
16	102.0	4.3	1.7	37.6	44.6	2.6	9.0	20.6	4.0	1740.3
17	90.1	3.9	1.2	21.2	24.3	2.1	7.8	13.8	5.7	1895.1
18	118.5	4.1	1.3	15.7	16.7	1.9	7.0	10.6	8.3	2140.6
19	105.1	5.0	0.9	48.0	63.9	3.7	9.5	17.3	4.1	1589.0
20	94.6	3.1	1.2	17.4	30.1	1.8	17.5	40.0	5.6	1569.5
Complex Datasets										
1	97.7	3.5	1.8	42.0	58.8	3.7	7.1	15.0	13.4	1630.0
2	87.9	3.6	0.6	36.4	53.1	2.0	6.3	12.0	3.6	1561.0
3	564.8	6.1	2.1	28.5	37.7	2.5	12.1	17.6	11.6	5839.7
4	484.2	5.6	2.0	23.1	27.5	2.3	10.9	16.4	8.7	5044.9
5	199.9	3.7	1.6	22.2	32.5	1.8	5.8	9.9	3.5	2211.4
6	112.8	3.9	1.9	18.8	23.4	1.9	8.5	16.6	10.3	2635.7
7	97.5	3.3	1.1	26.3	33.7	2.2	10.3	19.3	11.9	1870.2
8	348.6	5.4	2.0	22.2	22.2	4.3	8.7	12.5	7.0	3028.8

4. RESULTS AND DISCUSSION

#	LS	DT-N	DT-S	RF-N	RF-S	LL	XTree	XDart	XLin	Ireos
9	130.3	3.4	1.4	48.9	67.1	3.1	9.1	17.2	3.0	1554.8
10	221.2	5.0	1.8	25.0	31.9	2.0	11.2	17.5	12.9	4078.8
11	103.2	3.5	1.0	30.7	41.9	2.2	5.8	11.8	6.2	3265.7
12	98.0	3.4	1.2	35.6	49.3	2.4	6.8	13.4	3.2	1863.5
13	217.5	5.0	2.2	25.3	36.7	2.4	12.7	20.1	7.3	4596.4
14	272.1	4.4	1.5	19.4	24.0	2.0	7.8	11.1	6.3	3868.4
15	103.1	4.2	1.6	24.8	28.8	2.3	11.9	23.4	7.6	1900.8
16	167.2	4.5	1.6	19.0	26.4	1.9	9.2	15.7	6.7	3133.9
17	130.5	4.7	1.1	31.4	47.4	2.1	10.4	14.4	3.4	2370.6
18	255.7	6.0	2.1	25.7	35.2	2.3	11.1	15.5	10.0	6200.5
19	99.9	4.3	1.3	29.2	35.6	2.8	8.6	20.7	10.8	2821.5
20	294.8	5.0	2.3	18.9	20.5	2.1	8.5	12.9	6.0	3816.8

Table 4.1: Runtimes of different FIREOS predictors and IREOS. All results are measured in seconds, use the parallel implementation and are run on 32 threads.

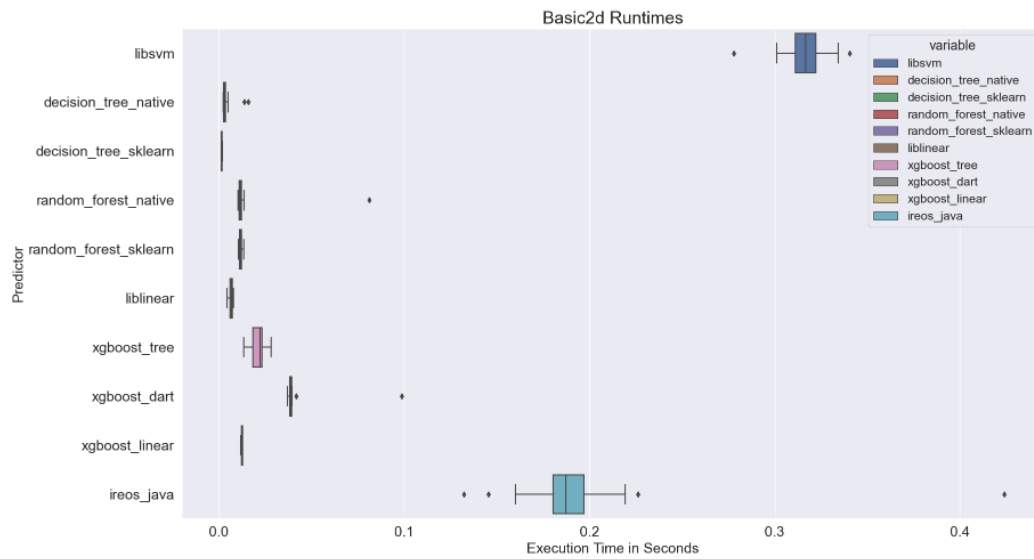


Figure 4.9: Average runtime in seconds of different predictors on basic2d datasets. All results are conducted by using the parallel implementation on 32 threads. As shown in the chart LibSVM is slower than IREOS for smaller problem instances like datasets of the basic2d category.

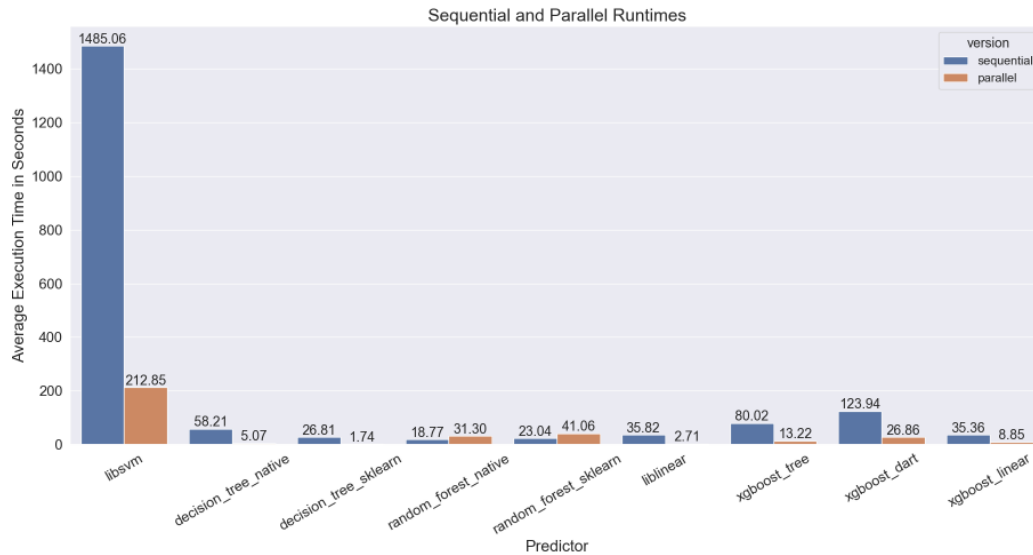


Figure 4.10: Average runtime in seconds between the sequential and parallel implementation of different FIREOS predictors on complex, high-noise and low-noise datasets.

Speedup of different Predictors			
Classifier	Sequential Runtime	Parallel Runtime	Speedup
LibSVM	1485.06	212.85	6.97
Decision Tree Native	58.21	5.07	11.48
Decision Tree SkLearn	26.81	1.74	15.45
Random Forest Native	18.77	31.30	0.6
Random Forest SkLearn	23.04	41.06	0.56
LibLINEAR	35.82	2.71	13.21
XgBoost Tree	80.02	13.22	6.05
XgBoost DART	123.94	26.86	4.61
XgBoost Linear	35.36	8.85	4.0

Table 4.2: Runtime in seconds of different predictors using sequential and parallel FIREOS as well as the corresponding speedup. Presented numbers are averages across all complex/high-noise/low-noise datasets and parallel results were conducted on 32 threads.

4. RESULTS AND DISCUSSION

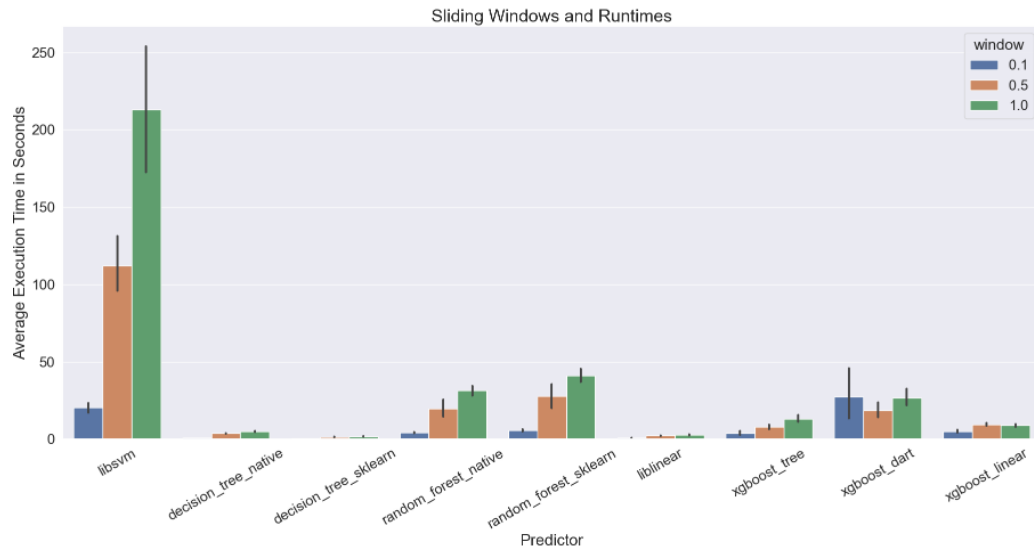


Figure 4.11: Average runtime in seconds of FIREOS classifiers and different window ratios. Values of 0.1 mean 10%, 0.5 represents 50% and 1.0 means that all data samples are considered.

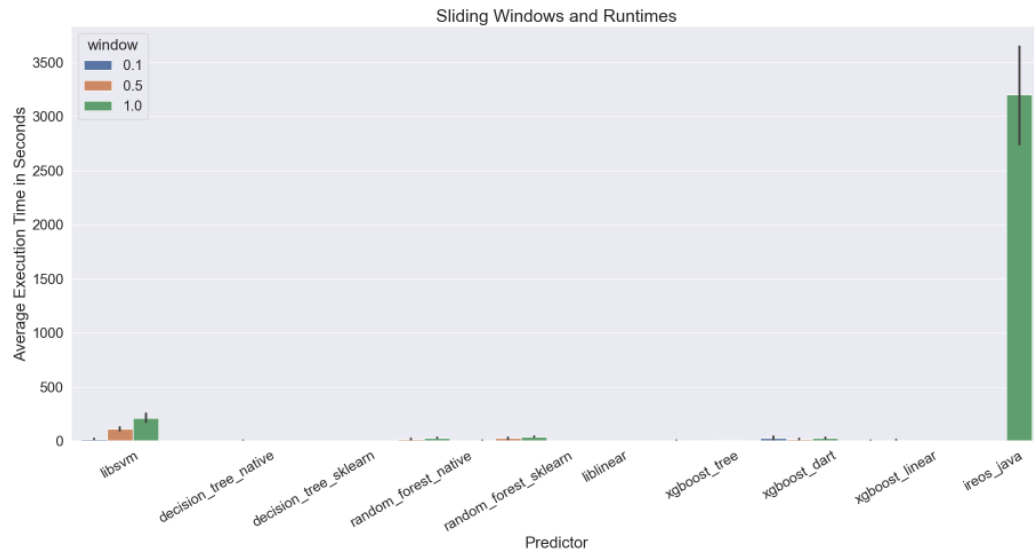


Figure 4.12: Same chart as [Figure 4.11](#) chart with IREOS runtime as a comparison. Since IREOS does not feature a mechanism for sliding windows, only one bar is displayed.

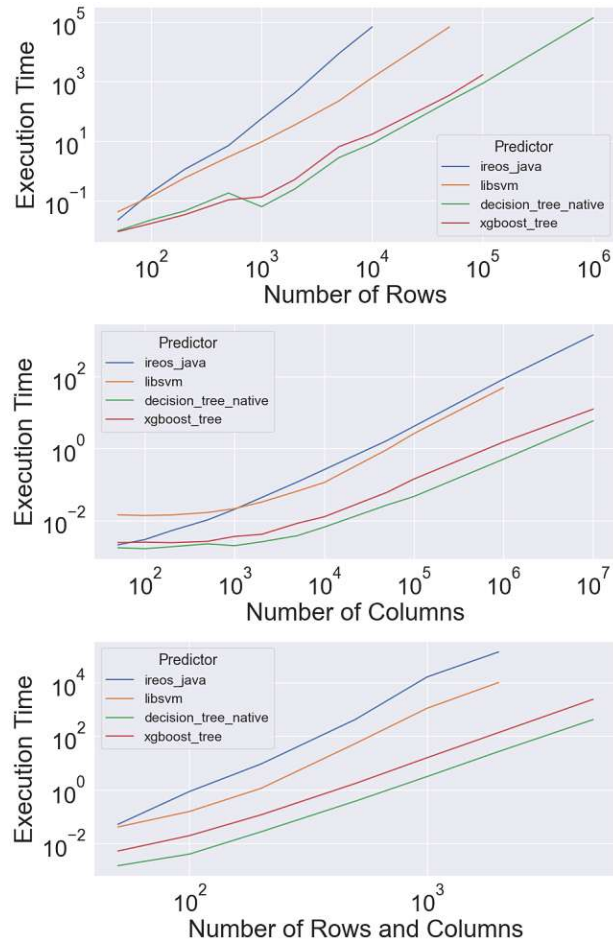


Figure 4.13: Runtime experiment on a subset of predictors for different key figures. Those KPIs are rows, columns/features and both. Each chart shows the runtime curves measured in seconds of three FIREOS predictors (LibSVM, Decision Tree Native and XgBoost Tree) as well as the original IREOS over some increasing key figure. For example, the top plot shows the course of execution time when exponentially increasing the total number of rows of a random dataset. The steeper the curve the more computationally expensive the corresponding predictor. All datasets used for this experiment are randomly generated matrices of normally distributed variables. The top plot shows the course of runtime for a variable number of rows (x-axis) and a fixed number of features which is 10. The middle chart shows the runtime behavior for 10 rows but a variable number of features. Finally, the bottom plot shows the course of execution time for both variable rows and features. The x-axis in the last chart represents both rows and features which means that all datasets are quadratic matrices. As shown all three FIREOS predictors are faster than the original IREOS for either KPI except libsvm for very small problems. This chart demonstrates that IREOS is computationally more complex than FIREOS.

4.3 Discussion

Looking at the results, both scores and ranks of the FIREOS evaluation show a high similarity to the existing implementation for most of the predictors. As already forecasted by the authors of the original paper, especially nonlinear classifiers such as LibSVM, XgBoost DART, XgBoost Tree and Decision Tree Native are similar to IREOS. Most linear predictors such as LibLINEAR and XgBoost Linear on the other hand show remarkable deviations from all existing measures. A possible explanation for that might be the frequent occurrence of instances that are not linear separable resulting in unexpected behavior during the classification phase. A major outlier when it comes to the results of the evaluation is the Random Forest Native classifier, which does not apply any tree pruning. The resulting problem of not limiting the maximum depth of a tree is already described in Section [3.3.3](#). When properly pruned, random forests still provide decent results as Random Forest SkLearn shows.

During our experiments, we observed certain differences between FIREOS, IREOS and external scorings in all matrices:

- External validation scorings show smaller distances to each other, therefore also to consensus, than against FIREOS or IREOS.
- Scores from internal evaluation such as IREOS and most FIREOS scores are more similar to each other than external validation methods.

These observations lead to the impression that internal and external validation are entirely different perspectives on outlierness and do not fit together. However, each series of experiments points out that most of the observed methods provide consistent results and clearly indicate relationships between them. Furthermore, results show that both internal and external evaluation methods share a baseline agreement when it comes to scores and we are not able to prove a statistically significant deviation or even strong divergence between them. Although some of the distances of the rank matrices point to large deviations, we must be careful due to the fact that they are very sensitive to minor changes and difficult to interpret. The experiments show behavior that leads to the assumption that FIREOS is more similar to IREOS than external validation. However, despite all those differences we still do not know how much in terms of real accuracy or even real distance.

When it comes to runtimes, FIREOS is more lightweight than its predecessor. As already smaller datasets like basic2d show, most of the FIREOS predictors evaluate the same instance in a fraction of the time. Furthermore, IREOS runs over 50 minutes on average to evaluate an instance coming from complex, high-noise or low-noise. In comparison, the fastest FIREOS classifier "Decision Tree SkLearn" never exceeds 4 seconds on each instance, which represents a speedup of 750. Decision trees are not the only quick alternative as most of the evaluations do not take longer than one minute. However, the

slowest alternative by far is the support vector machine (LibSVM) which is still over 10 times faster than the original KLR. There are several reasons for that behavior:

1. First, there is of course the base costs of each estimator. As already mentioned in Section 3 the algorithmic complexity of KLR is $O(n^3)$ which is worse to support vector machines $O(n^2m)$ which is again worse than decision trees with $O(nd \cdot \log(n))$.
2. Adaptive quadrature is disabled on all predictors except LibSVM and of course IREOS. This numerical integration method is computationally very expensive as it triggers the re-execution of training separabilities until an error criterion is met. FIREOS has a general fail-save that prevents the algorithm from getting stuck in an endless loop if this criterion is never met. However, in the worst case, 14 more predictors still need to be trained for each sample resulting in a multitude of internal classifiers. Numerical integration is a remainder of the original IREOS and is designed to estimate the separability function of each γ instead of randomly guessing it. This γ parameter, however, is only required for predictors utilizing the radial basis kernel. As tree-based and linear predictors do not need kernel functions, this feature is disabled for all classifiers except LibSVM which is the reason why they are significantly faster.
3. The FIREOS module utilizes multithreading in its parallel implementation. However, the largest chunk of the speedup is dependent on the parallelization capabilities of the underlying library. As Table 4.1 shows, the highest speedup between single and multithreaded implementation is achieved by decision trees and LibLINEAR. When, for instance, compared to LibSVM, those predictors accelerate by twice as much.

Another major accelerator of FIREOS is the application of sliding windows. In most cases, the "window_ratio" parameter does not only correlate with the input space rather than controlling the runtime. However, as results show, this loss of information comes with a price. Results from instances evaluated with only a subset of samples show noticeable deviations in comparison to the whole set. This variance diverges even further the smaller the window size and therefore fewer instances are considered. In our experiments, only nonlinear support vector machines were not prone to this behavior.

Looking at the speedup table, we can observe major differences when it comes to the scalability of different estimators. As all of our experiments are executed on a 16 core CPU, we would expect a significant reduction in runtime. However, measures show that especially the performance of random forests suffers when executed in parallel, being over 40% slower than the sequential algorithm. All three XgBoost implementations as well as LibSVM show mediocre speedup where LibLINEAR and decision trees scale the best with the latter almost reaching the ideal speedup. These insights are particularly beneficial when considering a time-critical predictor for systems with many CPU cores.

Although decision trees already being the fastest predictor, this trend will be further reinforced as the core count increases.

In conclusion, experiments show that the recommended options for FIREOS are:

1. XgBoost DART
2. Decision Tree Native
3. LibSVM

As shown in the experiments, XgBoost DART provides the results closest to the original IREOS with still decent speedup. In cases where the most lightweight predictor is desired, we recommend the Decision Tree Native, which offers results close to XgBoost and near-optimal multicore capabilities. Finally, we must not forget to mention the LibSVM predictor that provides the second-best but most stable results even with smaller sliding windows which is especially advantageous for datasets with lots of instances.

CHAPTER 5

Conclusions

In this chapter, we summarize our findings and insights from the previous sections to draw a conclusion and answer the research questions. We proposed an alternative implementation to the existing IREOS algorithm by modifying several components of the algorithm to make it more lightweight.

We also introduced 12 additional classifiers following different principles with different strengths and weaknesses. Furthermore, our novel implementation features a sampling option as well as additional parameters to control the overall complexity of the algorithm. We included interfaces for preliminary tasks such as data normalization by Kriegel's outlier scoring unification framework in our implementation and used fixed seeds to ensure reproducibility. In addition to a sequential as well as parallel FIREOS module, we also adjusted the workflow of the existing Java implementation.

We used 80 synthetic datasets with different characteristics for training each algorithm and evaluated 7 different outlier detection solutions coming from algorithms namely ABOD, HBOS, KNN, LOF, IFOREST, OCSVM and SDO. In addition to the resulting seven scorings evaluated each by IREOS and FIREOS, the exact same solutions are assessed by Adjusted P@n, Adjusted Max F1 Score, Adjusted Average Precision, Area Under the ROC Curve and Adjusted Mutual Information which are popular algorithms that use external validation.

While retaining the original intuition and structure of the algorithm, the proposed software shows a considerable performance increase when compared to the original implementation. We showed in our experiments that most predictors of FIREOS indicate a high similarity towards the base implementation. Especially tree-based approaches as well as nonlinear support vector machines have a large amount of agreement with IREOS in our experiments. We further showed in our study that outlier detection algorithms evaluated by FIREOS might differ from other external validation methods but very similarly to how IREOS does it.

When it comes to performance comparison, our runtime benchmark showed significantly faster results for most of the FIREOS predictors. In our experiments decision trees and LibLINEAR are the fastest alternatives to the KLR of the original algorithm. Especially the runtime analysis plot in the previous chapter shows the impact of those less complex estimators when increasing the number of samples and/or dimensions.

We then showed that the use of sliding windows further decreases the complexity and leads to significant performance gains. However, we also noted that this loss of input data decreases the overall quality and stability of the results. Finally, we provided some recommendations about the ideal FIREOS predictor for various scenarios.

Considering this gained knowledge, we are finally able to make statements about our goals and answer the research questions formulated in Section [1.3](#).

The first goal that is aimed to be achieved is to reduce the computational complexity of the algorithm and reach a faster implementation of the original IREOS (G1)

We reached that goal by introducing FIREOS, a modified version of IREOS that uses computationally less complex predictors than its predecessor to evaluate outlier solutions. Furthermore, we proposed and integrated a sampling method into FIREOS that enables the option to further reduce computational costs by reducing the number of samples during the separability calculation.

As shown, FIREOS finishes each evaluation of the 80 datasets in significantly less time than the original implementation.

Which components of IREOS can be replaced by equivalent methods in order to obtain faster performances and minimally affect accuracy? (RQ1)

In this thesis, we performed several adjustments to the original algorithm to make it faster:

1. **Replacing the internal predictor:** Discarding the KLR predictor as the central element of the algorithm in order to use classifiers close to linear complexity is the most crucial step as it not only makes the implementation faster when we test its performance empirically but also reduces the real algorithmic complexity. The results coming from most FIREOS predictors are quite similar to those coming from IREOS indicating strong accuracy.
2. **Leave out numerical integration:** The omission of the numerical integration mechanism for most predictors leads to faster results and less allocated memory due to recursion.
3. **Sliding windows:** The application of sliding windows in FIREOS is a double-edged sword. On the one hand, we were able to observe significant performance gains for smaller subsets when it comes to speed, however, the accuracy of the algorithm is severely affected when the sliding window gets too small. Furthermore, this

accuracy degradation is very dependent on the underlying instance and cannot be reliably predicted beforehand.

The second goal is to evaluate to which degree IREOS and FIREOS are consistent with existing external validation metrics. (G2)

In this thesis, we compared evaluations by 9 predictors from FIREOS and the original IREOS against 5 external validation metrics by contrasting the evaluated solutions of seven outlier detection algorithms of overall 80 problem instances. We presented in our results that neither FIREOS nor the original IREOS algorithm shows strong deviations from any external validation measure. Our experiments concluded that scores from FIREOS and the external consensus metric have an average score difference of 0.76 - 1.18 standard deviations. IREOS for comparison provides a Z -score difference of 0.98.

The reasons for the discrepancy between internal validation (IREOS/FIREOS) and external validation are to be studied carefully in future work. We hypothesize that the main reasons might have to do with the mechanisms to generate and label the synthetic data used in combination with the different dynamics of the tested outlier detection algorithms when establishing outlierness. External validation matches better the binary set of the synthetic ground truth (0/1), whereas the principles behind internal validation are more fine and consistent with non-crisp (gradual, gradient, variable) assessments of outlierness. This causes some biases on both sides when comparing the performances of algorithms.

How does IREOS/FIREOS align with external evaluation metrics and, consequently, which type of outlierness definition is favored by IREOS based on empirical evidence? (RQ2)

As already stated in the answer of (G2), the interpretation of distance values regarding outlierness is delicate. In our experiments, we were able to present that for those seven outlier detection algorithms considered most FIREOS predictors are very similar to IREOS but both IREOS and FIREOS show some deviation to external validation. However, those differences are still in ranges to be not considered significantly diverging from FIREOS.

Tree-based predictors specifically show high similarities with the original implementation with Random Forest SkLearn and XgBoost DART being the most alike. Non-linearity is an important property of a versatile classifier since linear methods such as LibLINEAR are less stable.

In summary, our empiric study of 80 datasets shows that tree-based methods are a viable and lightweight replacement for the KLR of the original algorithm and also similar in terms of quality to the nonlinear support vector machine, which is also recommended by the authors of IREOS.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendix

A.1 FIREOS Results

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
basic2d_1										
sdo	0.008	0.601	0.049	0.175	0.077	0.93	0.966	0.995	1.000	0.826
abod	0.007	0.539	0.060	0.141	0.020	0.93	0.934	0.949	0.998	0.768
hbos	0.008	0.551	0.062	0.147	0.028	0.37	0.550	0.735	0.938	0.351
iforest	0.007	0.563	0.067	0.156	0.033	0.86	0.898	0.934	0.997	0.700
knn	0.008	0.600	0.049	0.174	0.075	0.93	0.966	0.995	1.000	0.826
lof	0.008	0.600	0.036	0.161	0.053	0.65	0.790	0.922	0.962	0.522
ocsvm	0.008	0.563	0.051	0.154	0.043	0.65	0.790	0.922	0.955	0.414
basic2d_2										
sdo	0.008	0.615	0.061	0.192	0.079	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.546	0.029	0.145	0.023	0.86	0.934	0.975	0.999	0.639
hbos	0.007	0.543	0.037	0.145	0.024	0.65	0.674	0.786	0.949	0.445
iforest	0.007	0.561	0.036	0.155	0.030	0.93	0.966	0.979	0.999	0.826
knn	0.008	0.610	0.058	0.189	0.073	1.00	1.000	1.000	1.000	1.000
lof	0.009	0.630	0.076	0.202	0.101	1.00	1.000	1.000	1.000	1.000
ocsvm	0.008	0.576	0.046	0.164	0.045	0.79	0.850	0.951	0.957	0.593
basic2d_3										
sdo	0.007	0.552	0.046	0.152	0.037	0.93	0.966	0.990	1.000	0.826
abod	0.007	0.539	0.051	0.141	0.014	0.72	0.841	0.867	0.993	0.388
hbos	0.007	0.532	0.036	0.137	0.014	0.09	0.163	0.128	0.729	0.010
iforest	0.007	0.555	0.044	0.151	0.019	0.58	0.596	0.809	0.978	0.339

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
knn	0.007	0.552	0.044	0.151	0.032	0.93	0.966	0.990	1.000	0.758
lof	0.007	0.550	0.055	0.151	0.046	0.86	0.860	0.964	0.996	0.700
ocsvm	0.007	0.550	0.045	0.148	0.026	0.58	0.737	0.888	0.813	0.339
basic2d_4										
sdo	0.008	0.599	0.058	0.177	0.082	0.93	0.930	0.990	0.997	0.826
abod	0.007	0.549	0.031	0.146	0.027	0.79	0.850	0.788	0.993	0.593
hbos	0.007	0.554	0.034	0.150	0.032	0.65	0.745	0.737	0.942	0.414
iforest	0.008	0.570	0.037	0.161	0.043	0.86	0.860	0.945	0.996	0.700
knn	0.008	0.607	0.061	0.181	0.088	0.93	0.930	0.990	0.998	0.826
lof	0.008	0.625	0.054	0.207	0.100	0.72	0.747	0.850	0.957	0.499
ocsvm	0.008	0.570	0.045	0.159	0.052	0.93	0.930	0.990	0.935	0.826
basic2d_5										
sdo	0.007	0.587	0.049	0.170	0.053	1.00	1.000	1.000	1.000	1.000
abod	0.006	0.537	0.026	0.139	0.015	0.79	0.819	0.924	0.962	0.555
hbos	0.007	0.557	0.033	0.152	0.028	0.58	0.650	0.621	0.979	0.393
iforest	0.007	0.570	0.035	0.158	0.034	1.00	1.000	1.000	1.000	1.000
knn	0.007	0.610	0.051	0.177	0.074	1.00	1.000	1.000	1.000	1.000
lof	0.007	0.614	0.054	0.182	0.076	1.00	1.000	1.000	1.000	1.000
ocsvm	0.007	0.588	0.047	0.167	0.056	1.00	1.000	1.000	1.000	1.000
basic2d_6										
sdo	0.007	0.615	0.031	0.182	0.062	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.538	0.046	0.140	0.015	0.93	0.930	0.939	0.998	0.826
hbos	0.008	0.550	0.031	0.147	0.019	0.58	0.737	0.888	0.876	0.469
iforest	0.007	0.568	0.042	0.158	0.027	1.00	1.000	1.000	1.000	1.000
knn	0.007	0.615	0.030	0.180	0.061	1.00	1.000	1.000	1.000	1.000
lof	0.008	0.650	0.028	0.209	0.089	1.00	1.000	1.000	1.000	1.000
ocsvm	0.007	0.576	0.043	0.161	0.037	1.00	1.000	1.000	1.000	1.000
basic2d_7										
sdo	0.009	0.650	0.032	0.215	0.089	0.93	0.966	0.995	1.000	0.826
abod	0.007	0.540	0.040	0.142	0.016	0.93	0.930	0.985	0.998	0.639
hbos	0.007	0.573	0.041	0.161	0.031	0.65	0.695	0.866	0.980	0.414
iforest	0.008	0.590	0.038	0.174	0.041	0.93	0.964	0.995	0.999	0.826
knn	0.009	0.638	0.034	0.209	0.080	0.93	0.964	0.995	1.000	0.911
lof	0.009	0.622	0.034	0.199	0.070	1.00	1.000	1.000	1.000	1.000
ocsvm	0.008	0.589	0.038	0.173	0.039	0.93	0.966	0.995	1.000	0.826
basic2d_8										
sdo	0.008	0.591	0.035	0.177	0.075	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.538	0.038	0.141	0.016	1.00	1.000	1.000	1.000	0.915
hbos	0.007	0.541	0.034	0.144	0.019	0.51	0.560	0.678	0.759	0.182

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
iforest	0.007	0.561	0.036	0.155	0.028	0.93	0.964	0.995	0.999	0.826
knn	0.008	0.584	0.035	0.172	0.072	1.00	1.000	1.000	1.000	1.000
lof	0.008	0.598	0.033	0.181	0.074	1.00	1.000	1.000	1.000	1.000
ocsvm	0.007	0.566	0.036	0.160	0.047	0.93	0.964	0.995	0.996	0.826
basic2d_9										
sdo	0.009	0.648	0.057	0.215	0.120	0.93	0.966	0.995	1.000	0.826
abod	0.007	0.542	0.024	0.143	0.021	0.86	0.905	0.886	0.996	0.654
hbos	0.007	0.574	0.026	0.163	0.040	0.58	0.714	0.820	0.982	0.339
iforest	0.007	0.581	0.029	0.167	0.045	0.86	0.905	0.862	0.996	0.700
knn	0.009	0.629	0.047	0.200	0.095	0.93	0.966	0.995	1.000	0.826
lof	0.009	0.644	0.058	0.211	0.101	1.00	1.000	1.000	1.000	1.000
ocsvm	0.008	0.604	0.036	0.183	0.061	0.93	0.934	0.990	0.999	0.826
basic2d_10										
sdo	0.008	0.629	0.051	0.199	0.065	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.541	0.036	0.143	0.016	0.86	0.891	0.945	0.997	0.700
hbos	0.007	0.575	0.033	0.165	0.028	0.51	0.650	0.827	0.963	0.270
iforest	0.007	0.573	0.038	0.163	0.029	0.93	0.966	0.995	1.000	0.826
knn	0.008	0.624	0.047	0.190	0.069	1.00	1.000	1.000	1.000	1.000
lof	0.009	0.641	0.060	0.200	0.074	0.93	0.930	0.985	0.998	0.826
ocsvm	0.007	0.593	0.039	0.175	0.043	0.79	0.883	0.971	0.801	0.593
basic2d_11										
sdo	0.008	0.604	0.047	0.188	0.078	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.547	0.031	0.146	0.025	0.86	0.891	0.909	0.995	0.695
hbos	0.007	0.555	0.040	0.150	0.031	0.51	0.560	0.713	0.910	0.270
iforest	0.008	0.572	0.041	0.162	0.041	0.86	0.934	0.975	0.999	0.700
knn	0.008	0.612	0.048	0.192	0.083	1.00	1.000	1.000	1.000	1.000
lof	0.009	0.602	0.075	0.196	0.100	0.65	0.677	0.874	0.951	0.445
ocsvm	0.008	0.570	0.043	0.161	0.045	0.93	0.930	0.979	0.999	0.826
basic2d_12										
sdo	0.009	0.678	0.041	0.231	0.068	0.93	0.966	0.990	1.000	0.826
abod	0.007	0.546	0.040	0.146	0.014	0.93	0.966	0.973	0.999	0.758
hbos	0.007	0.577	0.046	0.164	0.022	0.79	0.819	0.953	0.977	0.639
iforest	0.007	0.581	0.042	0.167	0.024	1.00	1.000	1.000	1.000	1.000
knn	0.008	0.656	0.043	0.215	0.056	1.00	1.000	1.000	1.000	1.000
lof	0.008	0.694	0.038	0.234	0.066	1.00	1.000	1.000	1.000	1.000
ocsvm	0.008	0.601	0.041	0.180	0.035	0.86	0.925	0.986	0.908	0.700
basic2d3										
sdo	0.008	0.604	0.043	0.194	0.076	0.93	0.966	0.990	1.000	0.826
abod	0.007	0.543	0.031	0.145	0.018	0.72	0.777	0.627	0.984	0.439

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
hbos	0.008	0.557	0.028	0.156	0.026	0.37	0.506	0.713	0.926	0.198
iforest	0.007	0.571	0.032	0.163	0.030	0.86	0.860	0.726	0.992	0.700
knn	0.008	0.600	0.042	0.187	0.065	0.93	0.966	0.990	1.000	0.826
lof	0.008	0.609	0.061	0.192	0.088	0.93	0.966	0.995	1.000	0.826
ocsvm	0.008	0.588	0.044	0.177	0.054	0.72	0.790	0.932	0.984	0.499
basic2d_14										
sdo	0.009	0.638	0.051	0.205	0.126	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.540	0.029	0.141	0.023	0.79	0.905	0.797	0.994	0.536
hbos	0.008	0.573	0.033	0.159	0.044	0.65	0.758	0.822	0.977	0.414
iforest	0.008	0.586	0.034	0.168	0.056	0.79	0.876	0.876	0.996	0.593
knn	0.009	0.628	0.048	0.197	0.112	1.00	1.000	1.000	1.000	1.000
lof	0.009	0.610	0.045	0.187	0.094	0.93	0.966	0.995	1.000	0.826
ocsvm	0.008	0.595	0.036	0.173	0.064	1.00	1.000	1.000	1.000	1.000
basic2d_15										
sdo	0.008	0.548	0.053	0.145	0.052	0.93	0.966	0.995	1.000	0.826
abod	0.007	0.537	0.058	0.141	0.016	0.93	0.966	0.979	0.999	0.758
hbos	0.007	0.530	0.059	0.136	0.019	0.44	0.758	0.795	0.984	0.244
iforest	0.007	0.550	0.061	0.148	0.020	0.86	0.925	0.986	0.997	0.700
knn	0.008	0.550	0.054	0.146	0.045	0.93	0.966	0.995	1.000	0.826
lof	0.007	0.569	0.052	0.158	0.039	0.51	0.680	0.846	0.963	0.547
ocsvm	0.007	0.545	0.059	0.146	0.043	0.51	0.680	0.846	0.888	0.270
basic2d_16										
sdo	0.015	0.709	0.035	0.253	0.121	1.00	1.000	1.000	1.000	1.000
abod	0.009	0.548	0.039	0.147	0.018	1.00	1.000	1.000	1.000	0.915
hbos	0.010	0.611	0.035	0.185	0.055	0.79	0.883	0.971	0.995	0.779
iforest	0.010	0.595	0.040	0.177	0.041	1.00	1.000	1.000	1.000	1.000
knn	0.016	0.708	0.036	0.254	0.116	1.00	1.000	1.000	1.000	1.000
lof	0.017	0.728	0.034	0.269	0.128	1.00	1.000	1.000	1.000	1.000
ocsvm	0.012	0.628	0.038	0.203	0.060	1.00	1.000	1.000	1.000	1.000
basic2d_17										
sdo	0.007	0.601	0.063	0.181	0.057	0.93	0.966	0.990	1.000	0.826
abod	0.007	0.546	0.037	0.145	0.016	0.65	0.753	0.719	0.964	0.266
hbos	0.007	0.551	0.047	0.146	0.019	0.23	0.392	0.296	0.887	0.065
iforest	0.007	0.570	0.048	0.161	0.027	0.79	0.831	0.948	0.996	0.593
knn	0.007	0.600	0.062	0.178	0.051	0.93	0.966	0.995	1.000	0.826
lof	0.007	0.599	0.055	0.181	0.054	0.93	0.966	0.995	1.000	0.826
ocsvm	0.007	0.570	0.048	0.161	0.036	0.86	0.891	0.980	0.927	0.700
basic2d_18										
sdo	0.009	0.588	0.046	0.171	0.083	1.00	1.000	1.000	1.000	1.000

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
abod	0.007	0.538	0.043	0.140	0.019	0.72	0.730	0.827	0.985	0.439
hbos	0.007	0.555	0.051	0.148	0.027	0.58	0.706	0.795	0.958	0.522
iforest	0.007	0.568	0.052	0.158	0.036	0.65	0.758	0.915	0.991	0.414
knn	0.009	0.596	0.048	0.171	0.082	0.93	0.966	0.995	1.000	0.911
lof	0.008	0.640	0.044	0.188	0.113	0.65	0.706	0.885	0.872	0.364
ocsvm	0.007	0.574	0.061	0.159	0.049	0.58	0.737	0.888	0.965	0.339

basic2d_19

sdo	0.008	0.604	0.041	0.176	0.075	0.86	0.934	0.975	0.999	0.700
abod	0.007	0.540	0.030	0.141	0.021	0.72	0.737	0.674	0.986	0.414
hbos	0.007	0.551	0.031	0.146	0.028	0.44	0.440	0.529	0.861	0.209
iforest	0.008	0.569	0.035	0.157	0.040	0.72	0.803	0.718	0.990	0.499
knn	0.008	0.605	0.040	0.178	0.079	0.86	0.934	0.939	0.997	0.639
lof	0.009	0.627	0.039	0.188	0.107	0.79	0.876	0.861	0.994	0.593
ocsvm	0.008	0.576	0.038	0.161	0.050	0.79	0.841	0.953	0.989	0.593

basic2d_20

sdo	0.009	0.566	0.047	0.163	0.088	1.00	1.000	1.000	1.000	1.000
abod	0.007	0.538	0.065	0.141	0.016	1.00	1.000	1.000	1.000	1.000
hbos	0.007	0.542	0.085	0.145	0.022	0.72	0.745	0.824	0.953	0.499
iforest	0.007	0.560	0.056	0.156	0.028	1.00	1.000	1.000	1.000	1.000
knn	0.009	0.564	0.047	0.160	0.075	1.00	1.000	1.000	1.000	1.000
lof	0.009	0.603	0.018	0.186	0.105	1.00	1.000	1.000	1.000	1.000
ocsvm	0.009	0.571	0.042	0.165	0.095	1.00	1.000	1.000	1.000	1.000

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
complex_1										
sdo	0.001	0.532	0.008	0.136	0.155	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.508	0.004	0.124	0.040	0.995	0.997	1.000	1.000	0.981
hbos	0.001	0.512	0.006	0.125	0.057	0.885	0.938	0.994	0.983	0.854
iforest	0.001	0.516	0.006	0.127	0.073	0.986	0.992	1.000	1.000	0.952
knn	0.001	0.533	0.009	0.135	0.156	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.547	0.002	0.139	0.120	0.322	0.445	0.693	0.746	0.118
ocsvm	0.001	0.510	0.004	0.125	0.053	0.977	0.986	1.000	0.984	0.926

complex_2

sdo	0.552	0.555	0.003	0.148	0.812	0.998	0.998	1.000	1.000	0.992
abod	0.153	0.517	0.001	0.128	0.254	0.987	0.990	0.997	1.000	0.949
hbos	0.192	0.524	0.002	0.130	0.319	0.797	0.834	0.958	0.977	0.570
iforest	0.263	0.530	0.002	0.134	0.412	0.984	0.987	0.999	1.000	0.941
knn	0.543	0.554	0.003	0.147	0.800	1.000	1.000	1.000	1.000	1.000

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
lof	0.523	0.548	0.002	0.145	0.770	0.528	0.655	0.874	0.851	0.273
ocsvm	0.253	0.527	0.002	0.134	0.378	0.972	0.973	0.999	0.998	0.909
complex_3										
sdo	0.170	0.526	0.002	0.131	0.504	0.702	0.767	0.591	0.984	0.470
abod	0.059	0.510	0.002	0.123	0.251	0.608	0.723	0.477	0.978	0.376
hbos	0.102	0.517	0.002	0.127	0.363	0.362	0.430	0.276	0.916	0.148
iforest	0.124	0.520	0.002	0.128	0.417	0.469	0.566	0.353	0.956	0.231
knn	0.176	0.527	0.002	0.132	0.515	0.721	0.801	0.544	0.986	0.446
lof	0.070	0.513	0.002	0.125	0.273	0.261	0.286	0.474	0.655	0.095
ocsvm	0.119	0.519	0.002	0.128	0.387	0.482	0.562	0.396	0.956	0.242
complex_4										
sdo	0.002	0.529	0.004	0.132	0.385	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.509	0.002	0.124	0.117	0.998	0.999	1.000	1.000	0.984
hbos	0.001	0.515	0.003	0.126	0.193	0.838	0.851	0.971	0.989	0.664
iforest	0.001	0.518	0.003	0.127	0.223	0.985	0.987	1.000	1.000	0.946
knn	0.002	0.528	0.004	0.132	0.387	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.512	0.002	0.123	0.152	0.116	0.197	0.439	0.573	0.017
ocsvm	0.001	0.515	0.003	0.126	0.213	0.989	0.991	1.000	0.996	0.958
complex_5										
sdo	0.412	0.535	0.003	0.135	0.731	0.923	0.936	0.874	0.995	0.793
abod	0.133	0.512	0.002	0.125	0.296	0.902	0.928	0.879	0.993	0.721
hbos	0.197	0.519	0.002	0.128	0.416	0.630	0.651	0.620	0.948	0.357
iforest	0.280	0.526	0.002	0.131	0.552	0.833	0.847	0.815	0.987	0.627
knn	0.423	0.535	0.003	0.136	0.739	0.938	0.958	0.869	0.995	0.797
lof	0.301	0.532	0.002	0.132	0.509	0.143	0.240	0.494	0.529	0.030
ocsvm	0.244	0.523	0.002	0.131	0.455	0.787	0.801	0.739	0.975	0.556
complex_6										
sdo	0.481	0.538	0.006	0.136	0.754	0.655	0.764	0.477	0.977	0.403
abod	0.175	0.513	0.002	0.124	0.384	0.593	0.723	0.434	0.971	0.324
hbos	0.298	0.524	0.004	0.129	0.548	0.312	0.390	0.216	0.893	0.108
iforest	0.318	0.525	0.004	0.130	0.558	0.543	0.682	0.401	0.965	0.287
knn	0.478	0.538	0.006	0.136	0.751	0.682	0.798	0.458	0.978	0.371
lof	0.187	0.513	0.002	0.126	0.377	0.198	0.272	0.505	0.545	0.060
ocsvm	0.349	0.529	0.004	0.132	0.557	0.488	0.638	0.263	0.949	0.237
complex_7										
sdo	0.020	0.533	0.002	0.135	0.654	1.000	1.000	1.000	1.000	1.000
abod	0.005	0.506	0.002	0.124	0.162	1.000	1.000	1.000	1.000	0.989
hbos	0.007	0.516	0.005	0.126	0.256	0.771	0.869	0.973	0.975	0.735
iforest	0.010	0.519	0.004	0.128	0.339	0.998	0.999	1.000	1.000	0.992

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
knn	0.019	0.532	0.002	0.134	0.641	1.000	1.000	1.000	1.000	1.000
lof	0.012	0.512	0.001	0.124	0.368	0.121	0.256	0.483	0.457	0.020
ocsvm	0.012	0.522	0.005	0.130	0.393	0.907	0.950	0.996	0.990	0.761
complex_8										
sdo	0.290	0.546	0.003	0.144	0.601	0.961	0.964	0.990	0.999	0.893
abod	0.105	0.522	0.001	0.130	0.375	0.899	0.915	0.943	0.998	0.783
hbos	0.161	0.532	0.002	0.135	0.485	0.756	0.767	0.830	0.990	0.547
iforest	0.223	0.539	0.002	0.140	0.546	0.903	0.921	0.971	0.998	0.775
knn	0.298	0.547	0.003	0.145	0.614	0.973	0.979	0.990	1.000	0.920
lof	0.227	0.533	0.002	0.135	0.545	0.566	0.649	0.860	0.828	0.338
ocsvm	0.254	0.543	0.002	0.142	0.563	0.915	0.938	0.979	0.999	0.797
complex_9										
sdo	0.306	0.537	0.003	0.138	0.725	0.909	0.923	0.885	0.996	0.775
abod	0.082	0.507	0.002	0.125	0.244	0.828	0.863	0.828	0.992	0.624
hbos	0.125	0.520	0.003	0.129	0.386	0.360	0.508	0.299	0.920	0.138
iforest	0.163	0.522	0.003	0.130	0.448	0.712	0.733	0.749	0.982	0.471
knn	0.307	0.537	0.003	0.138	0.723	0.926	0.944	0.885	0.996	0.776
lof	0.161	0.516	0.001	0.129	0.383	0.109	0.207	0.419	0.608	0.021
ocsvm	0.171	0.522	0.002	0.133	0.435	0.643	0.671	0.653	0.965	0.389
complex_10										
sdo	0.637	0.545	0.004	0.142	0.841	0.904	0.934	0.884	0.994	0.754
abod	0.244	0.517	0.002	0.127	0.431	0.780	0.823	0.694	0.980	0.524
hbos	0.331	0.526	0.002	0.131	0.517	0.644	0.645	0.606	0.942	0.373
iforest	0.404	0.530	0.003	0.134	0.579	0.803	0.853	0.765	0.984	0.581
knn	0.634	0.545	0.004	0.142	0.838	0.906	0.938	0.844	0.993	0.726
lof	0.308	0.518	0.002	0.127	0.520	0.079	0.162	0.392	0.524	0.013
ocsvm	0.341	0.527	0.003	0.133	0.484	0.759	0.811	0.650	0.974	0.516
complex_11										
sdo	0.114	0.528	0.002	0.131	0.655	0.943	0.950	0.974	0.998	0.835
abod	0.032	0.510	0.002	0.123	0.242	0.842	0.886	0.851	0.989	0.610
hbos	0.062	0.518	0.002	0.127	0.412	0.654	0.666	0.752	0.951	0.381
iforest	0.088	0.524	0.002	0.129	0.535	0.874	0.879	0.932	0.992	0.695
knn	0.109	0.527	0.002	0.131	0.640	0.949	0.958	0.965	0.998	0.808
lof	0.044	0.512	0.002	0.124	0.326	0.120	0.187	0.450	0.515	0.023
ocsvm	0.061	0.516	0.002	0.126	0.358	0.857	0.861	0.930	0.984	0.665
complex_12										
sdo	0.003	0.527	0.004	0.132	0.260	0.916	0.921	0.984	0.997	0.790
abod	0.001	0.508	0.002	0.123	0.072	0.937	0.943	0.980	0.998	0.806
hbos	0.002	0.518	0.002	0.128	0.155	0.589	0.599	0.753	0.961	0.336

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
iforest	0.002	0.519	0.003	0.128	0.173	0.810	0.814	0.945	0.990	0.610
knn	0.003	0.527	0.004	0.132	0.264	0.956	0.960	0.993	0.999	0.883
lof	0.004	0.519	0.002	0.126	0.256	0.331	0.424	0.690	0.713	0.132
ocsvm	0.002	0.513	0.002	0.125	0.121	0.807	0.827	0.958	0.978	0.606
complex_13										
sdo	0.525	0.553	0.005	0.147	0.807	0.905	0.926	0.872	0.995	0.765
abod	0.162	0.516	0.002	0.127	0.329	0.828	0.869	0.778	0.990	0.617
hbos	0.358	0.537	0.003	0.138	0.630	0.700	0.736	0.667	0.978	0.452
iforest	0.386	0.539	0.004	0.139	0.620	0.833	0.864	0.827	0.991	0.639
knn	0.530	0.553	0.005	0.147	0.813	0.919	0.947	0.864	0.996	0.766
lof	0.284	0.529	0.008	0.135	0.468	0.212	0.269	0.525	0.579	0.061
ocsvm	0.369	0.537	0.004	0.138	0.564	0.851	0.876	0.820	0.992	0.668
complex_14										
sdo	0.481	0.548	0.003	0.144	0.760	0.998	0.998	1.000	1.000	0.992
abod	0.199	0.524	0.002	0.131	0.492	0.977	0.981	0.998	1.000	0.907
hbos	0.240	0.527	0.002	0.132	0.489	0.892	0.895	0.975	0.995	0.729
iforest	0.338	0.535	0.002	0.137	0.596	0.983	0.984	0.999	1.000	0.938
knn	0.485	0.548	0.003	0.144	0.763	0.998	0.999	1.000	1.000	0.992
lof	0.461	0.548	0.002	0.141	0.780	0.461	0.530	0.805	0.794	0.194
ocsvm	0.400	0.541	0.002	0.141	0.641	0.983	0.984	0.999	1.000	0.938
complex_15										
sdo	0.404	0.549	0.003	0.144	0.801	0.980	0.986	0.988	1.000	0.932
abod	0.150	0.521	0.002	0.129	0.394	0.940	0.941	0.973	0.998	0.803
hbos	0.238	0.532	0.002	0.134	0.580	0.604	0.631	0.723	0.956	0.334
iforest	0.286	0.537	0.003	0.137	0.624	0.901	0.909	0.964	0.996	0.750
knn	0.412	0.549	0.003	0.144	0.803	0.991	0.992	0.988	1.000	0.965
lof	0.331	0.530	0.002	0.135	0.578	0.188	0.301	0.465	0.764	0.039
ocsvm	0.241	0.534	0.002	0.136	0.487	0.863	0.875	0.907	0.989	0.681
complex_16										
sdo	0.392	0.553	0.003	0.147	0.645	0.851	0.863	0.868	0.995	0.682
abod	0.096	0.514	0.001	0.126	0.196	0.813	0.851	0.823	0.993	0.621
hbos	0.198	0.528	0.002	0.133	0.405	0.490	0.568	0.449	0.965	0.252
iforest	0.258	0.536	0.002	0.138	0.476	0.743	0.785	0.765	0.989	0.526
knn	0.387	0.552	0.003	0.146	0.636	0.903	0.926	0.861	0.997	0.719
lof	0.409	0.551	0.002	0.146	0.619	0.327	0.421	0.668	0.749	0.134
ocsvm	0.343	0.547	0.003	0.143	0.612	0.785	0.808	0.806	0.992	0.583
complex_17										
sdo	0.326	0.537	0.002	0.137	0.712	0.982	0.983	0.998	1.000	0.935
abod	0.089	0.513	0.001	0.125	0.229	0.970	0.977	0.980	0.999	0.884

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
hbos	0.143	0.521	0.002	0.129	0.369	0.576	0.628	0.647	0.945	0.301
iforest	0.178	0.524	0.002	0.130	0.431	0.861	0.867	0.949	0.993	0.676
knn	0.335	0.537	0.002	0.137	0.720	0.995	0.996	0.999	1.000	0.979
lof	0.189	0.518	0.002	0.126	0.397	0.255	0.352	0.629	0.689	0.072
ocsvm	0.132	0.519	0.001	0.128	0.306	0.820	0.822	0.936	0.979	0.606
complex_18										
sdo	0.055	0.534	0.002	0.135	0.577	0.988	0.991	1.000	1.000	0.956
abod	0.012	0.511	0.002	0.124	0.142	0.974	0.978	0.980	0.999	0.891
hbos	0.022	0.516	0.002	0.127	0.254	0.862	0.866	0.962	0.990	0.682
iforest	0.028	0.520	0.002	0.128	0.302	0.955	0.958	0.996	0.999	0.867
knn	0.056	0.534	0.002	0.135	0.584	0.992	0.993	1.000	1.000	0.969
lof	0.021	0.518	0.002	0.128	0.295	0.302	0.407	0.672	0.615	0.103
ocsvm	0.029	0.519	0.002	0.128	0.293	0.939	0.939	0.995	0.988	0.831
complex_19										
sdo	0.568	0.570	0.004	0.158	0.766	1.000	1.000	1.000	1.000	1.000
abod	0.119	0.516	0.002	0.128	0.214	0.985	0.986	0.997	1.000	0.929
hbos	0.256	0.536	0.002	0.138	0.408	0.864	0.867	0.971	0.994	0.689
iforest	0.295	0.540	0.003	0.141	0.440	0.989	0.991	0.999	1.000	0.960
knn	0.567	0.569	0.004	0.158	0.767	1.000	1.000	1.000	1.000	1.000
lof	0.302	0.532	0.002	0.144	0.428	0.220	0.335	0.591	0.561	0.068
ocsvm	0.406	0.551	0.003	0.148	0.543	0.989	0.990	1.000	1.000	0.960
complex_20										
sdo	0.001	0.518	0.011	0.128	0.038	0.982	0.985	1.000	1.000	0.938
abod	0.000	0.502	0.004	0.121	0.008	0.970	0.975	0.994	0.999	0.881
hbos	0.001	0.510	0.006	0.124	0.019	0.848	0.859	0.979	0.981	0.666
iforest	0.001	0.510	0.007	0.124	0.020	0.970	0.974	0.999	1.000	0.904
knn	0.001	0.519	0.010	0.129	0.039	0.991	0.993	1.000	1.000	0.968
lof	0.000	0.508	0.002	0.122	0.016	0.079	0.143	0.372	0.478	0.010
ocsvm	0.001	0.510	0.009	0.124	0.024	0.910	0.925	0.993	0.986	0.768
high-noise_1										
sdo	0.312	0.545	0.003	0.141	0.917	1.000	1.000	1.000	1.000	1.000
abod	0.175	0.522	0.002	0.131	0.531	1.000	1.000	1.000	1.000	1.000
hbos	0.166	0.524	0.002	0.131	0.502	0.826	0.906	0.987	0.989	0.791
iforest	0.198	0.528	0.002	0.133	0.581	1.000	1.000	1.000	1.000	1.000
knn	0.311	0.546	0.003	0.141	0.915	1.000	1.000	1.000	1.000	1.000
lof	0.230	0.531	0.001	0.141	0.838	0.036	0.162	0.391	0.513	0.001

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
ocsvm	0.273	0.539	0.003	0.138	0.777	0.982	0.990	1.000	0.999	0.932
high-noise_2										
sdo	0.016	0.529	0.003	0.132	0.730	1.000	1.000	1.000	1.000	1.000
abod	0.009	0.512	0.002	0.127	0.453	1.000	1.000	1.000	1.000	0.997
hbos	0.008	0.510	0.002	0.126	0.410	0.929	0.958	0.997	0.998	0.794
iforest	0.011	0.517	0.003	0.128	0.517	1.000	1.000	1.000	1.000	1.000
knn	0.017	0.530	0.003	0.132	0.732	1.000	1.000	1.000	1.000	1.000
lof	0.006	0.531	0.002	0.130	0.648	0.002	0.184	0.336	0.496	-0.00
ocsvm	0.008	0.505	0.002	0.126	0.337	0.969	0.984	1.000	0.985	0.892
high-noise_3										
sdo	0.046	0.528	0.002	0.131	0.840	1.000	1.000	1.000	1.000	1.000
abod	0.029	0.516	0.002	0.127	0.582	1.000	1.000	1.000	1.000	1.000
hbos	0.028	0.515	0.002	0.127	0.568	0.973	0.986	1.000	0.997	0.951
iforest	0.034	0.519	0.002	0.128	0.613	1.000	1.000	1.000	1.000	1.000
knn	0.047	0.528	0.002	0.131	0.839	1.000	1.000	1.000	1.000	1.000
lof	0.032	0.516	0.002	0.126	0.738	-0.04	0.204	0.224	0.493	0.001
ocsvm	0.021	0.507	0.001	0.125	0.334	0.912	0.954	0.997	0.936	0.753
high-noise_4										
sdo	0.385	0.569	0.004	0.154	0.970	1.000	1.000	1.000	1.000	1.000
abod	0.256	0.543	0.003	0.142	0.640	1.000	1.000	1.000	1.000	1.000
hbos	0.243	0.541	0.003	0.141	0.607	1.000	1.000	1.000	1.000	1.000
iforest	0.264	0.546	0.003	0.143	0.666	1.000	1.000	1.000	1.000	1.000
knn	0.386	0.569	0.004	0.154	0.973	1.000	1.000	1.000	1.000	1.000
lof	0.369	0.557	0.002	0.147	0.939	0.082	0.224	0.468	0.576	0.007
ocsvm	0.365	0.568	0.004	0.154	0.929	1.000	1.000	1.000	1.000	1.000
high-noise_5										
sdo	0.619	0.539	0.003	0.137	0.961	1.000	1.000	1.000	1.000	1.000
abod	0.468	0.523	0.002	0.132	0.729	1.000	1.000	1.000	1.000	1.000
hbos	0.510	0.531	0.003	0.134	0.795	0.998	0.999	1.000	1.000	0.993
iforest	0.516	0.530	0.003	0.134	0.801	1.000	1.000	1.000	1.000	1.000
knn	0.620	0.539	0.003	0.137	0.963	1.000	1.000	1.000	1.000	1.000
lof	0.513	0.534	0.001	0.132	0.883	-0.19	0.219	-0.01	0.377	0.036
ocsvm	0.616	0.539	0.003	0.137	0.949	1.000	1.000	1.000	1.000	1.000
high-noise_6										
sdo	0.631	0.536	0.003	0.136	0.960	1.000	1.000	1.000	1.000	1.000
abod	0.485	0.524	0.002	0.131	0.742	1.000	1.000	1.000	1.000	0.998
hbos	0.409	0.518	0.002	0.130	0.627	0.988	0.994	1.000	1.000	0.975
iforest	0.475	0.525	0.002	0.132	0.722	1.000	1.000	1.000	1.000	1.000
knn	0.631	0.537	0.003	0.136	0.961	1.000	1.000	1.000	1.000	1.000

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
lof	0.580	0.538	0.001	0.134	0.932	-0.07	0.239	0.211	0.496	0.004
ocsvm	0.616	0.537	0.003	0.136	0.929	1.000	1.000	1.000	1.000	1.000
high-noise_7										
sdo	0.001	0.513	0.004	0.125	0.066	1.000	1.000	1.000	1.000	1.000
abod	0.000	0.504	0.003	0.122	0.033	1.000	1.000	1.000	1.000	0.996
hbos	0.000	0.503	0.003	0.122	0.032	0.854	0.923	0.992	0.958	0.812
iforest	0.001	0.506	0.004	0.123	0.045	0.999	1.000	1.000	1.000	0.995
knn	0.001	0.513	0.004	0.125	0.065	1.000	1.000	1.000	1.000	1.000
lof	0.000	0.503	0.003	0.120	0.014	-0.04	0.246	0.242	0.499	0.002
ocsvm	0.000	0.502	0.003	0.122	0.028	0.636	0.786	0.946	0.766	0.330
high-noise_8										
sdo	0.004	0.526	0.002	0.130	0.620	1.000	1.000	1.000	1.000	1.000
abod	0.002	0.507	0.002	0.125	0.337	1.000	1.000	1.000	1.000	1.000
hbos	0.002	0.510	0.002	0.125	0.327	0.925	0.961	0.998	0.992	0.890
iforest	0.003	0.513	0.002	0.126	0.383	0.998	0.999	1.000	1.000	0.990
knn	0.004	0.526	0.002	0.130	0.617	1.000	1.000	1.000	1.000	1.000
lof	0.002	0.522	0.003	0.136	0.427	-0.05	0.178	0.219	0.482	0.002
ocsvm	0.003	0.514	0.002	0.127	0.451	0.991	0.996	1.000	0.997	0.962
high-noise_9										
sdo	0.401	0.541	0.002	0.138	0.931	1.000	1.000	1.000	1.000	1.000
abod	0.252	0.525	0.002	0.131	0.597	1.000	1.000	1.000	1.000	0.990
hbos	0.227	0.523	0.002	0.130	0.538	0.928	0.951	0.996	0.996	0.793
iforest	0.254	0.526	0.002	0.131	0.589	0.997	0.998	1.000	1.000	0.984
knn	0.402	0.542	0.002	0.138	0.933	1.000	1.000	1.000	1.000	1.000
lof	0.331	0.554	0.001	0.145	0.879	0.085	0.231	0.480	0.588	0.008
ocsvm	0.148	0.515	0.002	0.127	0.333	0.768	0.864	0.975	0.978	0.511
high-noise_10										
sdo	0.373	0.568	0.004	0.155	0.968	1.000	1.000	1.000	1.000	1.000
abod	0.248	0.543	0.003	0.142	0.640	1.000	1.000	1.000	1.000	0.972
hbos	0.225	0.540	0.003	0.140	0.581	1.000	1.000	1.000	1.000	1.000
iforest	0.234	0.543	0.003	0.142	0.609	1.000	1.000	1.000	1.000	1.000
knn	0.374	0.568	0.004	0.155	0.972	1.000	1.000	1.000	1.000	1.000
lof	0.337	0.566	0.002	0.154	0.911	0.001	0.167	0.332	0.485	-0.00
ocsvm	0.349	0.567	0.004	0.155	0.921	1.000	1.000	1.000	1.000	1.000
high-noise_11										
sdo	0.001	0.520	0.004	0.128	0.342	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.499	0.003	0.124	0.199	1.000	1.000	1.000	1.000	1.000
hbos	0.001	0.500	0.002	0.124	0.181	0.845	0.918	0.990	0.986	0.804
iforest	0.001	0.506	0.003	0.125	0.234	1.000	1.000	1.000	1.000	1.000

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
knn	0.001	0.521	0.004	0.128	0.341	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.518	0.001	0.130	0.247	-0.11	0.201	0.112	0.421	0.012
ocsvm	0.001	0.493	0.003	0.124	0.185	0.971	0.985	1.000	0.979	0.897
high-noise_12										
sdo	0.000	0.511	0.007	0.124	0.023	1.000	1.000	1.000	1.000	1.000
abod	0.000	0.495	0.005	0.121	0.009	0.999	1.000	1.000	1.000	0.994
hbos	0.000	0.504	0.006	0.121	0.010	0.677	0.787	0.808	0.930	0.448
iforest	0.000	0.505	0.007	0.122	0.015	0.824	0.893	0.979	0.990	0.583
knn	0.000	0.512	0.006	0.125	0.024	1.000	1.000	1.000	1.000	1.000
lof	0.000	0.506	0.003	0.123	0.007	-0.08	0.221	0.190	0.464	0.005
ocsvm	0.000	0.497	0.006	0.122	0.013	0.798	0.810	0.782	0.863	0.541
high-noise_13										
sdo	0.447	0.552	0.003	0.144	0.970	1.000	1.000	1.000	1.000	1.000
abod	0.328	0.535	0.002	0.137	0.708	1.000	1.000	1.000	1.000	0.982
hbos	0.250	0.529	0.002	0.133	0.538	0.967	0.983	1.000	0.997	0.940
iforest	0.272	0.532	0.002	0.135	0.590	1.000	1.000	1.000	1.000	1.000
knn	0.448	0.552	0.003	0.145	0.972	1.000	1.000	1.000	1.000	1.000
lof	0.417	0.548	0.002	0.145	0.933	-0.07	0.204	0.236	0.476	0.004
ocsvm	0.314	0.538	0.002	0.139	0.697	0.993	0.996	1.000	0.997	0.968
high-noise_14										
sdo	0.631	0.549	0.002	0.142	0.963	1.000	1.000	1.000	1.000	1.000
abod	0.423	0.530	0.002	0.134	0.646	1.000	1.000	1.000	1.000	0.987
hbos	0.454	0.533	0.002	0.136	0.695	0.999	1.000	1.000	1.000	0.997
iforest	0.465	0.535	0.002	0.136	0.710	1.000	1.000	1.000	1.000	1.000
knn	0.633	0.549	0.002	0.142	0.965	1.000	1.000	1.000	1.000	1.000
lof	0.584	0.538	0.002	0.139	0.921	-0.02	0.196	0.302	0.506	0.000
ocsvm	0.357	0.526	0.002	0.133	0.542	0.995	0.998	1.000	0.997	0.978
high-noise_15										
sdo	0.001	0.518	0.005	0.128	0.030	1.000	1.000	1.000	1.000	1.000
abod	0.000	0.499	0.004	0.122	0.010	1.000	1.000	1.000	1.000	0.993
hbos	0.001	0.504	0.003	0.123	0.013	0.904	0.950	0.996	0.977	0.871
iforest	0.001	0.506	0.004	0.123	0.014	0.993	0.995	1.000	1.000	0.971
knn	0.001	0.520	0.005	0.129	0.033	1.000	1.000	1.000	1.000	1.000
lof	0.000	0.522	0.004	0.132	0.018	0.058	0.144	0.371	0.518	0.004
ocsvm	0.000	0.501	0.004	0.123	0.014	0.416	0.689	0.816	0.874	0.157
high-noise_16										
sdo	0.545	0.546	0.003	0.141	0.974	1.000	1.000	1.000	1.000	1.000
abod	0.422	0.531	0.003	0.135	0.752	1.000	1.000	1.000	1.000	0.996
hbos	0.364	0.527	0.002	0.134	0.650	0.999	1.000	1.000	1.000	0.998

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
iforest	0.404	0.531	0.003	0.135	0.724	1.000	1.000	1.000	1.000	1.000
knn	0.546	0.546	0.003	0.141	0.976	1.000	1.000	1.000	1.000	1.000
lof	0.535	0.548	0.002	0.140	0.947	-0.05	0.239	0.276	0.520	0.002
ocsvm	0.527	0.547	0.003	0.142	0.954	0.999	1.000	1.000	1.000	0.995
high-noise_17										
sdo	0.118	0.540	0.004	0.139	0.873	1.000	1.000	1.000	1.000	1.000
abod	0.067	0.520	0.003	0.130	0.532	1.000	1.000	1.000	1.000	0.987
hbos	0.050	0.514	0.003	0.128	0.393	0.845	0.906	0.987	0.989	0.636
iforest	0.064	0.519	0.003	0.129	0.472	0.987	0.993	1.000	1.000	0.949
knn	0.118	0.541	0.004	0.138	0.877	1.000	1.000	1.000	1.000	1.000
lof	0.082	0.539	0.001	0.133	0.802	0.070	0.206	0.441	0.548	0.005
ocsvm	0.064	0.518	0.003	0.130	0.447	0.817	0.895	0.985	0.988	0.589
high-noise_18										
sdo	0.018	0.529	0.003	0.132	0.788	1.000	1.000	1.000	1.000	1.000
abod	0.010	0.515	0.002	0.126	0.468	1.000	1.000	1.000	1.000	0.997
hbos	0.008	0.513	0.002	0.126	0.398	0.755	0.863	0.973	0.985	0.725
iforest	0.010	0.516	0.002	0.127	0.472	0.996	0.997	1.000	1.000	0.980
knn	0.018	0.529	0.003	0.132	0.790	1.000	1.000	1.000	1.000	1.000
lof	0.010	0.519	0.001	0.125	0.686	0.030	0.175	0.376	0.523	0.001
ocsvm	0.006	0.507	0.002	0.124	0.237	0.699	0.792	0.936	0.885	0.419
high-noise_19										
sdo	0.404	0.546	0.003	0.141	0.977	1.000	1.000	1.000	1.000	1.000
abod	0.297	0.532	0.002	0.135	0.713	1.000	1.000	1.000	1.000	0.998
hbos	0.294	0.532	0.002	0.135	0.708	0.965	0.982	1.000	0.999	0.937
iforest	0.308	0.534	0.002	0.136	0.747	1.000	1.000	1.000	1.000	1.000
knn	0.404	0.546	0.003	0.141	0.979	1.000	1.000	1.000	1.000	1.000
lof	0.382	0.542	0.002	0.133	0.952	-0.08	0.235	0.225	0.486	0.005
ocsvm	0.222	0.526	0.002	0.132	0.549	0.994	0.997	1.000	0.997	0.973
high-noise_20										
sdo	0.001	0.516	0.003	0.126	0.169	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.506	0.003	0.123	0.098	1.000	1.000	1.000	1.000	0.998
hbos	0.001	0.504	0.003	0.123	0.090	0.804	0.885	0.981	0.968	0.729
iforest	0.001	0.507	0.003	0.124	0.107	0.977	0.981	1.000	0.999	0.914
knn	0.001	0.517	0.003	0.126	0.167	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.519	0.004	0.131	0.087	-0.08	0.215	0.204	0.452	0.005
ocsvm	0.001	0.499	0.002	0.122	0.071	0.779	0.796	0.925	0.856	0.514

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
low-noise_1										
sdo	0.001	0.532	0.005	0.134	0.120	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.508	0.003	0.124	0.036	1.000	1.000	1.000	1.000	1.000
hbos	0.001	0.508	0.003	0.124	0.038	0.894	0.923	0.990	0.973	0.763
iforest	0.001	0.513	0.003	0.125	0.050	0.972	0.986	1.000	0.999	0.912
knn	0.001	0.533	0.004	0.135	0.120	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.531	0.005	0.132	0.082	0.364	0.526	0.752	0.717	0.144
ocsvm	0.001	0.505	0.003	0.123	0.030	0.813	0.881	0.981	0.891	0.605
low-noise_2										
sdo	0.337	0.557	0.002	0.148	0.894	1.000	1.000	1.000	1.000	1.000
abod	0.162	0.528	0.002	0.133	0.438	1.000	1.000	1.000	1.000	0.992
hbos	0.124	0.522	0.002	0.130	0.336	0.976	0.983	1.000	0.999	0.930
iforest	0.155	0.527	0.002	0.132	0.412	1.000	1.000	1.000	1.000	1.000
knn	0.339	0.558	0.002	0.149	0.895	1.000	1.000	1.000	1.000	1.000
lof	0.302	0.560	0.002	0.149	0.837	0.222	0.394	0.623	0.608	0.057
ocsvm	0.142	0.524	0.002	0.132	0.364	0.988	0.993	1.000	0.996	0.955
low-noise_3										
sdo	0.570	0.623	0.004	0.193	0.857	1.000	1.000	1.000	1.000	1.000
abod	0.201	0.546	0.002	0.145	0.296	1.000	1.000	1.000	1.000	0.987
hbos	0.183	0.545	0.002	0.143	0.271	0.946	0.955	0.997	0.999	0.858
iforest	0.174	0.542	0.002	0.142	0.258	1.000	1.000	1.000	1.000	1.000
knn	0.576	0.623	0.004	0.193	0.856	1.000	1.000	1.000	1.000	1.000
lof	0.655	0.625	0.003	0.194	0.835	0.604	0.756	0.914	0.845	0.366
ocsvm	0.327	0.576	0.003	0.163	0.489	0.993	0.996	1.000	1.000	0.974
low-noise_4										
sdo	0.558	0.612	0.004	0.187	0.809	1.000	1.000	1.000	1.000	1.000
abod	0.177	0.538	0.002	0.141	0.256	1.000	1.000	1.000	1.000	0.968
hbos	0.137	0.531	0.002	0.136	0.199	0.983	0.992	1.000	0.999	0.953
iforest	0.156	0.535	0.002	0.138	0.227	1.000	1.000	1.000	1.000	1.000
knn	0.559	0.611	0.004	0.187	0.809	1.000	1.000	1.000	1.000	1.000
lof	0.526	0.606	0.002	0.180	0.734	0.416	0.596	0.794	0.699	0.207
ocsvm	0.332	0.570	0.004	0.160	0.483	1.000	1.000	1.000	1.000	1.000
low-noise_5										
sdo	0.001	0.525	0.003	0.131	0.119	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.505	0.003	0.123	0.038	1.000	1.000	1.000	1.000	0.996
hbos	0.001	0.507	0.004	0.123	0.042	0.733	0.840	0.963	0.967	0.586
iforest	0.001	0.511	0.004	0.125	0.055	0.972	0.978	0.999	0.999	0.907
knn	0.001	0.526	0.003	0.131	0.121	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.518	0.007	0.126	0.060	0.225	0.372	0.615	0.636	0.054

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
ocsvm	0.001	0.504	0.004	0.123	0.034	0.563	0.621	0.831	0.907	0.286
low-noise_6										
sdo	0.002	0.536	0.004	0.136	0.458	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.508	0.002	0.126	0.188	1.000	1.000	1.000	1.000	0.993
hbos	0.002	0.521	0.003	0.130	0.297	0.986	0.993	1.000	0.999	0.966
iforest	0.002	0.520	0.003	0.130	0.292	1.000	1.000	1.000	1.000	1.000
knn	0.002	0.537	0.004	0.137	0.458	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.531	0.002	0.135	0.278	0.063	0.179	0.391	0.537	0.006
ocsvm	0.002	0.532	0.004	0.135	0.425	1.000	1.000	1.000	1.000	1.000
low-noise_7										
sdo	0.365	0.572	0.002	0.157	0.856	1.000	1.000	1.000	1.000	1.000
abod	0.137	0.527	0.002	0.134	0.329	1.000	1.000	1.000	1.000	0.995
hbos	0.109	0.523	0.001	0.131	0.261	0.899	0.942	0.994	0.992	0.755
iforest	0.130	0.527	0.001	0.133	0.306	1.000	1.000	1.000	1.000	1.000
knn	0.370	0.573	0.002	0.158	0.870	1.000	1.000	1.000	1.000	1.000
lof	0.309	0.553	0.002	0.144	0.817	0.422	0.598	0.806	0.777	0.191
ocsvm	0.255	0.552	0.002	0.147	0.573	0.958	0.979	0.999	0.998	0.877
low-noise_8										
sdo	0.001	0.530	0.005	0.134	0.278	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.508	0.003	0.125	0.113	1.000	1.000	1.000	1.000	1.000
hbos	0.001	0.509	0.004	0.124	0.090	0.610	0.765	0.924	0.946	0.617
iforest	0.001	0.511	0.004	0.125	0.119	0.992	0.995	1.000	1.000	0.969
knn	0.001	0.531	0.005	0.134	0.277	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.523	0.003	0.129	0.160	0.201	0.356	0.592	0.655	0.046
ocsvm	0.001	0.502	0.003	0.124	0.080	0.817	0.892	0.984	0.921	0.599
low-noise_9										
sdo	0.157	0.556	0.003	0.147	0.843	1.000	1.000	1.000	1.000	1.000
abod	0.068	0.524	0.002	0.131	0.378	1.000	1.000	1.000	1.000	0.995
hbos	0.053	0.520	0.002	0.129	0.296	0.723	0.771	0.938	0.978	0.549
iforest	0.070	0.525	0.002	0.132	0.381	1.000	1.000	1.000	1.000	1.000
knn	0.161	0.557	0.003	0.148	0.856	1.000	1.000	1.000	1.000	1.000
lof	0.143	0.552	0.002	0.147	0.782	0.266	0.444	0.666	0.658	0.077
ocsvm	0.127	0.545	0.003	0.142	0.664	0.990	0.995	1.000	0.998	0.963
low-noise_10										
sdo	0.579	0.586	0.004	0.165	0.896	1.000	1.000	1.000	1.000	1.000
abod	0.241	0.536	0.002	0.138	0.376	1.000	1.000	1.000	1.000	0.980
hbos	0.206	0.533	0.003	0.136	0.322	0.903	0.941	0.995	0.991	0.776
iforest	0.213	0.534	0.003	0.136	0.331	1.000	1.000	1.000	1.000	1.000
knn	0.581	0.586	0.004	0.166	0.900	1.000	1.000	1.000	1.000	1.000

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
lof	0.554	0.572	0.002	0.162	0.885	0.493	0.670	0.858	0.812	0.251
ocsvm	0.401	0.562	0.004	0.153	0.616	0.993	0.997	1.000	0.999	0.974
low-noise_11										
sdo	0.001	0.526	0.012	0.132	0.117	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.490	0.004	0.123	0.038	1.000	1.000	1.000	1.000	1.000
hbos	0.001	0.509	0.006	0.125	0.050	0.961	0.978	0.999	0.995	0.929
iforest	0.001	0.504	0.007	0.125	0.056	1.000	1.000	1.000	1.000	1.000
knn	0.001	0.528	0.011	0.133	0.116	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.533	0.001	0.141	0.072	0.062	0.187	0.398	0.478	0.007
ocsvm	0.001	0.496	0.007	0.125	0.056	0.976	0.988	1.000	0.981	0.921
low-noise_12										
sdo	0.492	0.612	0.005	0.186	0.924	1.000	1.000	1.000	1.000	1.000
abod	0.224	0.550	0.003	0.149	0.413	1.000	1.000	1.000	1.000	0.995
hbos	0.180	0.541	0.002	0.143	0.335	0.998	0.998	1.000	1.000	0.990
iforest	0.216	0.549	0.003	0.148	0.401	1.000	1.000	1.000	1.000	1.000
knn	0.500	0.612	0.005	0.186	0.930	1.000	1.000	1.000	1.000	1.000
lof	0.583	0.601	0.003	0.179	0.895	0.477	0.654	0.847	0.806	0.233
ocsvm	0.373	0.585	0.004	0.169	0.698	1.000	1.000	1.000	1.000	1.000
low-noise_13										
sdo	0.410	0.596	0.003	0.176	0.771	1.000	1.000	1.000	1.000	1.000
abod	0.130	0.532	0.002	0.138	0.248	1.000	1.000	1.000	1.000	0.951
hbos	0.171	0.541	0.002	0.143	0.325	1.000	1.000	1.000	1.000	1.000
iforest	0.139	0.534	0.002	0.139	0.264	1.000	1.000	1.000	1.000	1.000
knn	0.424	0.599	0.003	0.178	0.800	1.000	1.000	1.000	1.000	1.000
lof	0.407	0.594	0.002	0.180	0.803	0.599	0.753	0.911	0.852	0.374
ocsvm	0.291	0.569	0.002	0.160	0.548	1.000	1.000	1.000	1.000	1.000
low-noise_14										
sdo	0.052	0.553	0.003	0.145	0.808	1.000	1.000	1.000	1.000	1.000
abod	0.022	0.520	0.002	0.130	0.372	1.000	1.000	1.000	1.000	0.992
hbos	0.019	0.518	0.002	0.129	0.309	1.000	1.000	1.000	1.000	1.000
iforest	0.024	0.522	0.002	0.131	0.381	1.000	1.000	1.000	1.000	1.000
knn	0.052	0.553	0.003	0.145	0.813	1.000	1.000	1.000	1.000	1.000
lof	0.031	0.549	0.002	0.143	0.695	0.245	0.422	0.649	0.697	0.068
ocsvm	0.027	0.524	0.002	0.132	0.419	0.998	0.999	1.000	0.998	0.991
low-noise_15										
sdo	0.002	0.549	0.004	0.143	0.485	1.000	1.000	1.000	1.000	1.000
abod	0.001	0.514	0.002	0.127	0.156	1.000	1.000	1.000	1.000	0.990
hbos	0.001	0.520	0.002	0.129	0.213	0.947	0.961	0.996	0.996	0.854
iforest	0.001	0.518	0.002	0.128	0.192	1.000	1.000	1.000	1.000	1.000

Continued on next page

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
knn	0.002	0.549	0.004	0.143	0.482	1.000	1.000	1.000	1.000	1.000
lof	0.001	0.535	0.003	0.135	0.334	0.362	0.544	0.756	0.739	0.144
ocsvm	0.001	0.524	0.003	0.132	0.257	0.976	0.986	1.000	0.994	0.923
low-noise_16										
sdo	0.545	0.589	0.004	0.170	0.943	1.000	1.000	1.000	1.000	1.000
abod	0.283	0.546	0.003	0.145	0.484	1.000	1.000	1.000	1.000	0.940
hbos	0.190	0.533	0.002	0.137	0.327	0.931	0.948	0.997	0.998	0.816
iforest	0.218	0.537	0.002	0.140	0.376	1.000	1.000	1.000	1.000	1.000
knn	0.548	0.590	0.004	0.170	0.942	1.000	1.000	1.000	1.000	1.000
lof	0.609	0.600	0.003	0.174	0.923	0.340	0.526	0.744	0.727	0.122
ocsvm	0.427	0.574	0.004	0.162	0.743	1.000	1.000	1.000	1.000	1.000
low-noise_17										
sdo	0.560	0.648	0.005	0.209	0.862	1.000	1.000	1.000	1.000	1.000
abod	0.178	0.549	0.002	0.147	0.264	1.000	1.000	1.000	1.000	0.971
hbos	0.118	0.535	0.002	0.138	0.175	0.993	0.994	1.000	1.000	0.980
iforest	0.140	0.540	0.002	0.141	0.208	1.000	1.000	1.000	1.000	1.000
knn	0.569	0.647	0.005	0.209	0.862	1.000	1.000	1.000	1.000	1.000
lof	0.704	0.649	0.003	0.209	0.856	0.468	0.644	0.834	0.742	0.237
ocsvm	0.378	0.602	0.004	0.180	0.583	0.996	0.998	1.000	1.000	0.986
low-noise_18										
sdo	0.605	0.595	0.005	0.173	0.914	1.000	1.000	1.000	1.000	1.000
abod	0.259	0.541	0.002	0.142	0.390	1.000	1.000	1.000	1.000	0.969
hbos	0.235	0.539	0.003	0.140	0.356	0.998	0.999	1.000	1.000	0.990
iforest	0.255	0.542	0.003	0.142	0.385	1.000	1.000	1.000	1.000	1.000
knn	0.609	0.596	0.004	0.173	0.917	1.000	1.000	1.000	1.000	1.000
lof	0.598	0.594	0.002	0.175	0.854	0.363	0.546	0.758	0.750	0.136
ocsvm	0.479	0.576	0.004	0.162	0.724	1.000	1.000	1.000	1.000	1.000
low-noise_19										
sdo	0.536	0.601	0.003	0.179	0.817	1.000	1.000	1.000	1.000	1.000
abod	0.176	0.536	0.002	0.139	0.268	1.000	1.000	1.000	1.000	0.981
hbos	0.128	0.528	0.002	0.134	0.197	0.873	0.918	0.988	0.997	0.811
iforest	0.150	0.532	0.002	0.136	0.229	1.000	1.000	1.000	1.000	1.000
knn	0.546	0.601	0.003	0.180	0.829	1.000	1.000	1.000	1.000	1.000
lof	0.545	0.585	0.002	0.177	0.818	0.572	0.732	0.898	0.825	0.337
ocsvm	0.298	0.561	0.002	0.154	0.450	0.964	0.982	0.999	0.999	0.898
low-noise_20										
sdo	0.462	0.588	0.005	0.171	0.839	1.000	1.000	1.000	1.000	1.000
abod	0.167	0.532	0.002	0.137	0.305	1.000	1.000	1.000	1.000	0.994
hbos	0.158	0.533	0.002	0.137	0.291	0.832	0.885	0.981	0.984	0.736

Continued on next page

A. APPENDIX

Score	LS	DT	LL	Xtree	Ireos	P@n	F1	AP	AUC	AMI
iforest	0.163	0.533	0.002	0.137	0.297	0.997	0.999	1.000	1.000	0.988
knn	0.467	0.589	0.005	0.172	0.847	1.000	1.000	1.000	1.000	1.000
lof	0.444	0.598	0.002	0.177	0.790	0.322	0.501	0.713	0.659	0.125
ocsvm	0.224	0.545	0.003	0.144	0.407	0.976	0.988	1.000	0.997	0.926

List of Figures

1.1	Different types of outliers. Contextual outliers are not represented in the figure. Drawn based on [Kha21, Figure 2]	2
2.1	Outlier detection by Z -Scores and modified Z -Scores. Each data point outside the outermost blue ellipsis is considered an outlier. The underlying code is inspired by [Mis23f]	9
2.2	Visualization of LOF outlier scores using ELKI. The dataset is artificially generated to highlight LOF strengths. Some interesting LOF scores are printed for $k=5$. [WC10]	10
2.3	Principle of distance-based outlier detection. A datapoint x is classified as an outlier if less than k points are within distance d from x . Drawn based on [KN22, Slide 4].	11
2.4	The intuition of angle-based outlier detection. Samples that are further away tend to construct smaller angles. [KSZ08, Figure 1].	11
2.5	The intuition of histogram-based outlier scores. High bin heights in the histogram indicate low HBOS scores and inliers. Low bin heights are interpreted as outliers.	12
2.6	Outlier detection intuition in one-class SVMs. The data is separated from the origin with a maximum margin by applying a function f that returns +1 in "small" regions capturing most of the data points and -1 elsewhere. [AbdO06, Figure 1]	13
2.7	Isolation trees for different data samples. Inliers tend to be harder separable than outliers. [HCB18, Figure 5]	14
2.8	Confusion Matrix for Outlier Detection Problems. [WT20, Table 2]	16
2.9	Different ROC Curves and their Interpretation. [cmg18]	17
2.10	Silhouette Plot with corresponding data and clustering [Lea23].	21
2.11	Visualization of the Kernel Trick. A kernel function projects the input space into a higher dimensional space where both classes are linearly separable [GCP05, Figure 1].	24
2.12	Separability curves for different points labeled as outliers. [MCZS15, Figure 1]	26

3.1	2D representation of different problem instances for each category. Each dataset was dimensionality reduced via T-SNE, which preserves the local structure and clustering of the data. Instances in grey or '-1' correspond to outliers and from 0 to X to different clusters of inliers. (but for the last figure, in which '0' means outliers)	30
3.2	2D representation by using PCA of the same problem instances shown in Figure 3.1. Different from T-SNE, PCA only preserves the global structure rather than local similarities and is highly affected by outliers. Alike Figure 3.1 instances in grey correspond to outliers.	31
3.3	Histograms of outlier scores for data points across all 80 datasets before and after the normalization by Kriegel. All normalized scores are transformed into the interval [0,1] where values close to 0 represent inliers and 1 outliers. Each histogram consists of 100 bins.	35
3.4	The main principle of SVMs. The predictor tries to find an optimal hyperplane by maximizing the margin between the support vectors. [SO17, Figure 2]	39
3.5	Principles of decision trees and ensemble models. Random forests use parallel bagging to improve the predictive performance of decision trees. In gradient-boosted trees, multiple weak learners are sequentially combined into a stronger model. [Si20, Figure 1]	43
3.6	Sliding windows in FIREOS. The smaller the ratio the smaller the window frame. A window ratio of 1.0 equals the full dataset.	46
3.7	Performance comparison of different programming languages for different computations. [GMP+20, Figure 2]	49
3.8	Service diagram and workflow of the three main functions of the FIREOS module. 1) fireos: Main implementation of the algorithm that calculates the separabilities of a given problem instance and defined predictor. 2) normalize_solutions!: Loads raw solution scores from disk and transforms them into normalized solutions that can be interpreted by FIREOS. 3) evaluate_solutions: Calculates and persists the final FIREOS scorings from given solutions and a trained model.	50
3.9	Technical conceptualization and relationships between components of FIREOS. Both helper APIs that access the FIREOS module interface can be called from outside. "fireos_utils" provides additional functions to call fireos "out-of-the-box" and "fireos_cli" provides a basic command line interface. All components below the interface definition can be summed up as the backend of the module.	52
3.10	This figure serves as an extension to Figure 3.3 as it shows all the remaining outlier scores that are not mentioned in [KKSZ11] but are treated in this work. The column on the left shows histograms of raw unprocessed scores differently from the right column which shows the same scores but normalized. Each histogram consists of 100 bins.	54

3.11	Excerpt of the SVM predictor function in parallelized FIREOS. To prevent PyCall from crashing all Scikit-Learn functions are outsourced into the function "sk_svm_par" and called sequentially.	59
3.12	Excerpt of different runtimes in seconds for smaller datasets. Outliers in the first row of the table show the impact of precompilation tasks in Julia. . .	60
3.13	Workflows for IREOS and FIREOS. The main difference between them is that FIREOS trains the predictor independent from any solution. Since FIREOS always calculates the separability of all data samples, no solution has to be known during the training phase of the algorithms. After the training, an arbitrary number of solutions can be evaluated at the same time. IREOS on the other hand trains and evaluates once for each solution. In terms of performance, this is a disadvantage when evaluating a large set of different solutions. However, IREOS ignores the separability calculation for data points that are rated as 0 in the solutions, since they do not contribute to the final score. This accelerates the algorithm, especially for sparse solutions. . . .	64
3.14	Examples of separability calculations between FIREOS and IREOS. Although the same data and solution matrix are used, FIREOS always outputs exactly one separability vector but IREOS one for each solution. Furthermore, the separability matrix of IREOS contains zero for any zero in the solution vector, since IREOS ignores those instances. In order to modify IREOS to behave similarly to FIREOS, we instead of passing the solution matrix, provide an artificial solution vector of ones to ensure the calculation of all samples and force IREOS to calculate a separability vector rather than a matrix. . . .	65
3.15	Outline of timeline and runtime measuring during the experiments.	66
4.1	Visualization of all the steps we perform to obtain distance matrices of Z-Scores from an example dataset.	72
4.2	Examples of different validation scorings where the consideration of only one distance matrix (scores or ranks) would lead to an incomplete picture: Given are six scoring results of six fictional problem instances where we consider score1 as the gold standard. Score1 and score2 are very similar due to their distance matrices although their initial scorings are very different. Score3 looks very similar to score2 but shows strong deviations when looking at ranks. The distance between score4 and score1 is larger than to score3 and score1, however, the distance between the ranks is completely different. Lastly, score5 and score6 show another ranking problem. Although both predictors "misclassify" exactly one rank (score5 misclassifies row 1 and score6 misclassifies row 5) the outcome of the distance matrix is very different considering the distance of score6 the second highest. The misclassification of higher ranks is penalized more than lower ranks because the error is propagating further.	73

4.3	Distance matrix of different evaluation metrics that are executed on the seven outlier solutions for each basic2d dataset. Since FIREOS predictors return values on different scales, Z-Scores are used. Lower values in red represent more similar scorings and high values are more distinctive. It can be observed that most of the scorings performed by FIREOS predictors are closer to IREOS scorings than external validation. Obviously, external scorings show a high degree of similarity to the "consensus" metric. Lastly, LibLINEAR is shown to be the method that evaluates most differently among the compared predictors. Although many distances are colored in blue, there are no signs of strong deviations between any algorithm since the maximum number of standard deviations is 1.01.	74
4.4	Distance matrix of different evaluation rankings executed on seven outlier solutions for each basic2d dataset. Lower values in red represent a lower distance between ranks, hence lower MAE. Most FIREOS predictors show similar distances to the consensus metric indicating common agreement of ranks. Only decision_tree_sklearn, random_forest_native and liblinear show strong deviations to consensus. When looking at other pairs they still show larger distances than any other FIREOS predictor leading to the assumption that one joint deviating prediction is the reason. Different from the distance matrix between Z-Scores, the MAE between predictors tends to be generally higher. However, as already stated before rank-similarity is especially sensitive to small misclassifications.	75
4.5	Z-Score distance matrix of different evaluation metrics executed on seven outlier solutions for each of the larger complex/high-noise/low-noise datasets. Lower values in red represent more similar scorings and high values are more distinctive. As shown most of the scorings performed by FIREOS predictors are closer to IREOS scorings than external validation. Although many distances are colored in blue, there are no signs of strong differences between any algorithm of FIREOS and the consensus metric since the maximum number of standard deviations is 1.17. However, random_forest_native clearly acts as a special case here since it shows large and consistent deviations from all other FIREOS predictors. The difference to any external validation method as well as consensus is lower but still aligned with the other FIREOS predictors which again shows the complex dynamics when comparing different outlieriness metrics.	76
4.6	Distance matrix of different evaluation rankings executed on seven outlier solutions for each of the complex/high-noise/low-noise datasets. Lower values in red represent a lower distance between ranks, hence lower MAE. It can be observed that most of the scorings performed by FIREOS predictors are closer to IREOS scorings than external validation. The random_forest_native predictor provides rankings that are close to the opposite of any other method. This plot shows the extreme sensitivity of rank distances in comparison to Z-score distances which can be observed in Figure 4.5].	77

4.7	Standard deviations of Z-Score distance matrices between complex/high-noise/low-noise datasets. External measures tend to have a slightly higher variance toward internal validation. One explanation for that might be that external validation depends on predefined labels which are not known by internal indices. These labels, however, agree sometimes more and sometimes less with the topological properties of the data depending on the instance and therefore appear "less stable" in terms of variance.	78
4.8	Distance matrix of different evaluation methods that are executed on seven outlier solutions for each of the complex/high-noise/low-noise datasets. This chart focuses on the quality of solutions when using sliding windows of different sizes. Suffixes of 0.1 mean 10%, 0.5 represents 50% and 1.0 means that all data samples are considered. Solutions from tree-based classifiers such as decision trees or XgBoost Tree show heavy degradation in quality when applied with sliding windows resulting in less agreement with IREOS as well as other FIREOS predictors for smaller window sizes. Results from linear predictors show a similar picture. Nonlinear support vector machines on the other hand seem to be not affected by this phenomenon as LibSVM results remain very stable over different settings.	79
4.9	Average runtime in seconds of different predictors on basic2d datasets. All results are conducted by using the parallel implementation on 32 threads. As shown in the chart LibSVM is slower than IREOS for smaller problem instances like datasets of the basic2d category.	82
4.10	Average runtime in seconds between the sequential and parallel implementation of different FIREOS predictors on complex, high-noise and low-noise datasets.	83
4.11	Average runtime in seconds of FIREOS classifiers and different window ratios. Values of 0.1 mean 10%, 0.5 represents 50% and 1.0 means that all data samples are considered.	84
4.12	Same chart as Figure 4.11 chart with IREOS runtime as a comparison. Since IREOS does not feature a mechanism for sliding windows, only one bar is displayed.	84

4.13 Runtime experiment on a subset of predictors for different key figures. Those KPIs are rows, columns/features and both. Each chart shows the runtime curves measured in seconds of three FIREOS predictors (LibSVM, Decision Tree Native and XgBoost Tree) as well as the original IREOS over some increasing key figure. For example, the top plot shows the course of execution time when exponentially increasing the total number of rows of a random dataset. The steeper the curve the more computationally expensive the corresponding predictor. All datasets used for this experiment are randomly generated matrices of normally distributed variables. The top plot shows the course of runtime for a variable number of rows (x-axis) and a fixed number of features which is 10. The middle chart shows the runtime behavior for 10 rows but a variable number of features. Finally, the bottom plot shows the course of execution time for both variable rows and features. The x-axis in the last chart represents both rows and features which means that all datasets are quadratic matrices. As shown all three FIREOS predictors are faster than the original IREOS for either KPI except libsvm for very small problems. This chart demonstrates that IREOS is computationally more complex than FIREOS.	85
---	----

List of Tables

3.1	Outlier scores used for the experiments and their scales and normalization into probabilities	34
3.2	Different classifier options in FIREOS and their characteristics	56
3.3	Additional tuning parameters of FIREOS	57
4.1	Runtimes of different FIREOS predictors and IREOS. All results are measured in seconds, use the parallel implementation and are run on 32 threads. . .	82
4.2	Runtime in seconds of different predictors using sequential and parallel FIREOS as well as the corresponding speedup. Presented numbers are averages across all complex/high-noise/low-noise datasets and parallel results were conducted on 32 threads.	83



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [AbdO06] Hany Alashwal, Safaai bin deris, and Razib Othman. One-class support vector machines for protein protein interactions prediction. *Int J Biomed Sci*, 1, 01 2006.
- [AG12] Mennatallah Amer and Markus Goldstein. Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer. 08 2012.
- [AGA13] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. pages 8–15, 08 2013.
- [Agg17] Charu C. Aggarwal. *An Introduction to Outlier Analysis*, pages 1–34. Springer International Publishing, 2017.
- [AHW⁺03] Charu Aggarwal, Jiawei Han, Jianyong Wang, Philip Yu, T. Watson, and Resch Ctr. A framework for clustering evolving data streams. 06 2003.
- [Aji23] Abhinav Ajitsaria. What is the python global interpreter lock (gil)?, 2023. <https://realpython.com/python-gil/> [Accessed: 25-04-2023].
- [AL14] Rami Albatal and Suzanne Little. Empirical exploration of extreme svm-rbf parameter values for visual object classification. In *MultiMedia Modeling*, pages 299–306. Springer International Publishing, 01 2014.
- [AMS⁺21] Md Manjurul Ahsan, M. Mahmud, Pritom Saha, Kishor Datta Gupta, and Zahed Siddique. Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9:52, 07 2021.
- [Ani12] Maria Anisimova. *Parametric models of codon evolution*. 02 2012.
- [AP02] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. volume 2431, pages 15–26, 08 2002.
- [AWS14] Amineh Amini, Teh Wah, and Hadi Saboohi. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, 29:116–141, 01 2014.

- [Ayu21] Collins Ayuya. Parametric versus non-parametric models, 2021. <https://www.section.io/engineering-education/parametric-vs-nonparametric/> [Accessed: 15-02-2023].
- [Bak23] Daniel Baker. Minicore: Fast generic coresets, 2023. <https://github.com/dnbaker/minicore> [Accessed: 22-04-2023].
- [Ban20] Writuparna Banerjee. Train/test complexity and space complexity of logistic regression, 2020. <https://levelup.gitconnected.com/train-test-complexity-and-space-complexity-of-logistic-regress> [Accessed: 15-04-2023].
- [Bar89] H.B. Barlow. Unsupervised learning. *Neural Computation*, 1(3):295–311, 09 1989.
- [BDH⁺19] Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Nate-san Ramamurthy, John Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. Ai fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM Journal of Research and Development*, PP, 09 2019.
- [BG05] Irad Ben-Gal. *Outlier Detection*, pages 131–146. 01 2005.
- [BHPI02] Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. page 250–257, New York, NY, USA, 2002. Association for Computing Machinery.
- [BKNS00] Markus Breunig, Peer Kröger, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000.
- [BL07] Léon Bottou and Chih-Jen Lin. *Support Vector Machine Solvers*, pages 301–320. 01 2007.
- [BL10] Richard G. Brereton and Gavin R. Lloyd. Support vector machines for classification and regression. *Analyst*, 135:230–267, 2010.
- [Bra96] Andrew Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145 – 1159, 11 1996.
- [Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [Bre99] Leo Breiman. Using adaptive bagging to debias regressions. 1999.

- [Bre01] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001.
- [Bro20] Jason Brownlee. Introduction to dimensionality reduction for machine learning, 2020. <https://machinelearningmastery.com/dimensionality-reduction-for-machine-learning/> [Accessed: 18-02-2023].
- [BWXW15] Mei Bai, Xite Wang, Junchang Xin, and Guoren Wang. An efficient algorithm for distributed density-based outlier detection on big data. *Neurocomputing*, 181, 12 2015.
- [CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41, 07 2009.
- [CF03] A.L.M. Chiu and Ada Fu. Enhancements on local outlier detection. pages 298–307, 08 2003.
- [CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.
- [CJ20] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 01 2020.
- [CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [cmg18] MartinThoma (Wikimedia Commons) cmglee. Roc curve, 2018. https://upload.wikimedia.org/wikipedia/commons/1/13/Roc_curve.svg [Accessed: 18-02-2023].
- [Cra09] Nick Craswell. *Precision at n*, pages 2127–2128. Springer US, Boston, MA, 2009.
- [CSAT17] Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. *Outlier Detection with Autoencoder Ensembles*, pages 90–98. 06 2017.
- [CT06] T. Cover and J. Thomas. *Elements of information theory 2nd edition*. 01 2006.
- [CZS⁺16] Guilherme Campos, Arthur Zimek, Joerg Sander, Ricardo Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30, 07 2016.

- [DHP⁺11] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Rich Zemel. Fairness through awareness. *CoRR*, abs/1104.3913, 04 2011.
- [Die00] TG Dietterich. Ensemble methods in machine learning. pages 1–15, 01 2000.
- [DK22] Ali Degirmenci and Omer Karal. Efficient density and cluster based incremental outlier detection in data streams. *Information Sciences*, 607:901–920, 2022.
- [Dri21] Salim Dridi. Unsupervised learning - a systematic literature review, 12 2021.
- [DT14] Sharda Ramesh Dursun Delen and Efraim Turban. *Business Intelligence and Analytics : Systems for Decision Support. Tenth edition.* Global ed. Boston: Pearson, 2014.
- [Dun74] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.
- [DYW19] Hu Ding, Haikuo Yu, and Zixiu Wang. Greedy strategy works for k-center clustering with outliers and coresets construction. In *Embedded Systems and Applications*, 2019.
- [EMS97] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19:476–491, 1997.
- [EPP00] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Adv. Comput. Math.*, 13:1–50, 04 2000.
- [ERKL16] Sarah Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58, 04 2016.
- [EZ19] Kenneth Ezukwoke and Samaneh Zareian. Logistic regression and kernel logistic regression a comparative study of logistic regression and kernel logistic regression for binary classification, 12 2019.
- [Faw06] Tom Fawcett. Introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 06 2006.
- [FCH⁺08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 08 2008.

- [FGK⁺10] Ines Färber, Stephan Günnemann, Hans-Peter Kriegel, Peer Kröger, Emmanuel Müller, Erich Schubert, Thomas Seidl, and Arthur Zimek. On using class-labels in evaluation of clusterings. 01 2010.
- [GCP05] Wu Gang, Edward Y. Chang, and Navneet Panda. Formulating distance functions via the kernel trick. In *Knowledge Discovery and Data Mining*, 2005.
- [GD12] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. 09 2012.
- [GMP⁺20] Kaifeng Gao, Gang Mei, Francesco Piccialli, Salvatore Cuomo, Jingzhi Tu, and Zenan Huo. Julia language in machine learning: Algorithms, applications, and open issues, 03 2020.
- [Goi16] Nicolas Goix. How to evaluate the quality of unsupervised anomaly detection algorithms?, 2016.
- [Gon] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance.
- [Gru69] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.
- [GZSL19] Rongfang Gao, Tiantian Zhang, Shaohua Sun, and Zhanyu Liu. Research and improvement of isolation forest in detection of local anomaly points. *Journal of Physics: Conference Series*, 06 2019.
- [HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 02 1985.
- [HA04] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 10 2004.
- [Has03] Trevor Hastie. Support vector machines, kernel logistic regression, and boosting, 2003. <https://hastie.su.domains/Papers/svmtalk.pdf> [Accessed: 15-04-2023].
- [HC18] David Hand and Peter Christen. A note on using the f-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28:539–547, 05 2018.
- [HCB18] Sahand Hariri, Matias Carrasco, and Robert J. Brunner. Extended isolation forest. *CoRR*, abs/1811.02141, 2018.
- [HK04] Ville Hautamäki and Ismo Kärkkäinen. Outlier detection using k-nearest neighbour graph. 01 2004.

- [HLV03] Wenjie Hu, Yihua Liao, and Rao Vemuri. Robust anomaly detection using support vector machines. *Proceedings of the International Conference on Machine Learning*, 06 2003.
- [HM82] J.A. Hanley and Barbara Mcneil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143:29–36, 05 1982.
- [Ho98] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
- [Hoa13] David C. Hoaglin. Volume 16: How to detect and handle outliers. pages 9–12, 2013.
- [Hol21] Tim Holy. Tutorial on precompilation, 2021. https://julialang.org/blog/2021/01/precompile_tutorial/ [Accessed: 20-05-2023].
- [Hon20] Charmgil Hong. Identification of incorrect data labels using conditional outlier detection. *Journal of Korea Multimedia Society*, 23:915–926, 2020.
- [HPN11] Neminath Hubballi, Bidyut Patra, and Sukumar Nandi. Ndot: Nearest neighbor distance based outlier detection technique. volume 6744, pages 36–42, 06 2011.
- [HPWG20] Sung Ho, Kimberly Phua, Limsoon Wong, and Wilson Goh. Extensions of the external validation for checking learned model interpretability and generalizability. *Patterns*, 1:100–129, 11 2020.
- [HRTZ04] Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 10 2004.
- [HXD03] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster based local outliers. *Pattern Recognition Letters*, 24:1641–1650, 06 2003.
- [IBM23] IBM. What is unsupervised learning?, 2023. <https://www.ibm.com/topics/unsupervised-learning> [Accessed: 18-02-2023].
- [IV18] Tanja Zimek Arthur Iglesias Vázquez, Félix Zseby. Outlier detection based on low density models. *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 970–979, 2018.
- [IVZFZ19] Félix Iglesias Vázquez, Tanja Zseby, Daniel Ferreira, and Arthur Zimek. Mdcgen: Multidimensional dataset generator for clustering. *Journal of Classification*, 36:599–618, 04 2019.

- [JALK16] Surya Mattu Julia Angwin, Jeff Larson and ProPublica Lauren Kirchner. Machine bias, 2016. <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing> [Accessed: 12-02-2023].
- [Joa06] Thorsten Joachims. Training linear svms in linear time. volume 2006, pages 217–226, 08 2006.
- [Joh17] Steven G. Johnson. Introduction to julia: Why are we doing this to you?, 2017. <https://web.mit.edu/18.06/www/Spring17/Julia-intro.pdf> [Accessed: 22-04-2023].
- [Joh23] Steven G. Johnson. Pycall github repository, 2023. <https://github.com/JuliaPy/PyCall.jl> [Accessed: 25-04-2023].
- [JTHW06] Wen Jin, Anthony Tung, Jiawei Han, and Wei Wang. Ranking outliers using symmetric neighborhood relationship. pages 577–593, 04 2006.
- [Kan21] Vijay Kanade. What is fraud detection? definition, types, applications, and best practices, 2021. <https://www.spiceworks.com/it-security/vulnerability-management/articles/what-is-fraud-detection/> [Accessed: 12-02-2023].
- [KC14] Trupti A. Kumbhare and Santosh V. Chobe. An overview of association rule mining algorithms. *International Journal of Computer Science and Information Technologies*, 3(1):927–930, 2014.
- [KDSP02] S. Sathiya Keerthi, Kaibo Duan, Shirish K. Shevade, and Aun Neow Poo. A fast dual algorithm for kernel logistic regression. *Machine Learning*, 61:151–165, 2002.
- [KGP⁺11] Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. Continuous monitoring of distance-based outliers over data streams. In *2011 IEEE 27th International Conference on Data Engineering*, pages 135–146, 2011.
- [Kha21] Renu Khandelwal. Anomaly detection using local outlier factor, 2021. <https://arshren.medium.com/anomaly-detection-using-local-outlier-factor-4e52f16894f> [Accessed: 12-02-2023].
- [KJS17] Bogumił Kamiński, Michał Jakubczyk, and Przemysław Szufel. A framework for sensitivity analysis of decision trees. *Central European Journal of Operations Research*, 26:135 – 159, 2017.
- [KKSZ09] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: Local outlier probabilities. pages 1649–1652, 11 2009.

- [KKSZ11] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. pages 13–24, 04 2011.
- [KN98] Edwin Knorr and Raymond Ng. Algorithms for mining distance-based outliers in large datasets. *VLDB*, 06 1998.
- [KN22] Edwin Knorr and Raymond Ng. Distance-based method, 2022. <https://slideplayer.com/slide/14006457/> [Accessed: 15-02-2023].
- [KSZ08] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 444–452, New York, NY, USA, 2008. Association for Computing Machinery.
- [LB21] Denys Lazarenko and Thomas Bonald. Pairwise adjusted mutual information, 03 2021.
- [Lea23] Scikit Learn. Selecting the number of clusters with silhouette analysis on kmeans clustering, 2023. https://scikit-learn.org/stable/images/sphx_glr_plot_kmeans_silhouette_analysis_002.png [Accessed: 18-02-2023].
- [Ley13] Christophe Klein Olivier Bernard Philippe Licata Laurent Leys, Christophe Ley. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.
- [Li17] Danielle Li. Expertise versus bias in evaluation: Evidence from the nih. *American Economic Journal: Applied Economics*, 9:60–92, 04 2017.
- [Liu08] Kai Zhou Zhi-Hua Liu, Fei Tony Ting. Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.
- [LLP07] Longin Jan Latecki, Aleksandar Lazarevic, and David Pokrajac. Outlier detection with kernel density functions. pages 61–75, 07 2007.
- [LLW07] Hsuan-Tien Lin, Chih-Jen Lin, and Ruby Chiu-Hsing Weng. A note on platt’s probabilistic outputs for support vector machines. *Machine Learning*, 68:267–276, 2007.
- [LLX+10] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. pages 911–916, 12 2010.
- [Mah20] K Mahajan Mahajan. Anomaly detection using isolation forest in python, 2020. <https://blog.paperspace.com/anomaly-detection-isolation-forest/> [Accessed: 15-02-2023].

- [Mar20] Henrique Oliveira Marques. ireos-extension. <https://github.com/homarques/ireos-extension> [Accessed: 18-02-2023], 2020.
- [MCSZ20] Henrique Marques, Ricardo Campello, Jürg Sander, and Arthur Zimek. Internal evaluation of unsupervised outlier detection. *ACM Transactions on Knowledge Discovery from Data*, 14:1–42, 06 2020.
- [MCZS15] Henrique Marques, Ricardo Campello, Arthur Zimek, and Jörg Sander. On the internal evaluation of unsupervised outlier detection. pages 1–12, 06 2015.
- [MD03] Dragos D. Margineantu and Thomas G. Dietterich. *Improved Class Probability Estimates from Decision Tree Models*. 2003.
- [Men96] Scott Menard. Applied logistic regression analysis. 1996.
- [Mis20a] Misc. Scikit-learn documentation, 2020. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html> [Accessed: 15-02-2023].
- [Mis20b] Misc. Scikit-learn documentation, 2020. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.rbf_kernel.html [Accessed: 18-02-2023].
- [Mis22] Misc. Random forests(tm) in xgboost, 2022. <https://xgboost.readthedocs.io/en/stable/tutorials/rf.html> [Accessed: 18-04-2023].
- [Mis23a] Misc. Decisiontree.jl docstrings, 2023. <https://docs.juliahub.com/DecisionTree/pEDeB/0.10.8/autodocs/> [Accessed: 18-04-2023].
- [Mis23b] Misc. The julia programming language, 2023. <https://julialang.org/> [Accessed: 22-04-2023].
- [Mis23c] Misc. The julia programming language: Performance tips, 2023. <https://docs.julialang.org/en/v1/manual/performance-tips/> [Accessed: 24-04-2023].
- [Mis23d] Misc. The julia programming language: Style guide, 2023. <https://docs.julialang.org/en/v1/manual/style-guide/#bang-convention> [Accessed: 24-04-2023].
- [Mis23e] Misc. Liblinear github repository, 2023. <https://github.com/cjlin1/liblinear> [Accessed: 18-04-2023].
- [Mis23f] Misc. Plot a confidence ellipse of a two-dimensional dataset, 2023. https://matplotlib.org/stable/gallery/statistics/confidence_ellipse.html [Accessed: 23-05-2023].

- [Mis23g] Misc. Scikit learn svm documentation: 1.10 decision trees, 2023. <https://scikit-learn.org/stable/modules/tree.html> [Accessed: 17-04-2023].
- [Mis23h] Misc. Tree-based models, 2023. <https://c3.ai/glossary/data-science/tree-based-models/> [Accessed: 17-04-2023].
- [MIVZ22] Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Modeling data with observers. *Intelligent Data Analysis*, 26:785–803, 04 2022.
- [MJS02] Srinivas Mukkamala, Guadalupe Janoski, and Andrew Sung. Intrusion detection using neural networks and support vector machines. volume 2, pages 1702 – 1707, 02 2002.
- [MNPT18] Stratos Mansalis, Eirini Ntoutsi, Nikos Pelekis, and Yannis Theodoridis. An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 11, 06 2018.
- [Mod16] Krishna Modi. Outlier analysis approaches in data mining. *International Journal of Innovative Research in Technology*, 3, 12 2016.
- [Mom13] Mohssen N. Gowayyed Mohammad A. Momtaz, R. Dwof: A robust density-based outlier detection approach. In *Pattern Recognition and Image Analysis*, pages 517–525, 2013.
- [MS03] Markos Markou and Sameer Singh. Novelty detection: A review - part 2:: Neural network based approaches. *Signal Processing*, 83:2499–2521, 12 2003.
- [Mur] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*.
- [Mus19] Sana Mushtaq. Data preprocessing in detail, 2019. <https://developer.ibm.com/articles/data-preprocessing-in-detail/> [Accessed: 05-04-2023].
- [MZCS22] Henrique O. Marques, Arthur Zimek, Ricardo J. G. B. Campello, and Jörg Sander. Similarity-based unsupervised evaluation of outlier detection. In *Similarity Search and Applications*, pages 234–248. Springer International Publishing, 2022.
- [NCZW18] Jin Ning, Leiting Chen, Chuan Zhou, and Yang Wen. Parameter k search strategy in outlier detection. *Pattern Recognition Letters*, 112:56–62, 2018.
- [oST] US National Institute of Standards and Technology. Detection of outliers. <https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm> [Accessed: 12-02-2023].

- [PB19] Nerijus Paulauskas and Algirdas Baskys. Application of histogram-based outlier scores to detect computer network anomalies. *Electronics*, 8:1251, 11 2019.
- [PDR16] José Pasillas-Díaz and Sylvie Ratté. An unsupervised approach for combining scores of outlier detection techniques, based on similarity measures. *Electronic Notes in Theoretical Computer Science*, 329:61–77, 12 2016.
- [PKGf03] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. pages 315–326, 01 2003.
- [Pla00] John Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.*, 10, 06 2000.
- [PN19] Julio Palacio Niño. Evaluation metrics for unsupervised learning algorithms, 05 2019.
- [Pow08] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. 2:37–63, 01 2008.
- [PP12] Ninh Pham and Rasmus Pagh. A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2012.
- [PS91] Gregory Piatetsky-Shapiro. *Discovery, Analysis, and Presentation of Strong Rules*, pages 229–248. 01 1991.
- [PwC20] PwC. Fighting fraud: A never-ending battle, 2020. <https://www.global-screeningsolutions.com/industries/global-economic-crime-and-fraud-survey-2020-1.pdf> [Accessed: 12-02-2023].
- [RBtvV14] S. Romano, J. Bailey, Nguyen the vinh, and Karin Verspoor. Standardized mutual information for clustering comparisons: One step further in adjustment for chance. *31st International Conference on Machine Learning, ICML 2014*, 4:2873–2882, 01 2014.
- [Res22] Straits Research. Fraud detection and prevention market size is projected to reach usd 190 billion by 2030, growing at a cagr of 23.2%, 2022. <https://www.globenewswire.com/news-release/2022/07/25/2485302/0/en/Fraud-Detection-and-Prevention-Market-Size-is-projected-to-reach-US.html> [Accessed: 12-02-2023].

- [RKV⁺20] Lukas Ruff, Jacob R. Kauffmann, Robert A. Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109:756–795, 2020.
- [RLF⁺06] Fabrice Rossi, Amaury Lendasse, Damien François, Vincent Wertz, and Michel Verleysen. Mutual information for the selection of relevant variables in spectrometric nonlinear modelling. 80(2):215–226, 2006.
- [Rou87] Peter Rousseeuw. Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *comput. appl. math.* 20, 53–65. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987.
- [RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. volume 29, pages 427–438, 06 2000.
- [SAA20] Intisar Shadeded, Jwan Alwan, and Dhafar Abd. The effect of gamma value on support vector machine performance with different kernels. *International Journal of Electrical and Computer Engineering (IJECE)*, 10:5497, 10 2020.
- [Sal17] Chapter 2 - a taxonomy of example-based super resolution. In *Example-Based Super Resolution*, pages 15–29. Academic Press, 2017.
- [Sas07] Yutaka Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 01 2007.
- [SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
- [Sch00] Bernhard Schölkopf. The kernel trick for distances. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, page 283–289. MIT Press, 2000.
- [SG02] Alexander Strehl and Joydeep Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 01 2002.
- [Sil20] Rosaria Silipo. Ensemble models: Bagging boosting, 2020. <https://medium.com/analytics-vidhya/ensemble-models-bagging-boosting-c33706db0b0b> [Accessed: 25-04-2023].

- [Sim22] Milos Simic. Decisiontree.jl docstrings, 2022. <https://www.baeldung.com/cs/gradient-boosting-trees-vs-random-forests> [Accessed: 18-04-2023].
- [SJS06] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. volume Vol. 4304, pages 1015–1021, 01 2006.
- [SkL23] SkLearn. Isolationforest documentation, 2023. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html#sklearn.ensemble.IsolationForest> [Accessed: 10-04-2023].
- [SLN18] Habiba Sani, Ci Lei, and Daniel Neagu. *Computational Complexity Analysis of Decision Tree Algorithms: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11–13, 2018, Proceedings*, pages 191–197. 11 2018.
- [Smo22] Hrvoje Smolic. What is unsupervised learning?, 2022. <https://graphite-note.com/machine-learning-unsupervised-3-main-tasks> [Accessed: 18-02-2023].
- [SO17] Gulcan Sarp and Mehmet Ozcelik. Water body extraction and change detection using time series: A case study of lake burdur, turkey. *Journal of Taibah University for Science*, 11(3):381–391, 2017.
- [Sre07] Nathan Srebro. How good is a kernel when used as a similarity measure? pages 323–335, 06 2007.
- [SSB⁺97] B. Schölkopf, Kah-Kay Sung, Christopher Burges, Federico Girosi, Partha Niyogi, Tomaso Poggio, and Vladimir Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *Signal Processing, IEEE Transactions on*, 45:2758 – 2765, 12 1997.
- [SSB18] Bernhard Schölkopf, Alexander J. Smola, and Francis Bach. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2018.
- [Sta17] Artur Starczewski. A new validity index for crisp clusters. *Pattern Analysis and Applications*, 20:687–700, 08 2017.
- [Ste20] Doug Steen. Understanding the roc curve and auc, 2020. <https://towardsdatascience.com/understanding-the-roc-curve-and-auc-dd4f9a192ecb> [Accessed: 18-02-2023].

- [SU12] Karanjit Singh and Shuchita Upadhyaya. Outlier detection: Applications and techniques. *International Journal of Computer Science Issues*, 9, 01 2012.
- [Sut05] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobbs's Journal*, 30(3):202–210, 2005.
- [SWS⁺99] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. Cambridge, MA, USA, 1999. MIT Press.
- [TCFC02] Jian Tang, Zhixiang Chen, Ada Fu, and David Cheung. Enhancing effectiveness of outlier detections for low density patterns. pages 535–548, 05 2002.
- [Tob22] Adrian Tobisch. pyireos: An implementation of ireos in python, 2022. <https://github.com/ExoFlare/pyIREOS> [Accessed: 22-04-2023].
- [Tok22] A. Aylin Tokuç. k-nearest neighbors and high dimensional data, 2022. <https://www.baeldung.com/cs/k-nearest-neighbors> [Accessed: 15-02-2023].
- [Unk22] Unknown. Roc curves and auc for models used for binary classification, 2022. <https://data.library.virginia.edu/roc-curves-and-auc-for-models-used-for-binary-classification/> [Accessed: 18-02-2023].
- [Val21] Dilip Valeti. Detect and remove the outliers in a dataset, 2021. <https://medium.com/@dilip.voleti/detect-and-remove-the-outliers-in-a-dataset-1398f4cc7b44> [Accessed: 12-02-2023].
- [VdMPVDH09] L. J. P. Van der Maaten, E. O. Postma, and H. J. Van Den Herik. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, pages 1–41, 2009.
- [VEB09] Nguyen Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? page 135, 06 2009.
- [V.N64] A.Ya. Chervonenkis V.N.Vapnik. A class of algorithms for pattern recognition learning. *Avtomat. i Telemekh.*, 25, 1964.
- [VTS04] J.P. Vert, Koji Tsuda, and Bernhard Schölkopf. A primer on kernel methods. *Kernel Methods in Computational Biology*, 35-70 (2004), pages 35–70, 01 2004.

- [WBH⁺02] Graham Williams, Rohan Baxter, Hongxing He, Simon Hawkins, and Lifang Gu. A comparative study of rnn for outlier detection in data mining. pages 709 – 712, 02 2002.
- [WBH19] Hongzhi Wang, Mohamed Bah, and Mohamed Hammad. Progress in outlier detection techniques: A survey. *IEEE Access*, 7, 08 2019.
- [WC10] the free media repository Wikimedia Commons. Lof, 2010. <https://commons.wikimedia.org/wiki/File:LOF.svg> [Accessed: 25-05-2023].
- [WLW04] Ting-fan Wu, Chih-Jen Lin, and Ruby Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, pages 975–1005, 02 2004.
- [WT20] Duo Wang and Toshihisa Tanaka. Robust kernel principal component analysis with 2,1-regularized loss minimization. *IEEE Access*, 8:81864–81875, 2020.
- [WYT07] Defeng Wang, Daniel Yeung, and E.C.C. Tsang. Structured one-class classification. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 36:1283–95, 01 2007.
- [YCR17] Yizhou Yan, Lei Cao, and Elke Rundensteiner. Scalable top-n local outlier detection. pages 1235–1244, 08 2017.
- [ZE01] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. *ICML*, 1:609–616, 05 2001.
- [Zha09] Ethan Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009.
- [ZHJ09] Ke Zhang, Marcus Hutter, and Huidong Jin. A new local distance-based outlier detection approach for scattered real-world data. volume 5476, 03 2009.
- [ZNL19] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection, 2019.
- [ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. 25(2):103–114, 1996.