Rule Extraction in Unsupervised Anomaly Detection for Model Explainability: Application to OneClass SVM

Alberto Barbado[1,2], **Oscar Corcho**[2]
alberto.barbadogonzalez@telefonica.com, ocorcho@fi.upm.es
[1] Telefónica, 28050 Madrid, Spain
[2] Universidad Politécnica de Madrid, Departamento de Inteligencia Artificial, 28660 Boadilla del Monte, Spain

Abstract

OneClass SVM is a popular method for unsupervised anomaly detection. As many other methods, it suffers from the *black box* problem: it is difficult to justify, in an intuitive and simple manner, why the decision frontier is identifying data points as anomalous or non anomalous. Such type of problem is being widely addressed for supervised models. However, it is still an uncharted area for unsupervised learning. In this paper, we evaluate some of the most important rule extraction techniques over OneClass SVM models, as well as presenting alternative designs for some of those XAI algorithms. Together with that, we propose algorithms to compute metrics related with XAI regarding the "comprehensivility", "representativeness", "stability" and "diversity" of the rules extracted. We evaluate our proposals with different datasets, including real-world data coming from industry. With this, our proposal contributes to extend Explainable AI techniques to unsupervised machine learning models.

Keywords XAI, OneClass SVM, unsupervised learning, rule extraction, anomaly detection, metrics

Introduction

Responsible Artificial Intelligence [1] is an emerging discipline that is gaining relevance both in academic and industrial research. An increasing number of organisations are defining policies and criteria for the usage of data and the development of support decision systems using AI techniques.

For example, Telefónica has defined its own AI principles [2], which can be organised into the following categories:

- Detecting sensitive data in the datasets used to train Machine Learning (ML) models (for example, the use of gender information to support the decision of giving a credit score) or detecting whether there is a bias in the data, what may have an unwanted effect in decision making. Besides analysing directly the dataset, there are also methods to perform an evaluation on a trained model to check whether there is any bias on its decisions. This is done using a group of metrics known as fairness metrics [3, 4, 5].
- Explaining how an algorithm reaches a conclusion in a way that is clear and intuitive for a human being. This is crucial not only to avoid the fear of considering AI as black boxes that may suddenly take harmful decisions in the future, but also to contribute to the democratisation of AI and increase trust on these systems.

The first group of challenges is being widely addressed by researchers, with tools to audit datasets and trained models to detect risks, and at the same time provide solutions to mitigate those biases [6, 7]. The second group is addressed through the use of Explainable AI (XAI) techniques, that can be applied to black-box models in order to obtain post-hoc explanations based on information provided by it. In the literature, there are many XAI proposals for supervised ML models. However, some of the most recent and thorough reviews on XAI [8, 9, 1, 10] do not mention the application of such techniques to unsupervised learning.

Among the different applications of unsupervised learning, one example is outlier detection. Many times, there is no prior information about the outliers, so, for those cases, unsupervised ML algorithms offer the chance to infer patterns and detect anomalies. However, it is important to not only detect outliers, but be able to explain both why a particular datapoint has been labelled anomalous, and how the model behaves on a general level. This is something that is resolved with XAI techniques like rule extraction. This family of techniques can explain with an "IF...THEN" schema both the output of a particular datapoint or the global behaviour of the original model. In the case of outlier detection, they can explain both a particular outlier and also how the features of the whole model contribute to identify points as outliers or inliers. There is recent literature deals with XAI applied over anomaly detection systems [11, 12]. However, to the best of our knowledge, there are no evaluations of rule extraction techniques for unsupervised ML outlier detection.

Many outlier detection systems may be more interested in explaining faithfully why a datapoint is an outlier, and what should have happened in order to be an inlier, rather than being able to cover all possible scenarios with explanations that might be wrong. For instance, at Telefónica, in a Call Center application where we explain why the num-

ber of total calls at one day is anomalous, we may prefer to explain some days with total confidence, even if that means not being able to explain all instances. Applying this over a rule extraction technique scenario, it means that the rules extracted need to be rules with a 100% precision (P@1); rules that classify only datapoints from one class (p.e. "outliers") without including datapoints from the other one. This is important because, if the rules extracted cover inliers, then the counterfactual explanation for an outlier should always lead to a scenario where it is completely sure that the point mentioned would be classified as an inlier.

Another open issue in XAI is how to evaluate the quality of explanations. The literature [13, 10] mention concepts such as:

- "Comprehensivility": Are the explanations easy enough to understand?
- "Representativeness": Are the explanations relevant? Do they explain all the possible cases?
- "Stability": Do the explanations match the predictions of the model? Or are there inconsistencies?
- "Diversity": Are the explanations different enough? Or are they redundant?

These concepts are addressed in the literature, but to the best of our knowledge there are not many algorithmic implementations of them, and there are no empirical evaluations for rule extraction techniques.

Following this, our main contributions are:

- Apply rule extraction techniques for unsupervised outlier detection models, using OCSVM as an example algorithm. The techniques evaluated are model-agnostic, so in order to offer a more general overview, we will analyse the results using two different kernels: Radial Basis Function (RBF) and a Linear one.
- Within the rule extraction techniques considered, we will also design and implement some alternatives over one of the algorithms from the literature. We will also propose an algorithm to turn a local rule extraction model (Anchors) into a global one.
- Quantify the quality of the explanations generated using XAI metrics that measure "comprehensivility", "representativeness", "stability" and "diversity" in a rule extraction technique scenario. We propose both how to implement each one of them, and then we will obtain their results in our empirical evaluation.
- Considering only P@1 rules, see the different results obtained and compare them using the metrics of the previous point. For this, we will also group all metrics into one single function.
- Also, since RBF kernels group inliers within one or more hyperspheres that separates them from outliers, we will also analyse if this process contributes to the rules extracted in this P@1 scenario. This could be true if the number of rules obtained for inliers using a RBF kernel are inferior than those obtained using a linear one.

The empirical evaluations carried out use both open datasets as well as real data from Telefónica. Our evaluation consists in analysing the results for the aforementioned metrics using different rule extraction algorithms over OCSVM models with the two kernel configurations already mentioned.

The rest of the paper is organized as follows. First, we describe some related work in the area of XAI and rule extraction applied to SVM. This first chapter will also introduce the rule extraction techniques considered, as well as literature related to XAI metrics. After identifying research opportunities derived from these works, the paper introduces some alternatives over one of the rule extraction techniques, as well as algorithms to compute the metrics described in the first chapter. Following this, we present an empirical evaluation of our algorithm with several datasets. We then conclude, showing also potential future research lines of work.

Related Work

This section reviews unsupervised ML models used for anomaly detection, and reviews previous work on rule extraction in SVM that is relevant for our proposal.

Unsupervised ML for Anomaly Detection

Many algorithms for unsupervised anomaly detection exist. Examples are IsolationForest [14], Local anomaly Factor (LOF) [15] and OCSVM [16]. The latter has relevant advantages over the former ones, mainly in terms of computational performance. This is due to the fact that it creates a decision frontier using only the support vectors (like general supervised SVM) and that model training always leads to the same solution because the optimization problem is a convex one. However, SVM (hence OCSVM) algorithms are some of the most difficult ones to explain due to the mathematically complex method that obtains the decision frontier.

SVM for classification theoretically maps the data points available in the dataset to a higher dimensional space than the one determined by their features, so that the separation among classes may be done linearly. It uses a hyperplane obtained from data points from all of the classes. These data points, known as support vectors, are the ones that are closer to each other and the only ones needed to determine the decision frontier. However, it is not really necessary to map to a higher dimension due to the fact that the equation that appears in the optimization of the algorithm uses a dot product of those mapped points. Because of that, the only thing to be calculated is such dot product, something that can be accomplished with the well-known kernel trick. Hence instead of calculating explicitly the mapping to a higher dimension the equation is solved using a kernel function.

In OCSVM there are no labels. Hence all data points are considered to belong to a same class at the beginning. The decision frontier is computed trying to separate the region of the hyperspace with a higher number of data points close to each other from another that has small density, considering those points as anomalies. To do so the algorithm tries to define a decision frontier that maximizes the distance to the origin of the hyperspace and that at the same time separates

from it the maximum number of data points. This compromise between those factors leads to the optimization of the algorithm and allows obtaining the optimal decision frontier. Those data points that are separated are labeled as non-anomalous (+1) and the others are labeled as anomalous (-1).

The optimization problem is reflected in the following equations:

$$\begin{aligned} min_{w,\xi_i,\rho} &= \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}(\xi_i - \rho) \\ &\text{subject to:} \\ (w, \phi(x_i)) &\geq \rho - \xi_i \;\; for\; i = 1, ..., n \\ \xi_i &\geq 0 \;\; for\; i = 1, ..., n \end{aligned} \tag{1}$$

In that equation, ν is a hyper-parameter known as *rejection rate*, which needs to be selected by the user. It sets an upper bound on the fraction of anomalies that can be considered, and also defines a lower bound on the fraction of support vectors that can be considered. Using Lagrange techniques, the decision frontier obtained is the following one:

$$\begin{aligned} f(x) &= sgn((w, \phi(x_i) - \rho) \Rightarrow \\ f(x) &= sgn(\sum_{i=i}^{n} \alpha_i K(x_i, x) - \rho) \end{aligned} \tag{2}$$

Hence the hyper-parameters that must be defined in this method are the rejection rate, ν, and the type of kernel used.

Rule Extraction in SVM

Several papers deal with the importance of XAI applied to models such as SVM. In particular [17] aims to resolve the black box problem in SVM for supervised classification tasks. It obtains a set of rules that explain in a simple manner the boundaries that contain the values of the different classes. Thanks to that, it is easier to understand what are the conditions that will identify a data point as belonging to one class or to another (in OCSVM it would be belonging to class +1 - normal data - or class -1 - anomaly -). The challenge consists in discovering a way to map the algorithm results to a particular set of rules. There are two general ways to do it. One consists in inferring the rules directly from the decision frontier, using a decompositional rule extraction technique. The other (simpler) one uses a method called pedagogical rule extraction technique that does not care about the decision frontier itself and considers the algorithm as a black box from where to extract those rules depending on which class it uses to classify different relevant data points used as an input.

The first method is clearly more transparent, as it deals directly with the inner structure of the model. However, it is generally more difficult to implement. One way to implement it is with a technique known as SVM+ Prototypes [18], which consists in finding hypercubes using the centroids (or prototypes) of data points of each class and using as vertices the data points from that hyperspace area farther away from that centroid (or use directly the support vectors themselves if they are available). It will then infer a rule from the values of the vertices of the hypercube that contain the limits of all the points inside it, creating one rule for each hypercube.

For example, a dataset that contains two numerical features X and Y will be defined in a 2-dimensional space. The algorithm will create a square that contains the data points on each of the classes, as shown in Figure 1. The rule that justifies that a data point belongs to class 2 is:

- Rule 1: CLASS 2 IF X$\geq$X1 $\wedge$ Y$\geq$Y1 $\wedge$ X$\leq$X2 $\wedge$ Y$\leq$Y2

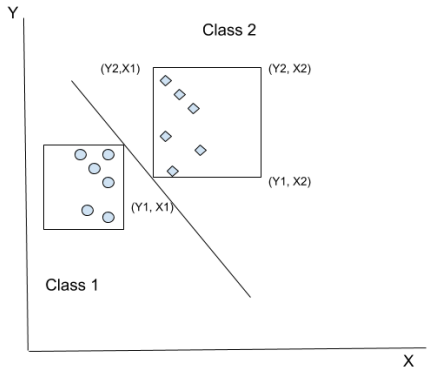

Figure 1: SVM with linear kernel classifying data points of two classes.

The generated hypercubes may wrongly include points from the other class when the decision frontier is not linear or spherical, as shown in Figure 2. In this case the algorithm considers an additional number of clusters trying to include the points into a smaller hypercube, as shown in Figure 3.

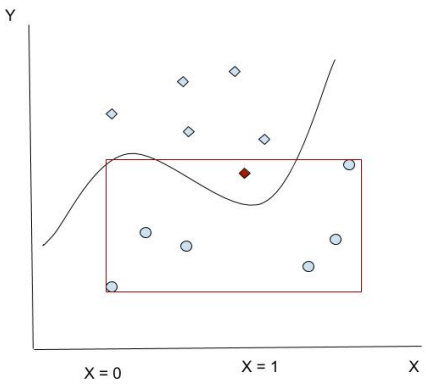

Figure 2: A hypercube generated using the farthest points leads to the wrong inclusion of data from the another class.

A rule will be generated for each hypercube, considering all those scenarios as independent, leading to this output:

- Group 1: CLASS 1 IF X...
- Group 2: CLASS 1 IF X...

There are some downsides of that method in supervised classification tasks, especially when the problem is not simply a binary classification or when the algorithm is performing a regression. For instance, the number of rules may grow

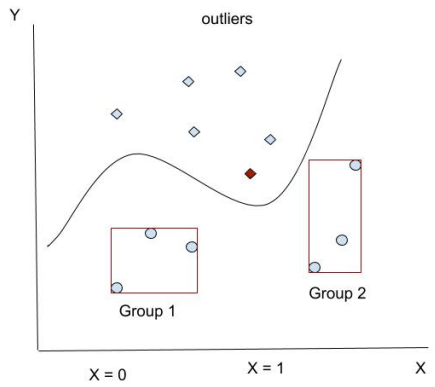

Figure 3: Using more hypercubes avoids the aforementioned problem. Now there is no wrong inclusion of data points from another class.

immensely due to the fact that a set of rules will be generated for each category and each set may contain a huge number of rule groups, leading to an incomprehensible output.

However, in OCSVM these difficulties may be potentially mitigated due to two reasons. On the one hand, the explanations are reduced to rules that explain when a data point is not an anomaly (so there would be no need to define rules for the anomalies). On the other hand, the algorithm tries to group all non-anomalous points together, setting them apart from the outliers. Because of this, the chance to define a hypercube that does not contain a point from the another class may be higher than in a standard classification task. Both the unbalanced inherent nature of data points in anomaly detection (few anomalies vs. many more non-anomalous data points) and the fact that non-anomalous points tend to be closer to each other may help achieving good results with this method.

Rule extraction techniques in XAI

The literature contains many rule extraction proposals that can contribute to XAI. The algorithms that provide these rule can be also classified according to XAI taxonomies, such as [1]. Among all the proposals, for this paper we are interested in rule extraction techniques that can be applied over an unsupervised machine learning model in order to generate explanations that explain how the whole model behaved. Thus, we analyse some model-agnostic post-hoc global techniques that, using as input the same dataset than the unsupervised model, and using as target the output of that model (indicating if a datapoint is an inlier or outlier), can generate rules that explain how the original model is behaving. We will also cover some references of model-agnostic post-hoc local techniques that provide rules to explain particular output instances.

As mentioned before, a general way to approximate any blackbox model globally is by using a surrogate supervised decision model trained over the same dataset, but instead of using the real labels (the ones used for the blackbox model), it is trained over the predictions of that blackbox model [10]. This could be accomplished with any ML model, but it is useful to do it with a whitebox model that can be directly interpreted. Among these whitebox models, some of the could be used for rule extraction. AN example is a Decision Tree (DT) model. DT allows to explain the classification logic of the blackbox model through the usage of rules, that can be used even for classifying new instances. The advantages of using a DT as a surrogate global model is its flexibility (it can be applied over any model in an agnostic way) and simplicity (it's a solution easy to explain). However, this approximation at the end leads to explain a proxy model, and not the actual data, since the surrogate model never sees the true target values.

Anchors [19] is a model-agnostic XAI technique that extracts rule explanations for individual datapoints. The purpose of Anchors is finding a decision rule that approximates the decision function of the black-box model around that individual datapoint. This rule "anchors" the prediction of that datapoint, so that any perturbation of the features of that point that are still inside the rule will always return the same output from the black-box model. The approach is as follows. First, the algorithm generates candidate rules that could explain the datapoint. Then, it evaluates those candidate rules. In order to do that, Anchors generates permutations around the datapoint (similar datapoints to that original one) that yield the same result. The result is evaluated by calling the black-box model (the oracle) and obtaining the classification for that datapoint. In order to optimize the exploration-exploitation of generating and evaluating datapoints, it uses a reinforcement learning approach with a Multi-Armed Bandit (MAB) approximation. In this MAB, each arm of the Bandit problem is a candidate rule, and the datapoints generated, after obtaining their classification result from the black-box model, are used to compute a precision metric used to evaluate the candidate rule's payoff. This reinforcement learning approach helps minimizing the number of calls to the model in order to reduce the computational cost of the algorithm. Among all the candidate rules, the algorithm then checks if the best one of them matches a predefined convergence criteria. To do that, it filters rules according to a precision threshold, and selects form the remaining ones the one with highest coverage. That rule is used to explain that original datapoint. If there are no rules that match the convergence criteria, then the algorithm keeps iterating (using a beam search approach) using the B best rules from the previous step in order to generate new candidate rules for the following one. In those following steps, Anchors keep extending the rules with more features (in the first step, it only uses one feature per candidate rule). Thus, Anchors offers a model-agnostic approach that generate IF-THEN rules, easy to interpret, that are generated in an efficient way thanks to the usage of reinforcement learning (MAB) that can be parallelized. However, Anchors is very sensitive to its initial configuration, like many permutation approach algorithm, such as LIME [20]. Other important consideration of Anchors is that, while it keeps the calls to the oracle to a minimum (thanks to MAB), it still requires a lot of calls, and that can affect the runtime of the algorithm.

RuleFit [21] is a model-agnostic surrogate model that

learns a linear regression model (Lasso) that uses as features both the original features of the model, as well as new generated features that represent decision rules. In order to accomplish that, first, a tree model is trained over the output and the input features, and the decision paths between the tree levels are turned into decision rules, except for the ones that lead to the leaf nodes, which are not considered. These rules are used as additional features, along with the original ones, on the Lasso surrogate model. Thanks to this, RuleFit yields both rules as well as their contribution, measured through the coefficients of the Lasso model. In summary, RuleFit generates a white-box model that includes rules as features, that can be interpreted as a standard linear regression one. The only caveat is that, for the original coefficients, the predicted outcome changes by $|\beta_j$ if feature x_j changes by one unit if the other features remain unchanged, while for a feature-rule r_k it is different; if all the conditions of the feature r_k are met, the predicted outcome changes by α_k (the weight associated to that rule-coefficient) for regression. Similarly, for classification tasks, when the conditions of r_k are met, the odds for event vs. no-event changes by a factor of α_k.

Similarly to RuleFit, SkopeRules [10] is another way to generate rules from tree ensembling techniques. They differ, however, in how they obtain the rules. First, SkopeRules generates the rules using surrogate tree ensembles trained using the input features and the target variable. Then, it applies a filtering step in which, using a threshold for Precision and Recall, some rules are removed and some are kept. This step allows to select only high-performing rules, and removing the ones that do not yield good results. The last step is known as "semantic rule duplication". This step eliminates duplicate rules (rules that are the same or very similar to other ones). It also eliminates again low-performing rules based on their results for a F1-metric. This allows to obtain high-performing as well as heterogeneous rules. The final set of rules is the output of SkopeRules, differing from RuleFit because it does not use a Lasso model to aggregate all rules.

Falling Rule Lists (FRL) [22] are classification models that generate a sorted list of IF-THEN rules, thus, they can serve as a model-agnostic global post-hoc rule extraction technique. The rules are binary, and are looked one after the other, in order to see if a particular datapoint can be classified into one of the classes. The rules are sorted according to the probability of classifying a datapoint into that class using that rule. Due to that, FRL offers a list of IF-ELSE IF rules associated to a particular class with a decreasing probability score. This is inspired in the concept of healthcare triage: patients are classified within risk level groups, and the highest-risk ones should be considered first. The particular algorithm cited and used in this paper uses an approach for learning based on a Bayesian framework, instead of a greedy decision tree learning method, named Bayesian Falling Rule Lists (BFRL).

Boolean Decision Rules via Column Generation (BRCG) [23] also provides a binary classifier by using disjuntive normal form (DNF, OR-of-ANDs) or conjuntive normal form (CNF, AND-of-ORs) through interpretable rules. In case of DNF (the one used in this paper), they provide an unordered set of decision rules that classify a datapoint into the positive category if at least one of the rules is satisfied. This is different than other methods already mentioned, such as BFRL where the rules are ordered in an IF-THEN schema, or the surrogate DT model, that provides the rules in a tree structure schema. In this article, we use the BRCG-light approximation from [24], that replaces the integer programming solver used in the original paper by a heuristic beam search one.

Generalized Linear Rule Models (GLRM) [25] generate decision rules and combines within a linear model (generalized additive model, GAM). Thus, they provide both a non-linear modelling thanks to the decision rules, while keeping the interpretability by using a linear model that ensembles them. However, as [26] notice, while it is feasable to interpret linear combination of rules, if the number of rules increases too much, there is a risk of losing the interpretability of the model. The authors of the original paper highlight that in order to reduce the rules generated and not lose interpretability, they use a rule selection technique based on column generation (CG). CG searches the spaces of rules and generates them only when they are needed, and then fits again the GLM model. This allows to analyse again old rules, reweight them, and discard the ones that are not needed anymore. This is different to other methods used in the literature, mainly pre-selecting a subset of candidate rules using optimization techniques, or a greedy optimization approach by adding rules one by one using sequential covering or boosting techniques.

XAI for anomaly detection

There are already existing references in the literature that highlight the importance of combining outlier detection with the generation of insights that provide explanations about the decision process. One example is the work of [11]. In that paper, the authors cover different aspects related to outlier detection. First, they reflect on the definition of "outlier" and the different possibilities to deal with them. Citing [27], an outlier is "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data". Here, they indicate that the important part of the sentence is "appears to", as it opens to the analysis of whether a datapoint is truly an outlier. This, combined to [28], defines a taxonomy of two levels: first, identifying "discordant" datapoints with the rest, and then, classifying them as "true outliers" (named "contaminants") or false positives. However, the recognize that solving this two-steps procedure strongly depends on domain knowledge for most cases. The second part of the article defines a taxonomy for the different outlier detection methods:

- Global vs local outliers: Outlier detection methods can use either the whole dataset to identify outliers, or only a fraction of it.
- Point anomalies, collective anomalies, and contextual anomalies: The first type refers to the detection of outliers at an individual level. The other two cases identify outliers inside a specific context.
- Labelling vs scoring:

- Supervised vs unsupervised: outlier detection methods can consider prior knowledge about outliers (that, for ML methods can be used as labels to train the model), or they may work without any prior knowledge at all about outliers.
- Parametric vs non-parametric: Parametric methods make assumptions about the underlying data distribution (mainly, that the data distribution is a Gaussian), and they adjust the dataset to that distribution in order to see what datapoints are outliers. In contrast, non-parametric methods do not make any assumption about the underlying distribution. However, non-parametric methods still need assumptions about their parameter's configuration.

The last part of the article, after analysing some of the different algorithms available, present as a new and recent field XAI applied to outlier detection, indicating some previous works that deal with it. Among our research, we only find [12] as a paper that explicitly deals with XAI applied to OCSVM models for outlier detection. They propose a model-specific method based on the concept that OCSVM models can be rewritten as pooling neural networks. Due to the asymmetry between inliers and outliers, they model with a min-pooling over distances for outliers, and a max-pooling over similarities for inliers. Thanks to turning OCSVM models to a neural network, they apply a deep Taylor decomposition (DTD) to obtain explanations in term of input features. DTD serves as a framework to apply layer-wise retropropagation (LRP) in order to obtain the feature contribution of the input features to a predicted output. The authors extend the explanations generated to include using both input features or support vectors.

The previous analysis of the literature show that there is still an open research area to analyse model-agnostic rule extraction techniques over unsupervised outlier detection models, particularly for OCSVM.

Metrics for XAI

Beyond indicating the importance of both detecting oultiers and being able to explain how the decision took place, it is also crucial to quantify the quality of those explanations. There are some recent reviews in the literature that deal with the challenge of metrics in XAI, such as in [13]. In that article, the authors analyse the literature and define a taxonomy of properties that should be considered in the individual expalnatiosn generated by XAI techniques.

- Accuracy: It is realed to the usage of the explanations to predict the output using usneen data by the model.
- Fidelity: It refers to how well does the explanations approximate the underlying model. The explanations will have high fidelity if their predictions are constantly simmilar to the ones obtained by the blackbox model. Generally, if the explanations have high consistency and the model has high accuracy, the explanations will also have high accuracy.
- Consistency: It refers to the similarity of the explanations obtained over two different models trained over the same input dataset. High consistency appears if the explanatiosn obtained from the two models are similar. However, a low consistency may not be a bad result since they models could be extracting different valid patterns from the same dataset due to the "Rashomon Effect".
- Stability: It measures how similar are the explanations obtaiend for similar datapoints. Opposed to consistency, stability measures the similarity of explanations using the same underlying model.
- Comprehensibility: This metric is related to how well will a human understand the explanation. Due to this, it is a very difficult metric to define mathematically, since it is affected by many subjective elements related to human's perception (such as context, background, prior knowledge...). However, there are some objective elements that can be considered in order to measure "comprehensibility", such as if the explanations are based on the original features (or if they are based on synthetic ones generated after them), the length of the explanations (how many features does they include), or how many explanations are considered. In general terms, using the original features, while keeping the number of explanations generated and the features used to a minimum, will increase the comprehensibility.
- Certainty: It refers to whether the explanations include the certainty of the model about the prediction or not.
- Importance: Some XAI method that use features for their explanations include a weight associated with the relative importance of each of those features.
- Novelty: Some explanations may include if the datapoint to be explained comes from a region of the feature space that is far away from the distribution of the training data. This is something important to consider in many cases, since the explanation may nor be reliable due to the fact that the datapoint to be explained is very different from the ones used to generate the explanations.
- Representativeness: It measures how many instances are covered by the explanation. Explanations can go from explaining a whole model (p.e. weights in linear regression) to only be able to explain one datapoint.

As already mentioned, this metric deal with quantifying the quality of the explanations for an individual datapoint. However, they are also perfectly applicable for rule extraction techniques in which the outputs (rules) for the whole dataset can also be analysed in those terms. In this context, one additional aspect to consider is "diversity", a metric that indicates whether the explanations are redundant or repetitive and can already be mostly covered by another explanation, or if they provide insights that are not deducible from the other explanations available.

For our research regarding rule extraction, we will focus in analysing the degree of "comprehensibility" of the rules, the coverage of those rules of the datapoints available ("representativeness"), if the rules approximate the underlying model ("stability"), and if they have overlapping between them and are redundant ("diversity"). The challenge here is

quantifying those metrics. Due to that, we will propose and use algorithms to quantify them in a rule extraction scenario.

Method

We first describe the intuition behind our rule extraction approach fom an OCSVM model for anomaly detection. Then, we describe in detail the algorithm implementation.

Algorithm Intuition

We propose using rule extraction techniques within OCSVM models for anomaly detection, by generating hypercubes that encapsulate the non-anomalous data points, and using their vertices as rules that explain when a data point is considered non-anomalous. As already mentioned in the introduction, [18] proposes an algorithm to extract rules from a SVM model by performing clustering over the datapoints that belong to one of the classes. The clustered datapoints will be used to obtain a geometric surface that enclose the rest of the datapoints inside. There are two ways to accomplish it: building hypercubes or building hyperspheres. This paper will focus the analysis over the first approach: building hypercubes. For this scenario, the algorithm obtains the furthermost datapoints from inside the cluster that will be used as vertices in a hypercube, so they enclose the rest of datapoints of that category inside. In case that the hypercube generated encloses points from the other category, then the number of clusters will be increased, aiming to obtain smaller cubes that could fit the data without including points from the other class. This is done in iteratively until no points from the other class are inside the hypercubes, or a maximum number of predefined iterations is reached. During the process, if a hypercube does not contain points from the other class, then that hypercube is translated into a rule, and those datapoints are removed from the following iteration steps.

Images 4 and 5 shows an example application of this algorithm for a 2D space. In image 4 appears the initial scenario, where the first step in the iteration process consists in applying one cluster over the dataset for datatpoints of one of the classes (blue ones). However, with one cluster, the 2D square that enclose the datapoints contains points from the other class, so more clusters need to be applied. As 5 shows, iteration 3 (with 3 clusters) is the first one with squares without red points, so those subspaces are turned into rules and the points inside them removed from the iteration process, that starts again with one cluster for the remaining datapoints. Iteration 6 will be the last one, and 5 rules have been extracted up to that point.

The approximation proposed before is not the only one that can be applied in order to extract the rules. Image 6 shows one of our alternative proposals over [18] method. Instead of removing datapoints that are inside a rule without points from the other class, the process always keeps all datapoints in every iteration since there could be clustering patters that could only be found if all points are together. In this approach, the number of clusters is constantly increased until no datapoints from the other class are inside the hypercubes, or the maximum number of iterations is reached. We

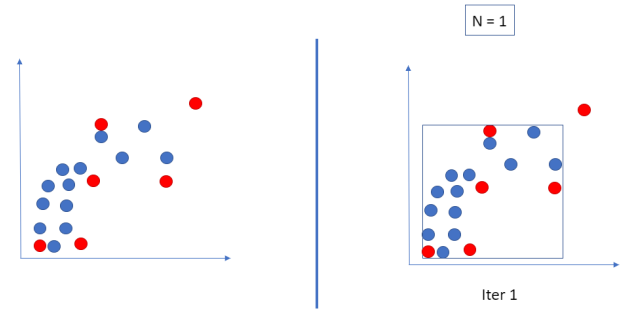

Figure 4: Clustering over a 2D space. With one cluster over datapoints from one class (blue), there are still others from the other class (red) inside the square.

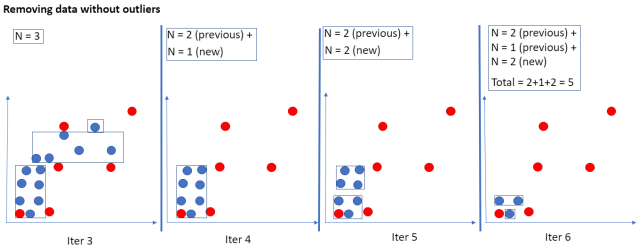

Figure 5: Applying the proposal of [18], the number of clusters keeps increasing until no points from the other class are inside, an then that hypercube is translated into a rule.

will further address this method as ”keep” in the remaining of the paper.

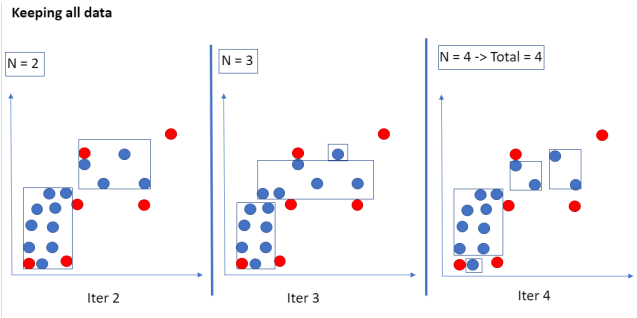

Figure 6: Keeping all datapoints in every iteration could lead to a reduced number of clusters since there may be data patterns that could only be found in this scenario.

Another proposal that we include in this paper over [18] is splitting the subspaces in a binary partition scheme. This is an alternative over the original proposal, that constantly increases the number of clusters until one rule has only datapoints from the same class, and then restarting the clustering process from the beginning for the remaining ones. We will address this method as ”split” for the remaining of the paper. Image 7 shows how the same 2D example using this approach.

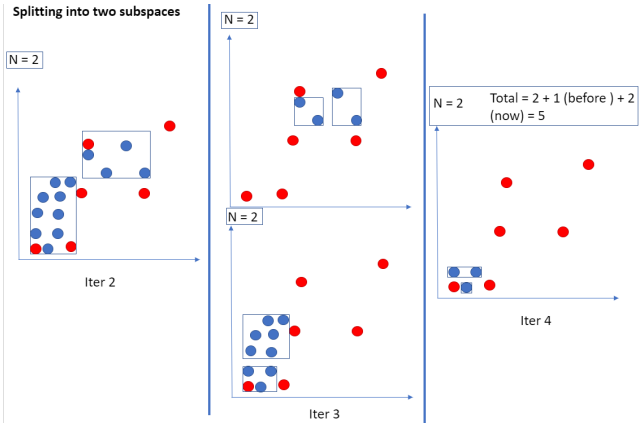

Figure 7: Splitting subspaces with a binary partition scheme until no red points are inside the rule.

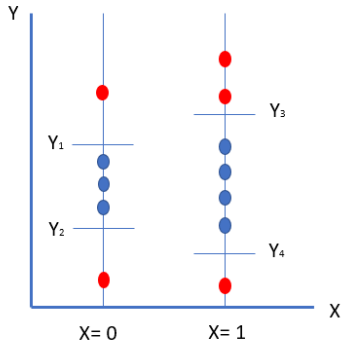

Figure 8: Rule extraction with a categorical variable.

According to the taxonomy for XAI in [10], our method has the following characteristics:

- Post-hoc: Explainability is achieved using external techniques.
- Global and individual: Explanations serve to explain how the whole model works, as well as why a specific data point is considered anomalous or non-anomalous.
- Model-agnostic: As with other techniques for global explanations [10], the only information needed to build the explanations are the input features and the outcomes of the system after fitting the model.
- Counterfactual: The explanations for why a data point is anomalous also include information on the changes that should take place in the feature values in order to consider that data point as non-anomalous.

Since the explanation algorithm is model-agnostic, it can work for any blackbox model. The only information needed is the train dataset and the outputs from the model. To illustrate it, this paper will show evaluations over OCSVM models with different kernels: radial basis function (RBF) and linear kernel.

Regarding the clustering technique itself, potentially any algorithm could be used, both for [18] or for any of out two proposals over it from this paper. However, there is a caveat that should be considered. The clustering algorithm needs to take into account if the features are only numerical, categorical (non ordinal), or both.

One algorithm that will be used in this paper for extracting the hypercubes is K-Means ++ [29]. However, this clustering algorithm is designed for numerical features, and categorical ones should be treated differently. In that case, the approximation would be to extract a rule for each of the possible combinations of categorical values among the data points that are not considered anomalous. Considering again the aforementioned 2-dimensional example, with variable X being binary categorical, a dataset may look like in Figure 8:

In that case, two rules would be extracted, one for each of the possible states of X:

- Rule 1: NOT OUTLIER IF $X = 0 \wedge Y \geq Y2 \wedge Y \leq Y1$
- Rule 2: NOT OUTLIER IF $X = 1 \wedge Y \geq Y4 \wedge Y \leq Y3$

Generally speaking, the algorithm logic can be summarised as:

- Apply OCSVM to the dataset to create the model.
- Depending on the characteristics of variables, do:
  - Case 1. Numerical only: Iteratively create clusters in the non-anomalous data (starting with one cluster) and create a hypercube using the centroid and the points further away from it. Check whether the hypercube contains any data point from the anomalous group; if it does, repeat using one more cluster than before. End when no anomalies are contained in the generated hypercubes. If there are anomalies and the data points in a cluster are inferior to the number of vertices needed for the hypercube, complete the missing vertices with artificial datapoints and end when there are no anomalies or when the convergence criterion is reached.
  - Case 2. Categorical only: The rules will correspond directly to the different value states contained in the dataset of non-anomalous points.
  - Case 3. Both numerical and categorical. This case would be analogous to Case 1, but data points will be filtered for each of the combinations of the categorical variables states. For each combination, there will be a set of rules for the numerical features.
- Use these vertices to obtain the boundaries of that hypercube and directly extract rules from them.

Besides K-Means++, there are other clustering algorithms that could be applied. In this paper we will analyse also the rules obtained by applying K-Prototypes [30]. The advantage of using K-Prototypes is that it can work directly with both categorical and numerical features.

Algorithm Description

Algorithm 1 contains the proposal for rule extraction for an OCSVM model that may be applied over a dataset with either categorical or numerical variables (or both).

ocsvm_rule_extract is the main function of the algorithm. Regarding input parameters, X is the input data frame with the features, l_n is a list with the numerical columns, l_c is a list with the categorical columns, d is a dictionary with the hyperparameters for OCSVM (kernel type, upper bound on the fraction of training errors and a lower bound of the fraction of support vectors, ν, and the kernel coefficient, γ). This function starts with the feature scaling of the numerical features (function *featureScaling*). After that, it fits an OCSVM model with all the data available and detects the anomalies within it, generating two datasets, X_y with the anomalous data points and X_n with the rest (function *filterAnomalies*).

The next step is checking the type of cluster algorithm, defined in the variable m. If it is K-Prototypes, then all columns are considered together. For this case, the algorithm directly calls $getR()$ function to obtain the rules.This function receives as input a matrix with anomalous and non-anomalous data points, X_y and X_n respectively. It also receives the list of names of those features, l, and the type of approximation for obtaining the hypercubes: [18], "keep", or "keep_reset". This is specified with the input variable t.

If cluster algorithm is K-Means++, the approach is similar, but first it is checked if the features are numerical, categorical or both. In case of only numerical columns, it directly calls function *getR*. If all the features are categorical, then the rules for non-anomalous data points will simply be the unique combination of values for them. If there are both categorical and numerical features, the algorithm obtains the hypercubes (as mentioned for numerical features only) for the subset of data points associated to each combination of categorical values.

Function $getR()$ simply then calls different subfunctions depending on the t parameter value, but in any of the cases, the approach is similar: cluster non-anomalous data points in a set of hypercubes that do not contain any anomalous data points. The "keep" apporach, described in algorithm 2, iteratively increases the number of clusters (hypercubes) until there are no anomalous points within any hypercube. The function *outPosition* checks whether the rules defined based on the vertices of the hypercube do not include any data point from the anomalous subset, X_y. *getRulesKeep* then calls function *getVertices* (described in algorithm 3) with a specific number of clusters, n_{cl}. This function performs the clustering over the non-anomalous data points, X_n, using the function *getClusters* that returns the label of the cluster for each data point, as well as the centroid position for each cluster using the specified cluster algorithm. Then, it iterates through each cluster and first obtains the subset of data points for that cluster X_{nc} with the function $insideCluster$. After that, if there are enough data points in that cluster (more data points than the vertices of the hypercube) it computes the distance of each of them to the centroid with *computeDistance* and uses the furthest n_v as vertices. In case there are less datapoints than the number of vertices that a hypercube of that dimensionality has, then all of them are used limits, and some of the vertices will be collapsed into the same datapoint. This scenario does not stop the iterations, since a hypercube in this situation could still include outliers, needing further splitting. As long as there are no outliers inside the rules, they are stored in $rules$ list. However, as soon as there is one rule with outliers inside, then the whole process is repeated again with one more cluster. This keeps taking place until no outliers are inside the rules or the maximum number of iterations is reached.

Algorithm 1 Main pipeline

procedure OCSVM_RULE_EXTRACT(X, l_n, l_c, d, m, t)
  for $c \in l_c$ **do**
    $X[:, c] \leftarrow featureScaling(X[:, c])$
  end for
  $model \leftarrow OneClassSVM(d)$
  $model.fit(X)$
  $preds \leftarrow model.train(X)$
  $distances \leftarrow model.decisionFunction(X)$
  $X_y, X_n \leftarrow filterAnomalies(X, preds)$
  if $m = kprototypes$ **then**
    $l \leftarrow l_n + l_c$
    $rules \leftarrow getR(X_n, X_y, X, l, m, t)$
  else
    if $len(l_1) = 0$ **then**
      $rules \leftarrow getR(X_n, X_y, X, l_n, m, t)$
    else if $len(l_2) = 0$ **then**
      $rules \leftarrow getUnique(X_n, l_c)$
    else
      $cat \leftarrow getUnique(X_n, l_c)$
      $rules$ empty list
      for $c \in cat$ **do**
        $X_{nf}, X_{yf} \leftarrow filterCat(X_n, X_y, c)$
        $rules.append(getR(X_{nf}, X_{ny}, l_n, m, t))$
      end for
    end if
  end if
  $rules \leftarrow featureUnscaling(rules, l_n)$
  $rules \leftarrow pruneRules(rules, l_n, l_c)$
  return $rules$
end procedure

The "split" approach is defined in algorithm 4. This function has some similarities with 2 with the following differences. Instead increasing the number of clusters in every iteration, n_{cl} is always 2. Also, l_sub receives the data after every split. Initially, l_sub contains only one dataset, the inliers X_n. However, after another iteration, its value is set to the data from the clusters in which the rules did contain some outlier.

In any of the three methods, after obtaining the rules, function $featureUnscaling$ is used to express rules in their original values (not the scaled ones used for the ML models). And function $pruneRules$ checks whether there are rules that may be included inside others; that is, for each rule it checks whether there is another with a bigger scope that will include it as a subset case.

Influence of the kernel

As mentioned before, OCSVM models are configured using mainly three hyperparameters: ν, γ and the kernel type. Depending on the kernel type, the construction of the decision

Algorithm 2 Rule Extraction - Keeping all datapoints

procedure GETRULESKEEP(X_n, X_y, n_v, l_n)
  max_iter reference value
  $check \leftarrow True$
  $n_{clusters} \leftarrow 0$
  while check **do**
    $rules$ empty list
    if $n_{clusters} > max\_iter$ **then**
      $check \leftarrow False$
    else
      $n_{cl} \leftarrow n_{cl} + 1$
      $vInfo \leftarrow getVertices(X_n, X, n_v, l_n, n_{cl})$
      for $iterValue \in vInfo$ **do**
        $rules_{cluster} \leftarrow iterValue[0]$
        $X_{nc} \leftarrow iterValue[1]$
        $l_y \leftarrow outPosition(rules_{cluster}, X_y)$
        if $len(l_y) = 0$ **then**
          $rules.append(rules_{cluster})$
          $check \leftarrow False$
        else
          $check \leftarrow True$
        end if
      end for
    end if
  end while
  return $rules$
end procedure

Algorithm 3 Additional functions

procedure GETVERTICES(X_n, n_v, l_n, n_{cl})
  d_{bounds} empty list
  d_{points} empty list
  $labels, centroids \leftarrow getClusters(X_n, l_n, n_{cl})$
  for $c \in n_{cl}$ **do**
    $X_{nc} \leftarrow insideCluster(labels, X_n)$
    if $len(X_{nc}) > n_v$ **then**
      $vertices \leftarrow computeDist(X_{nc}, labels[c])$
    else
      $vertices \leftarrow X_n$
    end if
    $d_{bounds}.append(vertices)$
    $d_{points}.append(X_{nc})$
  end for
  return d_{bounds}, d_{points}
end procedure

frontier to differentiate between outliers and inliers changes. In particular, Radial Basis Function (RBF) kernel will find hyperspheres (one or more) that enclose the inliers, leaving outliers outside.

The diverse density of outliers versus inliers highlights that there may be differences in the rules depending on which class they enclose. Mainly, since the decision function is a hypersphere, the intuition is that it will be easier to find rules that enclose all those points. Figure 9 illustrates this idea.

Algorithm 4 Rule Extraction - Binary partition apporach

procedure GETRULESSPLIT(X_n, X_y, n_v, l_n)
  max_iter reference value
  $check \leftarrow True$
  $l\_sub \leftarrow [X_n]$
  $rules$ empty list
  while check **do**
    if $len(l\_sub) == 0$ or $j > max\_iter$ **then**
      $break$
    end if
    $l\_ori \leftarrow l\_sub$
    $l\_sub \leftarrow []$
    for d in l_ori **do**
      $n_{cl} \leftarrow 2$
      $vInfo \leftarrow getVertices(X_n, X, n_v, l_n, n_{cl})$
      for $iterValue \in vInfo$ **do**
        $rules_{cluster} \leftarrow iterValue[0]$
        $X_{nc} \leftarrow iterValue[1]$
        $l_y \leftarrow outPosition(rules_{cluster}, X_y)$
        if $len(l_y) = 0$ **then**
          $rules.append(rules_{cluster})$
          $check \leftarrow False$
        else
          $check \leftarrow True$
          $l\_sub \leftarrow l\_sub.append(X_{nc})$
        end if
      end for
    end for
  end while
  return $rules$
end procedure

Algorithms for metrics

As mentioned before, the metrics considered in this paper are divided into four subsets: comprehensibility, representativeness, stability and diversity. Since, to the best of our knowledge, some of these metrics are not implemented within the main XAI framerworks, we propose within these paper a set of algorithms to compute them in a rule extraction scenario.

- **Metrics for comprehensibility**: Number of rules (n_rules), size of the rules ($size\_rules$).
- **Metrics for representativeness**: Percentage of datapoints explained with P@1 rules (per_p1) and the median percentage coverage of datapoints by each rule ($p1\_coverage$).
- **Metrics for stability**: How many artificial points (similar to a subset of prototypes from the dataset) are classified by the rules within the same class ($rule\_agreement$), how many of those new artificial datapoints have the same predictions that the original blackbox model ($precision\_vs\_model$).
- **Metrics for diversity**: Degree of hyperspace overlapping between all the rules ($score\_intersect$).

The metrics for "comprehensibility" are directly analyzed from the rules themselves; n_rules is computed counting

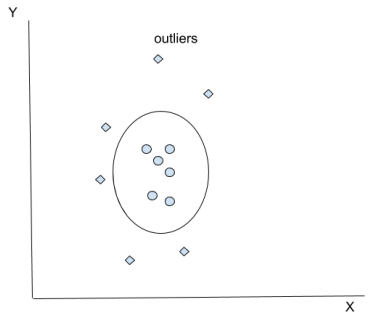

Figure 9: With an RBF Kernel the correct hypercube will be the one that encloses the points that are not anomalies, since the OCSVM algorithm will try to enclose most of the points inside the decision frontier and leave anomalies outside.

the number of rules generated, and $size\_rules$ is computed checking the elements that define the rule (p.e. $X > 3$ AND $X < 7$ AND $Y > 1$ have a $size\_rules = 3$ while $X > 3$ have a $size\_rules = 1$).

The metric per_p1 for "representativeness" simply checks the percentage of datapoints for the target class explained with P@1 rules. The other metric in this group is $p1\_coverage$. It checks the median performance of the rules themselves: it computes the median percentage of coverage for the target class by each rule.

The metrics $rule\_agreement$ and $precision\_vs\_model$ computes the "stability" metrics of the hypercubes. The first step is obtaining the prototypes from the dataset and generate random samples near them. Then, obtain the prediction of the original model for those dummy samples and checks, if when the prediction is inlier/outlier, there is at least one rule that includes that datapoint within it. To check the level of agreement between the rules, since the artificial datapoints near the prototypes should belong to the same class, the function checks if the final prediction using all rules is the same for all of them.

The steps for these metric are described below, and the detailed pseudocode appears in algorithm 5.

Rule agreement:

- Choose N prototypes that represent the original hyperspace of data
- Generate M samples close to each of those N prototypes using Protodash algorithm [31]; the hypothesis is that close points should be generally predicted belonging to the same class.
- For each of those N*M datapoints (M datapoints per each N prototype) check whether the rules (all of them) predict them as inliner or outlier; the datapoints that come into the function are either outliers or inliers. If they are inliers, then the rules identify an artificial datapoint (of those M*N) as inlier if it is outside every rule. If the datapoints are outliers it's the same reversed: a datapoint is an inlier if no rule includes it.
- Then, it is checked the % of datapoints labeled as the assumed correct class (inliers or outliers), neighbours of that prototype compared to the total neighbours of that prototype.
- All the % for each prototype are averaged into one %.

Model agreement:

- The % of predictions for the artificial datapoints aforementioned that are the same between the rules and the original OCSVM model.

Algorithm 5 receives the dataset X of inliers/outliers (depending if the rules are computed for inliers or outliers), the rules X_r and the OCSVM fitted and trained model clf. Then obtains the protoypes with $ProtodashExplainer()$ function and generates the random samples X_s near them with $randomNear()$, where an upper and lower limits (th_s, th_l) can be defined for how close are those points to the prototypes. Then, it checks which rules enclose that datapoint with $checkInR()$, and if at least one of them encloses the datapoint, it is considered that it can be classified using the rules. The metric $precision\_vs\_model$ is specified in $n\_precision$ variable, that checks the percentage of agreement between the classifications using the rules and the ones with the model, through $checkInModel()$ function. Finally, $rule\_agreement$ is included in the variable $n\_agreement$, that checks the percentage of predictions, using the rules, that match the category of datapoints used in X (inliers/outliers).

Algorithm 5 Stability

procedure GETAGREEMENT(X, X_r, clf)
  $X_p \leftarrow ProtodashExplainer(X)$
  $X_s \leftarrow []$
  for $p \in X_p$ **do**
    $X_s \leftarrow X_s.append(randomNear(p, th_l, th_s))$
  end for
  $n\_precision \leftarrow 0$
  $l\_rules \leftarrow []$
  for $d \in X_s$ **do**
    $l\_iter \leftarrow []$
    for $r \in X_r$ **do**
      $l\_iter \leftarrow l\_iter.append(checkInR(d, r))$
    end for
    $r\_rules \leftarrow max(l\_iter)$
    $r\_model \leftarrow checkInModel(d, clf)$
    if $r\_rules = r\_model$ **then**
      $n\_precision \leftarrow n\_precision + 1$
    end if
    $l\_rules \leftarrow l\_rules.append(r\_rules$
  end for
  $n\_precision \leftarrow n\_precision/len(X_s)$
  $n\_agreement \leftarrow l\_rules[= 1]/len(X_s)$
  return $n\_precision, n\_agreement$
end procedure

The metric to measure "diversity" is $score\_intersect$, and it analyses if the rules are different with few overlapping concepts. This is computed checking the area of the hypercubes of the rules that overlaps with another one. The way to

check this is by seeing the 2D planes of each hypercube (by keeping two degrees of freedom for the features in the hyperplane coordinates; n-2 features are maintained and the other two are changed between their max/min values in order to obtain the vertices of that 2D plane). Then, it is computed the area of the 2D planes for the rules that overlaps, adding for all possible 2D planes the total area overlapped for each rule. In order to compute a score, the features are normalized in order to have values between 0 and 1. The pseudocode for this metric appears in algorithm 6. Algorithm 6 receives the dataset X of inliers/outliers (depending if the rules are computed for inliers or outliers), the rules X_r, the list of columns for numerical features l_n and the one for categorical l_c. The first step is obtaining all the two tuples combinations of numerical features, using $combinations()$ function. After that, it obtains the combination of categorical values with function $unique()$, and then normalizes the numerical values with $normalize()$ in order to consider the overlapping of rules equal for all features. The algorithm then analyses separately the rules that belong to each categorical combination values. For each of those subset of rules X_r_i, if there are at least two rules, then it defines the tuples of possible rule combinations, $combR$. Then, it iterates per each combination of two numerical features. These two features will correspond to the features that will be changed, leaving the rest of the l_fix features fixed, in order to extract 2D planes from the hypercubes, using $scorePolys()$ function, and storing those planes in $polys$ variable.

Image 10 describes the process for an example in a 3D space. Since all the rules translate into a hypercube, we can choose two features at a time (leaving the rest fixed) and obtain the coordinates for those 2D planes (using their vertices values). Then, for two rules, we can see the area of overlapping between those 2D hyperplanes. This is repeated for all 2D planes of the hypercubes, computing the mean value of all overlapped planes. Since the features are normalized, all the overlapped areas will have a value between 0 (no overlap) and 1 (total overlap), so the final mean value of all overlaps will be between 1 and 0. In order to express a score value, the final result will be $1 - score\_i$, so a perfect score will be the one corresponding to no overlap between the rules. The algorithm also returns the number of intersections.

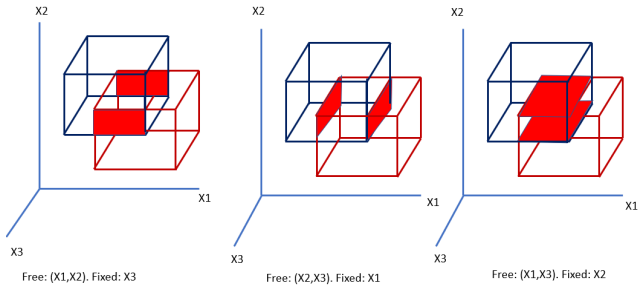

Figure 10: The overlapping between rules (hypercubes) approximated using their 2D planes' area of intersection.

Algorithm 6 Diversity

procedure GETINTERSCORE(X, X_r, l_n, l_c)
  $l\_free \leftarrow combinations(l\_n, 2)$
  $X\_c \leftarrow unique(X[l\_c])$
  $X[l\_n] = normalize(X[l\_n])$
  $score \leftarrow []$
  $n\_inter \leftarrow 0$
  for $cat \in rows(X\_c)$ **do**
    $X\_r\_i \leftarrow X\_r[cat]$
    if $len(X\_r\_i) > 2$ **then**
      $combR \leftarrow combinations(X\_r\_i, 2)$
      for $pair_f \in l\_free$ **do**
        $l\_fix \leftarrow l\_n[! = pair_f]$
        $polys \leftarrow get2D(combR, l\_fix, pair_f)$
        $score\_i, n\_i \leftarrow scorePolys(polys)$
        $score \leftarrow score.appends(1 - score\_i)$
        $n\_inter \leftarrow n\_inter + n\_i$
      end for
    end if
  end for
  $score \leftarrow mean(score)$
  return $rules$
end procedure

From local to global rules

Anchors [19] is one way of extracting rules for XAI. However, Anchors, as mentioned before, is a local method that explains one datapoint with rules. To be able to use it in the evaluation carried out in this paper, we need to turn Anchors into a global method. A simple way to do that is extracting rules for each datapoint of the input dataset, and prune those rules in order to keep the most relevant ones. This way, the whole dataset is explained, and the results can be compared with the remaining algorithms.

From the computational cost side, since obtaining Anchors rules for each datapoint is costly, we propose using Protodash [31] for scenarios when the dataset is too big. In this case, Protodash will select the relevant prototypes from the dataset, and Anchors will obtain the rules only for those points.

Pruning rules

Many of the rules obtained with all the methods described above are suboptimal, since they can be enclosed into another bigger rule. In order to reduce the number of rules, and remove redundancies, we apply a simple pruning technique prior to the evaluation and computing of metrics. We check every hypercube generated and see if their limits are inside any other rule. If they are, we eliminate that rule from the set of rules. We check this for every rule against every other rule in the dataset, and we keep checking it in a loop until no rules are eliminated.

Combining everything

There is a question that will arise at this point: What rule would be better? One with better results in "comprehensibility", or one with better results at, for instance, "Diver-

sity"? When there is a need to choose a tradeoff, what criteria should be prioritized? The answer to this will heavily depend upon the domain needs. However, in general terms, that criteria can be answered by computing the metrics all together with the usage of a function.

$final\_metric = f(C, R, S, D)$ with C representing the comprehensivility metrics, R the representativeness, S the stability and D the diversity. There is another aspect that can be considered while creating a function to encapsulate all metrics. In general, it is better to have a lower value for comprehensibility metrics (less rules, less rule size) since that contributes to an enhancement of comprehensibility. Regarding the rest of the metrics, higher values are better. Thus, a simple way to compute this is adding the results for representativeness, stability and diversity (adjusting their relative importance by a set of weights), and dividing by comprehensivility results. With this, a higher final value will be better. This is expressed in Equation 3.

$$\begin{aligned} C &= \frac{1}{\alpha_1 * n\_rules + \alpha_2 * size\_rules} \\ R &= \beta_1 * per\_p1 + \beta_2 * p1\_cov \\ S &= \gamma_1 * rule\_ag + \gamma_2 * p\_vs\_m \\ D &= \theta * score\_int \\ final\_metric &= C * (R + S + D) \end{aligned} \tag{3}$$

Since the values for the metric of comprehensibility are the only ones that are not in a range o 0 to 1, we normalize them before computing this metric in order to have all values in the same range.

Evaluation

We use our algorithm over different datasets (both public and from Telefonica's real data), to evaluate the following hypotheses:

- It is possible to apply post-hoc model-agnostic XAI techniques for unsupervised outlier detection models that offer both global and local explanations with counterfactual information. This is exemplified analysing rule extraction techniques over OCSVM models.
- It is possible to quantify the quality of explanations with XAI metrics that measure "comprehensivility", "representativeness", "stability" and "diversity" in a rule extraction technique scenario. This will be exemplified using the rule extraction algorithms detailed in this paper.
- Some rule extraction algorithms for OCSVM models are better suited if the aim are P@1 rules. In particular, [18] and the two modifications introduced in this paper will offer better P@1 rules than other rule extraction techniques. This will be evaluated using both the metrics and algorithms mentioned in the previous point. We will also compute the metrics into one single value, using the Equation 3 with all the weights with a value of 1.
- Even if the techniques are model-agnostic (can be applied to any model), there will be differences in the metrics obtained depending on the model hyperparameters. Since OCSVM using RBF kernel tends to group the inliers in one (or many) hyperspheres, the metrics for inlier's rules will be different than those for outlier's. In particular, the number of rules for inliers and RBF kernel will be less than the ones obtained.
- The same hypotheses from the previous point could be extended for the comparison between linear and RBF kernel. RBF kernel for inliers will yield less rules than the Linear one for inliers.

The datasets used belong to different domains, have different sizes and different number of features (both categorical and numerical). They are indicated in Table 1:

- Datasets 1 and 2 about seismic activity [32]. Dataset 1 is bi-dimensional with only numerical features ('gdenergy', 'gdpuls'). Dataset 2 has 2 categorical features ('hazard', 'shift') and 7 numerical ('seismoacoustic', 'shift', 'genergy', 'gplus', 'gdenergy', 'gdpuls', 'hazard', 'bumps', 'bumps2').
- Dataset 3 about cardiovascular diseases [33]. There are 4 categorical features ('smoke', 'alco', 'active', 'is_man') and 7 numerical ('age', 'height', 'weight', 'ap_hi', 'ap_lo','cholesterol','gluc').
- Dataset 4 from a call center at Telefónica (TEF Comms). It is real data that includes the total number of calls received in one of its services during every hour. Using these data, some features are extracted (weekday), and they are cyclically transformed, so that each time feature turns into two features for the sine and cosine components. The rules in this case are also transformed back into the original features in order to enhance rule comprehension.
- Dataset 5 contains Telefónica's data about IoT deviced attached to cars for vehicle tracking. The data is aggregated in daily windows for each vehicle, representing features that model the daily behaviour of that vehicle. It contains 49 numerical features (such as the number of events with high RPM or the maximum temperature of the coolant), and 12 categorical ones (binary variables that indicate the model and make of that car).
- Dataset 6 refers to US census for year 1990 [34]. It has 2 categorical features ('dAncstry1_3', 'dAncstry1_4') and 4 numerical ones ('dAge', 'iYearsch', 'iYearkwrk', 'dYrsserv').

Dataset	Ref.	Nº Cat.	Nº Num.	Nº Rows
1	[32]	0	2	669
2	[32]	2	7	1705
3	[33]	4	7	42000
4	TEF Comms	0	5	2712
5	TEF Fleet	12	49	59844
6	[34]	2	4	1000000

Table 1: Description of each dataset, with their reference (Ref.), categorical features (Nº Cat.), numerical features (Nº Num) and number of rows.

We ran experiments with the following infrastructure: the implementations of the OCSVM algorithm, the K-Means++ clustering and the DT algorithms are based on Scikit-Learn [35]. The rest of the code described in Algorithms 1 and 2 were developed from scratch, and available in Github [36].

OCSVM models use as hyperparameters: $\nu = 0.1, \gamma = 0.1, kernel = rbf$ or $\nu = 0.1, \gamma = 0.1, kernel = linear$ for linear kernel. K-means++ models use $max\_iter = 100, n\_init = 10, randomState = 0$. K-Prototypes uses $init =' Huang', max\_iter = 5, n\_init = 5$. DT uses default parameters, with $randomState = 42$ and Gini criterion to find the best splits. All Protodash applications use $kernelType =' Gaussian', sigma = 2$, with $m = 1000$ for the samples used in the Anchors rule extraction step, and $m = len(rules)$ for the computation of metrics, having m at least a value of 20. RuleFit uses $tree\_size = len(feature\_cols) * 2$, $rfmode =' classify'$ with $len(feature\_cols)$ the number of features that appear in each dataset. RuleFit also considers only rules with a non zero coefficient, and with an importance > 0. For SkopeRules, since we want only P@1 rules, we use $random\_state = 42$, $precision\_min = 1.0$, $recall\_min = 0.0$. FRL and Anchors use both their default library parameters. BRLG uses $lambda_0 = 1e-3, lambda_1 = 1e-3, CNF = False$. LOGRR uses $lambda_0 = 0.005, lambda_1 = 0.001, useOrd = True$. GRLM uses $maxSolverIter = 2000$ considering only coefficients with value > 0.

Dataset 1 allows us to visually see the rules obtained using different the different methods. Figure 12 shows the results of applying the proposal of [18], while 11 and 13 shows the results by applying our two alternative methods, using for all the cases the same RBF kernel and K-Means++ clustering. To visually see the changes with other methods, Figure 14 shows the rules for inliers obtained with a DT surrogate method, and 15 shows the results using Anchors while considering the restriction of using only P@1 rules. As we can see, the results are quite different. "Split" and "keep_reset" methods generate rules that cover the whole dataset, though they have overlapping between them. "Keep" method does not have any overlapping, but generates a lot of rules. Anchors does not seem to cover the whole dataset. And DT method generates good rules that do not have any degree of overlapping.

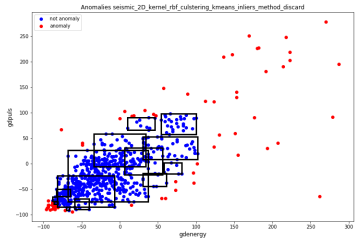

Figure 11: Hypercubes in a 2D space for inliers using "split" method, K-Means clustering, and RBF kernel.

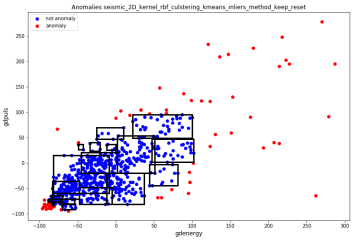

Figure 12: Hypercubes in a 2D space for inliers using "keep_reset", K-Means clustering, and RBF.

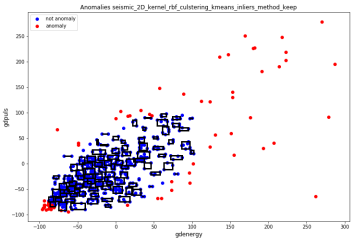

Figure 13: Hypercubes in a 2D space for inliers using "keep" method, K-Means clustering, and RBF kernel.

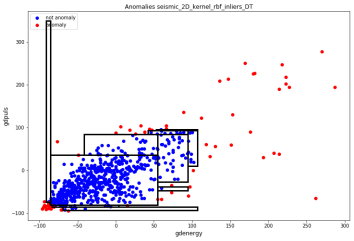

Figure 14: Hypercubes in a 2D space for inliers using "DT" method and RBF kernel.

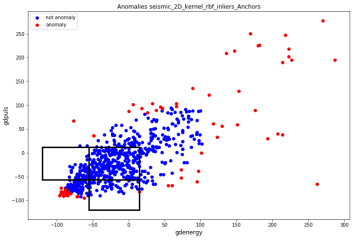

Figure 15: Hypercubes in a 2D space for inliers using "Anchors" method and RBF kernel.

As we mentioned before, the analysis will also consider a Linear kernel to show how the rules extraction methods can be applied over any model, as well as seeing if there are changes in the metrics used even though the methods are

model-agnostic by themselves. As an example, Figures and 17 show the rules extracted using the same Dataset 1 but considering a Linear kernel, and using RuleFit and BRLG methods respectively.

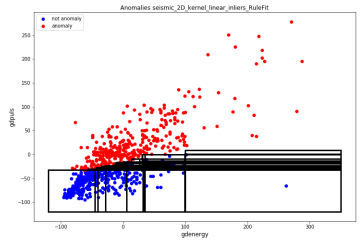

Figure 16: Hypercubes in a 2D space for inliers using "RuleFit" method and Linear kernel.

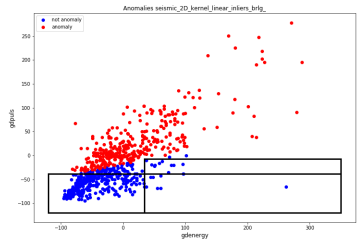

Figure 17: Hypercubes in a 2D space for inliers using "BRLG" method and Linear kernel.

As mentioned before, the rules considered are always P@1. This means that rules that are not P@1 are discarded for any of the analyses carried out. As an example, Figure 18 shows rules generated for BRLG, RBF kernel and outliers where some of those rules are not P@1 (they contain datapoints from the other class).

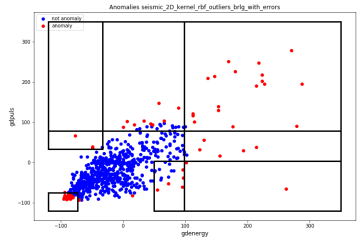

Figure 18: Hypercubes in a 2D space for outliers using "BRLG" method and RBF where rules that are not P@1 are included. kernel.

The previous Figures shown how if was possible to apply rule extraction techniques over a OCSVM model using a simple 2D dataset in order to be able to visualize the results. Going beyond this, we apply different rule extraction techniques over the datasets mentioned before, which include already existing solutions in the literature, as well as our proposed modifications over [18], and over Anchors [19] in order to turn it into a global rule extraction model. We have already seen that these rule extraction techniques are all model-agnostic because we applied them over a OCSVM model regardless of the kernel. However, we are going to keep these two kernel configurations for the next analysis in order to see potential differences in the metrics obtained.

Thus, for the following analysis (measurement of comprehensivility", "representativeness", "stability" and "diversity"), we are going to use the rule extraction algorithms described in the literature, as well as that kernel configuration of OCSVM (RBF or linear) over the 6 datasets mentioned before. Two caveats to take into account is that K-Prototypes clustering can only be use for the datasets that have categorical features (all expect datasets 1 and 4), and that we we are only going to use RBF kernel for the dataset 6.

The results for the clustering-based methods can be seen in Table 4, and for the remaining methods in 4. Those tables present the median aggregated value for each of the metrics aforementioned among the different datasets involved. We also include the results per datatset for Anchors modification in Table 6.

The first thing that can be seen is that clustering-based methods always yield P@1 rules regardless of the kernel type, the clustering technique used, or the algorithm alternative. This is an important aspect, specially compared to methods such as brlg, logrr or FRL that are not able to generate P@1 rules for some situations. This is also true for Anchors, RuleFit, DT and SkopeRules.

Regarding comprehensibility (n_rules and size_rules), for inliers and linear kernel, using K-Means clustering always yield more rules than using any of the other rule extraction methods, regardless of the algorithm used out of the three proposals. The number of rules obtained is mostly equivalent for the three clustering-based algorithm proposed. The same is applicable to linear kernel and outliers datapoints: more rules using clustering-based methods with K-Means than using the other methods, with a stable number of rules. Exactly the same can be said about the size of rules for this configurations. Analysing the results with RBF kernel, we can see that the number of rules obtained are higher than using linear one, and then, even higher also than with the other methods (though the rule size is similar regardless of the kernel). Thus, considering only the comprehensibility area, clustering-based methods generally yield rules with a similar rule size than the remaining methods, and regarding the number of rules, it slightly increases for RBF kernel, and increases more for K-Prototypes than for K-Means if the method used is split.

For per_p1, first, we can see that split and keep_reset methods yield constantly better results than keep method, regardless of the kernel, clustering method or if it is inliers or outliers. If the datapoints are inliers, the metric is similar for both split or keep_reset. However, if it is for outliers, split seems to offer better results. Regarding p1_cov, K-Means offer better results than K-Prototypes in general. The keep_reset method generates rules with better coverage, generally speaking, than the other cluster-based methods.

The per_p1 value for the remaining methods is similar for DT and RuleFit if the kernel is linear and for inliers and outliers. However, when using RBF kernel, the results fall for every other method except for RuleFit. The coverage of RuleFit rules is on pair with the clustering-based methods. It is worth noticing that one of the best coverages is obtained with Anchors when using a linear kernel and for outliers. This highlights that the rules extracted with this method are able to cover a huge amount of the hyperspace.

For all the methods that yield P@1 rules, the rule agreement seem to be on pair, offering similar results. Regarding the precision versus the original model, for the clustering-based methods, K-Means offer more better results than K-Prototypes. However, for inliers, the results are quite similar, with the exception of RBF kernel, where K-Prototypes and split outperforms every other combination. Comparing the three algorithm proposals, both split and keep_reset offer stable similar results, while keep drops in some of the scenarios. The precision versus model is clearly better than with the other rule extraction algorithm: clustering-based methods approximate better the decision frontier of the underlying model.

In this area, clustering-based methods are clearly outperformed by the other rule extraction techniques, though it is logical. Clustering-based methods yield more rules, and that may lead to more overlapping, and then less score in this metric. Regarding only the clustering-based methods, the results are quite similar for all of them. It is worth noticing how SkopeRules yields many perfect scores (no overlapping at all).

The aggregated metrics results into one single value appear at Table 7 for clustering-based methods, and in Table 8 for the remaining ones. Here, we can see for inliers and linear kernel how the final metric is higher for K-Means than the other rule extraction methods (it doubles them). K-Prototypes. however, yield similar results than those other rule extraction methods. It happens the same for outliers and linear kernel, though here, the final metric is significantly lower if we use dicard method than with the other two algorithms. For RBF kernel and inliers, the metric of SkopeRules is on pair with the one with split and keep, but keep_reset yields a very good metric. The metrics for outliers are also higher with clustering-based methods, where we highlight how K-Means for keep_reset and K-Prototypes with keep yield the higher metrics.

At this point, we can conclude the following things.

- First, even though the methods are model-agnostic, the kernel chosen does affect the final results. This is seen clearly in the non clustering-based methods, where the metrics drop significantly for every technique if we choose a RBF kernel. However, for the clustering-based methods, generally speaking, the results are quite similar (with some exceptions with some configurations). Thus, those algorithms yielded a more model-agnostic results in the experiments carried out.
- We can also conclude that RuleFit seems to be the best of those non cluster-based techniques, specially for outliers. It yields less rules, and thus approximates worse the whole model, but on the contrary, it offers good coverage for the rules extracted, with very little overlapping between them.
- We can also conclude that our proposal for K-Means (which includes dealing with categorical non-ordinal features) offers stable results that generally outperform K-Prototypes.
- Finally, among the clustering-based methods, out of our two modifications over [18], we can see that the split proposal is a good alternative, offering better results even though it sometimes means generating some rules more (but with similar rule size).

The next hypothesis checked is if the kernel used affects the number of rules generated, since a RBF kernel groups the inlier datapoints inside hypershperes. We will analyse these by comparing the P@1 rules extracted for each dataset for outliers with RBF kernel, inliers with a linear kernel, and inliers with RBF kernel, in order to see if this last configuration yield less number of rules. We are going to use for this part only the clustering-based methods since they were the ones that yielded more consistent P@1 rules. The results can be seen in Table 2 for k-means clustering and in 3 for K-Prototypes clustering.

For K-Means clustering and split method, out of the 6 datasets, the number of P@1 rules for inliers and RBF clustring is slightly inferior in 3 of them. For datasets 4, 5 and 6 (that have more rows and/or more features) the number of rules is higher. For keep method and K-Means clustering RBF and inliers are, in fact, the combination that constantly yields more rules. With K-Means and keep_reset method, RBF and inliers yield normally less rules than with outliers, but the results compared against inliers and linear kernel are mostly the same.

Using K-Prototypes does not change much the results over K-Means. The main difference is that K-Prototypes have more combinations of scenarios where there are no P@1 rules. In neither split, nor keep, nor keep_reset, the number of rules for inliers and RBF kernel is steadily lower than the ones with the other configurations.

Software Used

The main libraries used for the work done in this paper are the following ones.

- OCSVM, DT [37]
- Anchors [38]
- Protodash, GRLM, BRLG [24]
- RuleFit [39]
- SkopeRules [40]
- FRL [41]

As we mentioned before, we also provide a library with all the developed algorithms [36].

Limitations of our Approach

Regarding the XAI metrics and evaluations, the first limitation to consider is that the only model used is OCSVM

(with two types of kernel). Even though the algorithms used and the metric definitions are model-agnostic and could be applied over other outlier detection models, the results may differ if other unsupervised models or kernels are used. Together with this, our suggestion of a function that aggregates every metric is a simple baseline that can be further improved.

Also, the analyses carried out are focused in P@1 rules. Thus, the conclusions may be different if all the rules extracted are used, regardless of their precision value.

Finally, all the rules (for cluster-based methods) and all the checking to see if a data point is inside a hypercube (for all methods) are defined with inequalities (¿=, ¡=). Because of that, the results may be different if we allow values from the other class to be at the limit of the hypercube.

Conclusion

In this paper we have analysed the application of XAI techniques over unsupervised outlier detection models through the usage of rule extraction methods applied to OCSVM models. Among the rule extraction techniques, we used both algorithms from the literature, as well as new alternatives that we propose and evaluate together with them. Our first aim was analysing the quality of the rules extracted from a XAI perspective. We have done this by defining metrics for different aspects related to XAI: comprehensiviliy, representativeness, stability and diversity, as well as proposing a function to aggregate all those metrics together. We evaluated those metrics over different data sets, both from public sources as well as from Telefónica's, using communications and IoT generated data for that purpose. The results for the metrics shown how from among the rule extraction techniques considered, clustering-based techniques yield better results in a context of P@1 rules, at the expense of generating more rules than those obtained through other methods. Related to those clustering-based techniques, we saw how one of our algorithm proposals yield similar (and sometimes even better) results than the ones obtained with the baseline algorithm from the literature.

Our evaluation considered model-agnostic techniques that can be applied over any black-box model. In order to check this empirically, we have used OCSVM models with different types of kernel configurations (linear and RBF). We saw how for most of the rule extraction techniques, even though they are model agnostic, the metrics obtained for P@1 rules differ significantly depending on the type of kernel used (better metrics for linear kernel). However, clustering-based techniques yield results that are more model-agnositc, since they are more stable regardless of the kernel chosen.

We also analysed if the number of rules extracted for inliers and using a RBF kernel are significantly lower than those for outliers or those obtained with other kernels such as a linear one, since a RBF kernel in a OCSVM model tends to group all the points inside a hypersphere. This analysis was also focused for P@1 rules. In this regard, we were not able to see any significant differences in the number of rules extracted with any of the kernel configurations.

Future Work

There are several research lines that can be pursued following the work presented at this paper. The first one to consider is benchmarking these results against other rule extraction techniques not covered in this paper. An example is G-Rex algorithms [42]. Another research line that can be followed is analysing the metrics of the rule extraction techniques over other unsupervised anomaly detection models, such as IsolationForests [14] or LOF [15], as well as using other kernel configurations in OCSVM (such as a polinomial one). Also, while we have proposed a vanilla function to incorporate the metrics belonging to different XAI areas, there is much room of improvement over it in order to find an optimal function that weights appropriately every term. Finally, rule extraction should also be designed to consider all types of comparisons (¿=, ¡=, ¿ and ¡), and this is something that could also be considered in the cluster-based methods developed.

References

[1] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58 (2020), pp. 82–115.

[2] Richard Benjamins, Alberto Barbado, and Daniel Sierra. *Responsible AI by Design*. 2019. arXiv: 1909.12838 [cs.CY]. URL: https://arxiv.org/abs/1909.12838.

[3] Till Speicher et al. "A Unified Approach to Quantifying Algorithmic Unfairness". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '18* (2018). DOI: 10.1145/3219819.3220046. URL: http://dx.doi.org/10.1145/3219819.3220046.

[4] Moritz Hardt, Eric Price, and Nathan Srebro. "Equality of Opportunity in Supervised Learning". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 3323–3331. ISBN: 978-1-5108-3881-9. URL: http://dl.acm.org/citation.cfm?id=3157382.3157469.

[5] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. *Mitigating Unwanted Biases with Adversarial Learning*. 2018. arXiv: 1801.07593 [cs.LG]. URL: https://arxiv.org/abs/1801.07593.

[6] Rachel KE Bellamy et al. "AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias". In: *arXiv preprint arXiv:1810.01943* (2018).

[7] Pedro Saleiro et al. "Aequitas: A Bias and Fairness Audit Toolkit". In: *arXiv preprint arXiv:1811.05577* (2018).

[8] Leilani H Gilpin et al. "Explaining explanations: An overview of interpretability of machine learning". In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE. 2018, pp. 80–89.

[9] Shane T Mueller et al. “Explanation in Human-AI Systems: A Literature Meta-Review, Synopsis of Key Ideas and Publications, and Bibliography for Explainable AI”. In: *arXiv preprint arXiv:1902.01876* (2019).

[10] Christoph Molnar. *Interpretable machine learning*. Lulu.com, 2019. URL: https://christophm.github.io/interpretable-ml-book/.

[11] Arthur Zimek and Peter Filzmoser. “There and back again: Outlier detection between statistical reasoning and data mining algorithms”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.6 (2018), e1280.

[12] Jacob Kauffmann, Klaus-Robert Müller, and Grégoire Montavon. “Towards explaining anomalies: a deep Taylor decomposition of one-class models”. In: *Pattern Recognition* (2020), p. 107198.

[13] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. “Machine Learning Interpretability: A Survey on Methods and Metrics”. In: *Electronics* 8.8 (2019), p. 832.

[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation forest”. In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 413–422.

[15] Markus M Breunig et al. “LOF: identifying density-based local outliers”. In: *ACM sigmod record*. Vol. 29. 2. ACM. 2000, pp. 93–104.

[16] Bernhard Schölkopf et al. “Support vector method for novelty detection”. In: *Advances in neural information processing systems*. 2000, pp. 582–588.

[17] David Martens et al. “Rule extraction from support vector machines: an overview of issues and application in credit scoring”. In: *Rule extraction from support vector machines*. Springer, 2008, pp. 33–63.

[18] Haydemar Núñez, Cecilio Angulo, and Andreu Català. “Rule extraction from support vector machines”. In: *Esann*. 2002, pp. 107–112.

[19] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Anchors: High-precision model-agnostic explanations”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[20] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “” Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[21] Jerome H Friedman, Bogdan E Popescu, et al. “Predictive learning via rule ensembles”. In: *The Annals of Applied Statistics* 2.3 (2008), pp. 916–954.

[22] Fulton Wang and Cynthia Rudin. “Falling rule lists”. In: *Artificial Intelligence and Statistics*. 2015, pp. 1013–1022.

[23] Sanjeeb Dash, Oktay Gunluk, and Dennis Wei. “Boolean Decision Rules via Column Generation”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 4655–4665. URL: http://papers.nips.cc/paper/7716-boolean-decision-rules-via-column-generation.pdf.

[24] Vijay Arya et al. *One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques*. Sept. 2019. URL: https://arxiv.org/abs/1909.03012.

[25] Dennis Wei et al. “Generalized Linear Rule Models”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 6687–6696. URL: http://proceedings.mlr.press/v97/wei19a.html.

[26] Vijay Arya et al. “One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques”. In: *arXiv preprint arXiv:1909.03012* (2019).

[27] Vic Barnett. *Outliers in statistical data*. Tech. rep. 1978.

[28] Richard J Beckman and R Dennis Cook. “Outlier..... s”. In: *Technometrics* 25.2 (1983), pp. 119–149.

[29] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. Stanford, 2006.

[30] Jinchao Ji et al. “An improved k-prototypes clustering algorithm for mixed numeric and categorical data”. In: *Neurocomputing* 120 (2013), pp. 590–596.

[31] Karthik S Gurumoorthy et al. “Efficient Data Representation by Selecting Prototypes with Importance Weights”. In: *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2019, pp. 260–269.

[32] Saket Sathe and Charu Aggarwal. “LODES: Local density meets spectral outlier detection”. In: *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM. 2016, pp. 171–179.

[33] Meghana Padmanabhan et al. “Physician-friendly machine learning: A case study with cardiovascular disease risk prediction”. In: *Journal of clinical medicine* 8.7 (2019), p. 1050.

[34] Catherine Blake. “UCI repository of machine learning databases”. In: *http://www. ics. uci. edu/˜ mlearn/MLRepository. html* (1998).

[35] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] Alberto Barbado. *Rule extraction in unsupervised outlier detection for XAI*. https://github.com/AlbertoBarbado/unsupervised-outlier-transparency. 2019.

[37] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[38] Janis Klaise et al. *Alibi: Algorithms for monitoring and explaining machine learning models*. Version 0.3.2. Feb. 17, 2020. URL: https://github.com/SeldonIO/alibi.

[39] Christoph Molnar. *RuleFit*. Version 0.2. Nov. 24, 2017. URL: https://github.com/christophM/rulefit.

[40] *SkopeRules*. URL: https://github.com/scikit-learn-contrib/skope-rules.

[41] Tong Wang Zhen Li. *Bayesian Rule Set Mining*. URL: https://pypi.org/project/ruleset/.

[42] Rikard Konig, Ulf Johansson, and Lars Niklasson. “G-REX: A versatile framework for evolutionary data mining”. In: *2008 IEEE International Conference on Data Mining Workshops*. IEEE. 2008, pp. 971–974.

[43] David Arthur and Sergei Vassilvitskii. “k-means++: The advantages of careful seeding”. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.

[44] Anil Chaturvedi, Paul E Green, and J Douglas Caroll. “K-modes clustering”. In: *Journal of classification* 18.1 (2001), pp. 35–55.

Annex

type_points	kernel	method	1	2	3	4	5	6
inliers	linear	split	17	56	1064	49	1075	9
inliers	rbf	split	14	53	1064	111	1092	276
outliers	rbf	split	23	63	1080	61	178	279
inliers	linear	keep	46	60	82	7	177	49
inliers	rbf	keep	100	166	349	26	199	56
outliers	rbf	keep	48	93	90	1	150	59
inliers	linear	keep_reset	13	52	344	39	704	12
inliers	rbf	keep_reset	21	56	354	7	552	47
outliers	rbf	keep_reset	15	60	937	49	119	157

Table 2: Number of rules for clustering methods and K-means clustering.

type_points	kernel	method	2	3	5	6
inliers	linear	split	116	966	55	12
inliers	rbf	split	49	972	1162	255
outliers	rbf	split	58	1051	18	271
inliers	linear	keep	10	4	0	37
inliers	rbf	keep	22	3	0	3
outliers	rbf	keep	0	8	0	13
inliers	linear	keep_reset	23	2	1	18
inliers	rbf	keep_reset	22	3	0	0
outliers	rbf	keep_reset	17	8	0	1

Table 3: Number of rules for clustering methods and K-prototypes clustering.

kernel	type_points	method	clustering	n_rules	size_rules	per_p1	p1_cov	rule_ag	p_vs_m	score_int
linear	inliers	split	kmeans	56	17.8271	0.9983	0.0039	0.9555	0.82	0.2316
linear	inliers	split	kprototypes	116.0	27.1552	0.9772	0.0007	0.9829	0.73	0.1207
linear	inliers	keep	kmeans	60.0	18.1463	0.5225	0.0046	1.0	0.69	0.3284
linear	inliers	keep	kprototypes	7.0	26.925	0.1805	0.0124	1.0	0.8	0.2318
linear	inliers	keep_reset	kmeans	52	17.4244	0.9336	0.0039	0.9636	0.65	0.2554
linear	inliers	keep_reset	kprototypes	2.0	31.6087	0.1982	0.0104	1.0	0.81	0.215
linear	outliers	split	kmeans	48.0	16.6676	1.0	0.0116	0.9993	0.762	0.32
linear	outliers	split	kprototypes	42.0	26.6667	0.9912	0.0033	1.0	0.21	0.0867
linear	outliers	keep	kmeans	41.0	13.6278	0.787	0.0144	0.9967	0.475	0.2909
linear	outliers	keep	kprototypes	6.0	22.8333	0.0919	0.0093	1.0	0.23	0.1257
linear	outliers	keep_reset	kmeans	34.5	14.2682	0.8212	0.0244	1.0	0.82	0.525
linear	outliers	keep_reset	kprototypes	8.0	28.75	0.1371	0.0151	1.0	0.28	0.1442
rbf	inliers	split	kmeans	193.5	13.7777	1.0	0.0034	0.9726	0.275	0.214
rbf	inliers	split	kprototypes	972.0	29.1224	0.9694	0.0007	0.9639	0.61	0.117
rbf	inliers	keep	kmeans	133.0	14.5853	0.6604	0.0037	0.9647	0.28	0.3188
rbf	inliers	keep	kprototypes	3.0	15.8334	0.1808	0.0178	0.9686	0.37	0.1662
rbf	inliers	keep_reset	kmeans	47.0	11.2857	0.5564	0.0052	0.957	0.23	0.3462
rbf	inliers	keep_reset	kprototypes	3.0	20.0	0.3185	0.0192	0.9682	0.32	0.1623
rbf	outliers	split	kmeans	120.5	12.9288	1.0	0.0074	0.9993	0.89	0.1744
rbf	outliers	split	kprototypes	164.5	22.015	0.5335	0.002	1.0	0.31	0.089
rbf	outliers	keep	kmeans	74.5	15.8889	0.4261	0.0131	1.0	0.84	0.128
rbf	outliers	keep	kprototypes	8.0	11.3846	0.124	0.0085	1.0	0.32	0.2208
rbf	outliers	keep_reset	kmeans	60.0	16.5507	0.9767	0.0116	0.9993	0.89	0.173
rbf	outliers	keep_reset	kprototypes	8.0	21.625	0.2043	0.0357	1.0	0.33	0.2441

Table 4: Metrics for the three clustering-based methods. The results are aggregated for each dataset by their median value

kernel	type_points	method	n_rules	size_rules	per_p1	p1_cov	rule_ag	p_vs_m	score_int
linear	inliers	Anchors	1.0	10.167	0.1189	0.007	0.9732	0.49	0.5067
linear	inliers	DT	9.0	17.0	0.9472	0.0026	1.0	0.91	1.0
linear	inliers	FRL	0.0	0.0	0.0	0.0	0.0	0.0	0.0
linear	inliers	RuleFit	28.0	9.75	0.7896	0.0039	1.0	0.74	0.605
linear	inliers	SkopeRules	32	15.1	0.3184	0.032	1.0	0.75	0.422
linear	inliers	brlg	0.0	0.0	0.0	0.0	0.0	0.0	0.0
linear	inliers	logrr	0.0	0.0	0.0	0.0	0.0	0.0	0.0
linear	outliers	Anchors	6.0	9.3333	0.371	0.1276	1.0	0.285	1.0
linear	outliers	DT	10.0	17.5	0.6882	0.0038	1.0	0.42	0.331
linear	outliers	FRL	1.0	8.0	0.0395	0.0041	0.9241	0.1	0.085
linear	outliers	RuleFit	16.5	8.7	0.8794	0.0321	1.0	0.415	0.8772
linear	outliers	SkopeRules	2.0	14.0	0.0536	0.0382	1.0	0.625	1.0
linear	outliers	brlg	0.0	0.0	0.0	0.0	0.0	0.0	0.0
linear	outliers	logrr	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rbf	inliers	Anchors	4.0	9.0	0.0314	0.0002	0.9022	0.425	0.745
rbf	inliers	DT	2.0	9.5882	0.1272	0.0016	0.94	0.64	1.0
rbf	inliers	FRL	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rbf	inliers	RuleFit	14.0	9.1429	0.6241	0.0114	0.895	0.56	0.7021
rbf	inliers	SkopeRules	1.0	0.0	0.0013	0.0013	1.0	0.19	1.0
rbf	inliers	brlg	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rbf	inliers	logrr	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rbf	outliers	Anchors	7.5	14.2	0.0132	0.0018	0.9802	0.355	0.573
rbf	outliers	DT	5.5	10.853	0.0839	0.01	0.9884	0.615	1.0
rbf	outliers	FRL	4.5	9.5	0.0723	0.0436	1.0	0.745	0.3756
rbf	outliers	RuleFit	31.0	7.6855	0.85	0.0432	0.9722	0.795	0.7842
rbf	outliers	SkopeRules	6.0	3.0	0.2647	0.0647	1.0	0.84	1.0
rbf	outliers	brlg	0.0	0.0	0.0	0.0	0.0	0.0	0.0
rbf	outliers	logrr	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5: Metrics for the other methods. The results are aggregated for each dataset by their median value

dataset	kernel	type_points	n_rules	size_rules	per_p1	p1_cov	rule_ag	p_vs_m	score_int
1	linear	outliers	2	4.5	0.5683	0.2888	1.0	0.75	1.0
1	linear	inliers	1	4.0	0.4696	0.4696	1.0	0.49	1.0
1	rbf	outliers	0	0.0	0.0	0.0	0.0	0.0	0.0
1	rbf	inliers	2	6.0	0.5622	0.4758	0.955	0.44	1.0
2	linear	outliers	6	3.8333	0.1941	0.0676	1.0	0.5	1.0
2	rbf	inliers	0	0.0	0.0	0.0	0.0	0.0	0.0
2	rbf	outliers	15	24.0667	0.1628	0.0116	0.9955	0.5	0.368
2	linear	inliers	0	0.0	0.0	0.0	0.0	0.0	0.0
3	rbf	inliers	4	9.0	0.3473	0.0461	0.854	0.41	0.49
3	linear	outliers	6	9.3333	0.548	0.1876	1.0	0.07	0.3267
3	linear	inliers	6	10.1667	0.6733	0.2657	1.0	0.95	0.5067
3	rbf	outliers	5	11.2	0.0638	0.0142	1.0	0.71	0.778
4	rbf	outliers	11	64.0909	0.0245	0.0033	1.0	0.84	1.0
4	rbf	inliers	154	61.9351	0.0557	0.0004	1.0	0.56	1.0
4	linear	outliers	6	3.833	0.1941	0.0676	1.0	0.5	1.0
4	linear	inliers	0	0	0	0	0	0	0
5	linear	inliers	1	13.0	0.0005	0.0005	0.0	0.0	0.0
5	linear	outliers	43	17.2326	0.0093	0.0002	0.9502	0.47	1.0
5	rbf	outliers	10	17.2	0.0019	0.0002	0.965	0.21	1.0
5	rbf	inliers	30	17.3333	0.007	0.0001	0.9503	0.59	1.0

Table 6: Metrics for Anchors. The results are aggregated for each dataset by their median value

kernel	type_points	method	clustering	final_metric
linear	inliers	split	kmeans	7.9694
linear	inliers	split	kprototypes	3.1295
linear	inliers	keep	kmeans	6.4058
linear	inliers	keep	kprototypes	2.8717
linear	inliers	keep_reset	kmeans	7.9369
linear	inliers	keep_reset	kprototypes	2.2336
linear	outliers	split	kmeans	9.8991
linear	outliers	split	kprototypes	2.8709
linear	outliers	keep	kmeans	16.494
linear	outliers	keep	kprototypes	2.5456
linear	outliers	keep_reset	kmeans	17.7
linear	outliers	keep_reset	kprototypes	1.8213
rbf	inliers	split	kmeans	7.7021
rbf	inliers	split	kprototypes	1.4172
rbf	inliers	keep	kmeans	7.49
rbf	inliers	keep	kprototypes	7.5773
rbf	inliers	keep_reset	kmeans	45.1546
rbf	inliers	keep_reset	kprototypes	4.1603
rbf	outliers	split	kmeans	15.1275
rbf	outliers	split	kprototypes	2.7815
rbf	outliers	keep	kmeans	7.9909
rbf	outliers	keep	kprototypes	151.4028
rbf	outliers	keep_reset	kmeans	9.5672
rbf	outliers	keep_reset	kprototypes	3.523

Table 7: Aggregated results into one metric for clustering-based methods.

kernel	type_points	method	final_metric
linear	inliers	Anchors	3.4075
linear	inliers	DT	3.0591
linear	inliers	FRL	
linear	inliers	RuleFit	2.1491
linear	inliers	SkopeRules	2.6287
linear	inliers	brlg	
linear	inliers	logrr	
linear	outliers	Anchors	3.8295
linear	outliers	DT	1.8471
linear	outliers	FRL	2.3553
linear	outliers	RuleFit	3.1122
linear	outliers	SkopeRules	3.1426
linear	outliers	brlg	
linear	outliers	logrr	
rbf	inliers	Anchors	3.2702
rbf	inliers	DT	4.4232
rbf	inliers	FRL	
rbf	inliers	RuleFit	2.867
rbf	inliers	SkopeRules	8.4063
rbf	inliers	brlg	
rbf	inliers	logrr	
rbf	outliers	Anchors	1.8258
rbf	outliers	DT	3.3818
rbf	outliers	FRL	3.2506
rbf	outliers	RuleFit	2.3935
rbf	outliers	SkopeRules	5.8055
rbf	outliers	brlg	
rbf	outliers	logrr	

Table 8: Aggregated results into one metric for the remaining rule extraction methods. Empty results indicate combinations that do not yield P@1 rules.