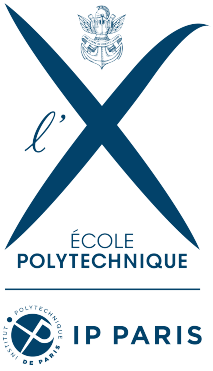



**École Polytechnique**

*BACHELOR THESIS IN COMPUTER SCIENCE*

# A very long and informative title for my thesis work

*Author:*

Remi Guillou, École Polytechnique

*Advisor:*

Yanlei Diao, École Polytechnique

*Academic year 2024/2025*

**Abstract**

The increasing amount of data being created end processed in all sectors has led to the use of automatic methods for filtering and analysing this influx of information. These methods often rely on complex models whose methods are intractable and as such act as "Black Boxes". Such techniques are used for monitoring traffic on many different types of servers. Anomaly detection methods are important as they make it possible to know in real time when an issue has occured. However, in order to efficiently resolve any issue, as important as detecting an anomaly is understanding why the flagged interval is anomalous. This is where the field of Explainable AI comes into play. In this paper, we will improve Exstream, which is a method relying on the single feature entropy measure between normal points and anomalous points to determine features that contribute the most to the anomaly and the feeature intervals where anomalies would happen. Our contribution is twofold, first we explored the possibility of sampling normal points using the latent space from the detection method. These points would be more representative of the current anomaly and wouldn't be correlated with time. We found that the domain dependency of the data makes this technique unfeasable for our type of data. Secondly, we showed that the scoring for different features isn't a submodular function due to correlated features but is simply increasing. We also generalized the entropy measure over multiple features which would enable the explanation of anomalies on more complex dataset where the interaction of two or more features is responsible for the anomaly.

# Contents

# 1 Introduction

# 2 Related works

# 3 The Exathlon project

The Exathlon project was started in 2021 as an anomaly detection and explanation benchmark [2]. The github repository implements a pipeline that implements all needed steps from preprocessing of the data to training and evaluating methods for detecting and explaining anomalies. The full pipeline can be seen in Figure 7.

## 3.1 Dataset

The pipeline supports any dataset and custom preprocessing steps can be implemented. For the sake of this paper we will be using a time series datasets consisting of traces taken from Apache Spark servers. These traces consist of real data collected from 93 repeated executions of 10 distributed streaming applications on a 4-node Spark cluster over a period of 2.5 months. Each of these executions includes 5 randomly selected applications running concurrently. In total, 2,283 metrics were monitored once per second creating a dataset totaling more than 24GB in size. For the sake of our experiments, the dataset went through some preprocessing, cutting it down to 237 features. It can be noted that this preprocessing step is done automatically in the pipeline. The dataset consists of 59 undisturbed traces and 34 disturbed traces constituting 97 anomaly intervals. There are 6 types of anomalies:

- T1: bursty input
- T2: bursty input until crash
- T3: stalled input
- T4: CPU contention
- T5: driver failure
- T6: executor failure

The data is processed and then turned into a *window dataset* which transforms the data into a sliding window format in order to detect the anomalies.

## 3.2 Anomaly detection

The anomaly detection consists in three steps. First training the *window model*. This model gives an anomaly score to a window of a certain size. Then following the "Unifying anomaly detection method" introduced by Vincent Jacob [1], we obtain an "Online" anomaly scoring function which given the scoring for windows, attributes a score for each point. This scoring function is used to determine a threshold above which points will be labelled as anomalies in the anomaly detection step.
Many different models belonging to various families of methods [7] are implemented and ready to be evaluated. Such methods include among others: pca, xgboost, Autoencoder, Variational Autoencoder, LSTM, deep SVDD, deep SAD, iForest...
The most recent and best model on this dataset is Divad.

### 3.2.1 Divad

Divad is an anomaly detection method based around a VAE architecture [1]. This method was create to address the difference in the behaviour of traces based on the parameters and properties of the Spark application. We will call "Domain" the context in which the application is run. This context is characterised by the following elements: processing period, number of active executors, memory profile and input rate.

The different domains pose a real problem when trying to identify anomalies. A certain behavior can be considered anomalous in one domain but not in another. ‘add plot‘. The limited amount of data also makes it impossible to train a model for each domain. Therefore we need a method that can detect anomalies and generalize to new domains. This is Divad.

The way it works is by assuming two independent priors that define the data z. The first prior is the class y, either anomaly or normal. The second prior is one that encodes the domain of the point. Therefore we are assuming that the points can be generated from their class and their domain and that these two priors are independent.
‘insert image‘ The objective is thus to approximate these prior spaces from the data. In order to do so, Divad employs a Variational Autoencoder structure with two independent encoders and one decoder. The first encoder is meant to capture the Class information and the second the Domain information. This is done using well chosen error functions.
We therefore end up with two latent spaces: $z_y$ and $z_d$ that contain respectively information about the class and the domain of the data. We then use these two latent space to reconstruct the data.
We get an anomaly score by comparing the class latent space with what we would expect from the normal class. This is done by computing the KL divergence between the two distributions.

## 3.3 Anomaly Explanation

The anomaly explanation step will be the main focus of this paper. The objective of this step is to give a human readable and actionable explanation of the anomaly. This can be done in many ways such as lists of feature importance or plots as we have seen in the related works. In this paper, the focus will be on the Exstream method. For each anoamaly, this method will output the features that cause the anomaly as well as the intervals where the anomaly is happening.
The explanation is given as a boolean expression in Conjunctive Normal Form. It consists of a conjunction of clauses, where each clause is a disjunction of predicates. Each predicate follows the form {v o c}, where v represents a feature, c is a constant, and o is one of the five operators: $o \in \{>, <, \geq, \leq, =\}$.
For example an explanation for an anomaly could be the following: (feature1 > 0.5) $\vee$ (feature1 < 0.8) $\wedge$ (feature2 = 0.7).
There previously existed two versions of Exstream. The original version was introduced by Haopeng Zhang, Yanlei Diao and Alexandra Meliou in 2017 [8]. It was then work uppon by Mija Pilkaite during her Bachelor thesis [5] and finally refined and improved by Clement Martineau [3].

### 3.3.1 Original Exstream

Originally, Exstream was created to provide a rigorous and formal method for providing short and human readable explanations for anomalies. This method presents the challenge of finding the optimal explanation to an anomaly as a submodular optimization problem. The intuition behind this model is that adding features to our explanation has a diminishing return. This is because the more features we add, the less information new features will provide.

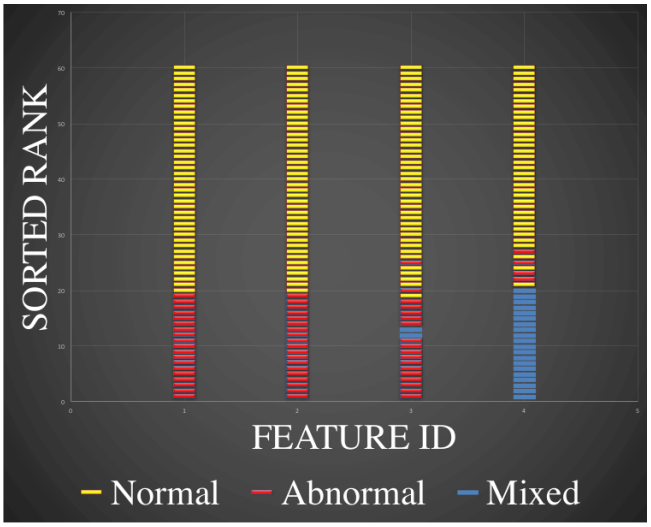



Figure 1: Visualization of the segments of 4 features. The red points are the anomalous points,the yellow points are the normal points and the blue are mixed. Segments are continuous points of the same colour.

Submodular optimization being NP-hard, a heuristice was therefore introduced to solve the problem. This method relies on Entropy to compute single feature scores.

We start with two sets of points $S_a$ and $S_n$ containing respectively the anomalous and normal points. For each feature $f_i$, we will sort the points in $S_a$ and $S_n$ and merge them into one set. We will then define segments on this set as neighbouring points of the same class.

We will compute a segmentation entropy defined as follows: If there are n segmentations, and $p_i$ represents the ratio of data points included in the ith segmentation, the segmentation entropy is:

$$H_{segmentation} = \sum_{i=1}^{n} p_i \log(\frac{1}{p_i}) \tag{1}$$

Exstream was originally aimed at categorical data. Therefore there can be features with values that belong both to anomaly and to normal points. We need to penalize these features. This is done by giving the worst score to those points of mixed class. Let $c_i$ be a mixed interval and $c_i^*$ be the segment rearanged in its worst ordering.
Then the maximum entropy is defined as:

$$H_{max} = \sum_{i=1}^{n} H_{segmentation}(c_i^*) \tag{2}$$

Finally, we normalize the score by the class entropy to get a values between 0 and 1. The class entropy is defined as follows: Let $|S_a|$ and $|S_n|$ be the number of points in $S_a$ and $S_n$ respectively. $p_a = \frac{|S_a|}{|S_a|+|S_n|}$ and $p_n = \frac{|S_n|}{|S_a|+|S_n|}$. The class entropy is then:

$$H_{class} = p_a \log(\frac{1}{p_a}) + p_n \log(\frac{1}{p_n}) \tag{3}$$

The final score for a feature is then:

$$score(f_i) = \frac{H_{segmentation} + H_{max}}{H_{class}} \tag{4}$$

This score reflects the segmentation of the feature with respect to anomaly and normal classes. The higher the score, the more separated normal and anomaly classes are. For example a feature with a score of 1 would have all the anomaly points in one segment and all the normal points in another. As we can see on Figure 1, the first two features have a score of 1. The third a score of 0.31 and the fourth 0.18.
Exstream selects the best features based on reward leap filtering. It also provides methods to filter out redundant features using correlation clusters. It also removes "false positive" features which are defined as those with excessively large standard deviation or that tend to only increase or decrease on both the normal and anomalous interval.

### 3.3.2 Exstream using Bins

An attempt to improve Exstream was made by Mija Pilkaite during her Bachelor thesis [5]. The first idea was to use bins to discretize the data. This would make the method more suited for continuous data. Each feature values would be divided into b bins. Each bin would be marked as either Normal, Anomalous or Mixed. The entropy would then be computed on these bins instead of points. The second idea was to reduce uncertainty in the anomalous interval boundaries. Even when using ground truth segments with anomalous intervals marked by hand, there in always some uncertainty as to exact endpoints of the intervals.
In order to minimize this labeling uncertainty, the points would have different weights in the computation of the entropy depending on their position in the trace. This weighting would be done using a Gaussian distribution centered around the middle of the interval as such:

$$w_t = \begin{cases} \exp\left(-\left(\frac{|position - \frac{interval\_length}{2}|}{\sigma \times \frac{interval\_length}{100}}\right)^{\beta}\right), & \text{if } 0 \leq position \leq interval\_length \\ 0, & \text{otherwise} \end{cases}$$

This was further worked on by Clement Martineau who fixed some issues and made the method work better in higher dimensions.

### 3.3.3 Metrics

The Exathlon pipeline provides a set of metrics to evaluate the performance of the explanation models. These metrics are the following:

1. ED1: These metrics are based on local behavior of the anomaly explanation.
    - Time: The time taken to compute the explanation.
    - Size: The size of the explanation. This is the number of features contained in the explanation.
    - Perturbed Size: Size of the explanation after perturbation. A perturbation is defined as a random sampling of 80% of the data in the normal and anomalous intervals. The perturbed size is the number of features in the explanation of a trace having been perturbed.
    - Instability: A measure of the difference in the explanation before and after a perturbation. A high instability means that the explanation isn't locally stable.
    - F1 Score, Precision, Recall: These metrics are computed by getting an explanation on a randomly sampled 80% of the normal and anomalous dat and then testing the explanation on the remaining 20%. The testing is done on the intervals being returns as being anomalous. From them a Precision and Recall score are computed. The F1 Score is the harmonic mean of the two. Similarly as for the instability, the random sampling and evaluation are carried out 5 times and the average is taken.

2. ED2: These metrics are based on global behavior of the anomaly explanation.
    - Discordance: This is the degree of disagreement between the explanation for anomalies of the same type. This measure is computed for each type of anomaly as the entropy of the number of each feature found in all explanations of this type. Then taking two to that power and normalizing by the average number of features in the explanations.
    - F1 Score, Precision, Recall: these metrics are computed by getting an explanation on one sample and evaluating it on all the other test samples. Similarly as for ED1, these scored are computed on the intervals being returned as being anomalous. The F1 Score is the average of the F1 Scores computed on each other test sample.

These metrics are all computed for each type of anomaly as well as for the whole dataset.
It is important to note the limitations of these metrics. For anomaly explanation contrary to anomaly detection, there is no ground truth. Therefore getting perfect Discordance and Stability does not imply that the explanation is good. For example a method returning *feature1* as an explanation every time would get perfect scores for both metrics. This is especially important for explanation methods that do not provide intervals but only important features. This is why inspecting specific examples and determining if the explanation provided is relevant is important.
Another limitation comes from ED2 F1 score, Precision and Recall. As stated in subsubsection 3.2.1, the domain of the trace being considered is important. A point that could be considered an anomaly in one domain could be normal in another. Therefore an interval being selected as an explanation in one domain could be considered normal in another. The fact that we compare explanations against all other test samples means that we are comparing explanations from different domains. This property of our data significantly undermines the importance of these metrics. We can even go further and state that the existence of different domains in our data undermines the global consistency of our anomaly explanator, at least when it comes to intervals.

Another limitation that stems from these metrics is that quantifying the importance of each is difficult. Especially when considering metrics such as Discordance and Instability. The way these metrics is computed is arbitrary and their interpretation is not straightforward. This is especially true for Discordance when being normalize, we raise two to the power of the entropy score and divide by the average number of features in an explanation. These manipulations cloud the interpretation of the metric and the way it would change when adding or removing features.

Overall, these metrics are very useful in comparing different methods. However, they should be taken with a grain of salt and the results should be interpreted with caution
*** add metrics for exstream and exstream bin and say that the bin improvement is useless ***

# 4 Objectives

The focus of this paper will be to improve Exstream. The version we intend to work on will be the original version. This choice is motivated by a few reasons.
First, looking at the metrics and inspecting a few examples shows that the version using bins does not offer any significant improvements. Secondly, the time to run is multiplied over x times. This performance issue is partly caused by unoptimized code and partly by an overcomplicated algorithm. The algorithm was also never previously evaluated as part of the Exathlon pipeline and only on a subset of the available data. The complexity of the algorithm is also shown by edge cases causing errors on specific traces when evaluating it on the entirety of the Exathlon data.
The complexity of the algorithm makes it very rigid and hard to modify in relevant ways while at the same time the runtime is too long to be used in a real time setting.

There are two ways in which we can improve Exstream. The first is to improve the sample set. The second is to improve the algorithm being used. In this paper we will address both of these points with varying degrees of success.

# 5 Improving the sample set

A problem noticed when evaluating Exstream on the Exathlon data is that the normal points used are not necessarily the most representative of the current anomaly. This is due to the fact that the normal points selected are right before the anomaly. There is therefore a strong correlation between these points and time. The most obvious examples of this are features that are only increasing or decreasing over the normal and anomalous intervals. The segmentation of these features will be perfect and the score will be 1 while these features aren't relevant explanations for the anomaly.
This specific case of time correlation is already addressed in the original Exstream paper using false positive filtering. However less obvious cases of correlation are not addressed and can lead to wrong explanations.
At the same time we will attempt to get sample of normal points "closer" to the anomaly. This sample should be more representative of the current anomaly and only the relevant features that induced the anomaly will significantly change. This should enable a better explanation.

## 5.1 Sampling using the latent space

The latent space of a VAE has a well defined structure. The normal points will be distributed following a normal distribution. The space is also continuous, at least close to the normal point distributions. The encoder has been trained to give the space other properties. The normal points will be close to each other and the anomaly points will be further away. This is how the anomaly score is defined. By sampling normal points that are close to the anomalies in this space, we hope to obtain points that are more representative. In that way, only features that contributed to the anomalous behavior will change.

### 5.1.1 Reconstuction

### 5.1.2 Similar domains

# 6 Generalizing Exstream to Higher dimensions

As seen earlier, exstream computes entropy on a single feature. This is a limitation as anomalies can be caused by the interaction of two or more features. This method was initially given as a way of finding an approximation to the problem of finding the best set of features to explain an anomaly. This problem was believed to be submodular.
In this section we will start by showing that this problem isn't submodular. We will then present multiple ways the entropy measure can be generalized to multiple dimensions. We will then evaluate these methods and empirically show benefits that are derived from using them.

### 6.1 Submodular optimization

Let $A$ be a set and $\mathbb{A} = \mathcal{P}(A)$ be its powerset. The set function $f : \mathbb{A} \to \mathbb{R}$ is submodular if $\forall X, Y \subseteq \mathbb{A}$,

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$$

It is also equivalent to saying $\forall X \subseteq Y \subseteq \mathbb{A}, \forall x \in \mathbb{A} \setminus Y$:

$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$$

The submodularity of a function reflects the diminishing return of adding elements to a set. This property is very useful in optimization problems as it allows for the use of greedy algorithms to obtain an approximate of the solution [4].
This property has been shown to occur in feature selection problems but only in the case where features are independent [6]. This is not the case for our data.
Let us show visually that the submodularity property does not hold. We start by formally defining our setting: We denote the features by numbers from 1 to 237. Let $D = [1, 237]$ be the set of all features. Then $\mathbb{D} = \mathcal{P}(D)$ be the powerset of $D$. We further annotate each interval of Anomalous and Normal points $(N_i, A_i)$ by a number $i$. We then define the function $f_i : \mathbb{D} \to [0, 1]$ as a function computing the entropy of the segmentation of the features being passed for the pair $(N_i, A_i)$.
We need the function $f_i$ to be submodular for each pair $(N_i, A_i)$. Therefore let us present an example for which this property does not hold, see Figure 2. Here in either dimension there will be very high segmentation. However, considering both features we see a clear separation between the anomalous and normal points. Using the tree method for multi feature scoring we respectively get 0.012, 0.012 and 0.55 considering $[f_1]$, $[f_2]$ and $[f_1, f_2]$.
Therefore the submodular property does not hold as $f([f_1]) + f([f_2]) < f([f_1] \cup [f_2]) + f([f_1] \cap [f_2])$.
This example was specifically chosen but we see that in practice the submodular property is also broken. This

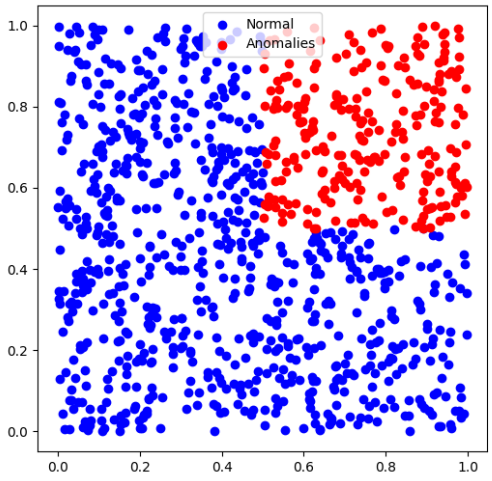



Figure 2: Plot of two features $f_1$ and $f_2$. Blue points are normal $N_i$ and red points are anomalies $A_i$.

property depends on the function being used to compute the entropy. Meaning that with a well chosen function it is possible that the submodular property holds. However, this function would loose a lot of information and would not choose the optimal selection of features.
In conclusion we have proven that the submodular property does not hold for our data. This implies that we do

not have an efficient way of finding the best set of features even with a multi dimensional entropy measure.

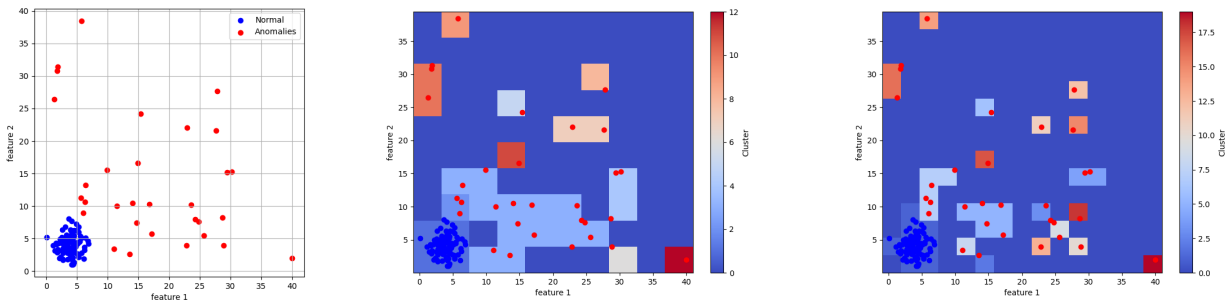
## 6.2 Methods for generalizing Entropy

Generalizing the computation of entropy to higher dimensions isn't trivial. The measure of entropy is done after creating clusters. In one dimension this is simply done by taking segments containing the same type of points. The number of points in each segment is then used to compute the entropy.
In higher dimensions we wish to keep a similar method. In order to do so we will thus only need to determine the optimal clustering method to be used.
There are certain properties that our clustering method must have in order to make sure it works properly.

- It must obtain a similar result to Exstream when applied to a single feature.
- It must classify discontinue points into different clusters. Meaning that if there exists an anomalous point in between two normal points, these two normal points should be in different clusters.
- The method must generalize to higher dimensions. The number of dimensions can be limited to $\approx 10$ as explanations above this would be too long and wouldn't be useful.
- The method must fit any distribution of the data and work on few points.
- It must be possible to find intervals for the anomalies being clustered.
- The computational complexity of the method must not be "too" high. This is a vague requirement but it is important to keep in mind that the method will be used in a real time setting and will need to be evaluated on many sets of features. An exponential growth in complexity in the number of points would for example make the method unusable.

We will now present a few methods that could be used to cluster the data.

### 6.2.1 Boxes



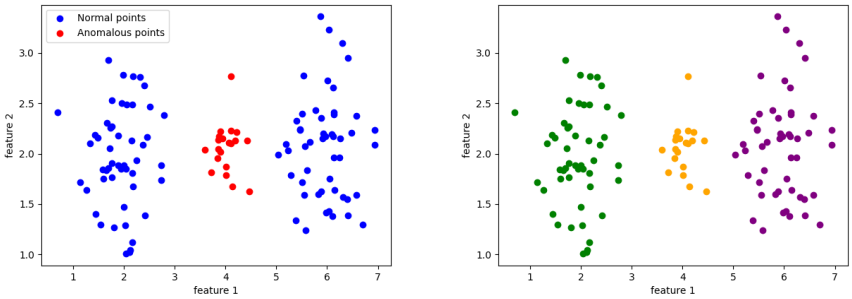

(a) Points in the space (b) Clustering using 10x10 bins (c) Clustering using 15x15 bins

Figure 3: Plot of points being clustered by using the Box method. Same coloured squares are in the same cluster.

The first method is inspired by the bin method for Exstream [5] presented above. The idea of dividing the feature space into bins to discretize the data can be easily generalized to higher dimensions. We would use n dimensional cubes to divide the space. We would then check the points in each cube and notate them as either

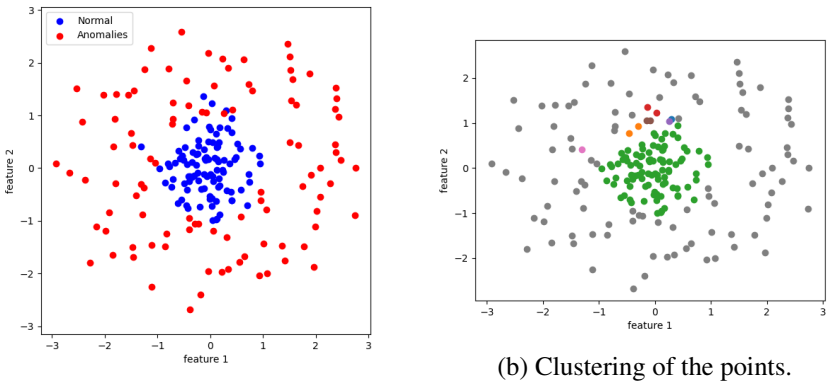
normal, anomalous or mixed. Having done that, we would then merge adjacent cubes of the same type unless they are mixed in which case it isn't usefull to merge them. This would give us the clusters we need to compute the entropy.
This method has the merit of being simple and computationally efficient running in $O(n)$ time with n being the number of points considered. However, this method does not generalize well to higher dimensions and to different distributions of the data.
For example in the case with outlying points, as it is expected with anomalous data, the method would not be able to correctly merge the cubes. This would lead to a high number of clusters and a high entropy. The clusters are also highly dependent on the number of bins. The ideal number is impossible to know in advance and can be different per dimension and per trace.
Overall this method can work on simple data but isn't resilient enough for our use.

### 6.2.2 Nearest neighbours




(a) Points in the space (b) Clustering of the points.

Figure 4: Plot of points being clustered by using the nearest neighbours method. Same coloured points are in the same cluster.

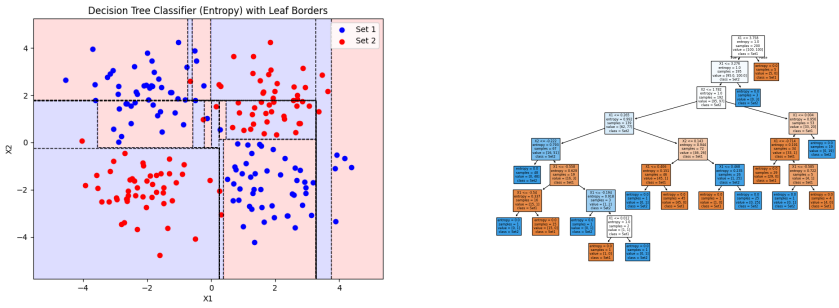



(a) Points in the space (b) Clustering of the points.

Figure 5: Plot of points being clustered by using the nearest neighbours method. Same coloured points are in the same cluster.

This method can be applied to single points or as an improvement to the post processing of the box method. It relies on the nearest neighbours of each points and is inspired directly by the property stating that we must "classify discontinue points into different clusters".
The method is as follows. For each points, we will iteritavely get it's next nearest neighbour until reaching a point of the other class. We will cluster these points together. On top of that merge clusters if a point is already part of a cluster. This method makes it so if two clusters are discontinue, meaning a point of the other class is placed in between them, they won't be merged together. See Figure 4.
In practice we will use a KDtree for storing points and efficiently get the nearest neighbours. We will also restrict the maximum number of neighbours being considered in order to improve the efficiency.
Let $n$ be the number of points and $k$ a hyper parameter being the maximum number of neighbours. The KDtree is created in $O(n\log(n))$, the nearest neighbours are then computed in $O(kn\log(n))$ and the clustering is done in $O(n)$. Overall the method runs in $O(n\log(n))$ time. The method is therefore efficient and can be used in real time.
The method obtains the same results as Exstream on a single feature. It also generalizes well to higher dimensions and different distributions of the data as only the segmentation thereof is considered when clustering. The method is relatively stable with respect to the hyper parameter $k$, the only case where it would matter is for a group of isolated points of size greater than k. Those could be clustered together but not with the rest of the group.
The big issue with this method is the lack of intervals being determined. We only get points in clusters. These clusters aren't based on intervals. For example Figure 5 shows a case where the gray class doesn't translate to a clear interval. This is an issue as intervals strengthen the explanation and give much more information about the anomaly.

### 6.2.3 Decision Trees




(a) Clustering of points.

(b) Corresponding decision tree.

Figure 6: Clustering of points and corresponding decision tree. Red points are anomalous and blue are normal. The lines mark the separation between clusters.

The final method we will present is the use of decision trees to cluster points. The idea is to train a decision tree on the two sets of points we have. We then use the leaves of the tree as clusters. Intuitively, a decision tree splits the space into hypercubes. Each node splits the space in two along a certain feature. The leaves are a conjunction of these splits, in essence creating a hypercubes subset of the space being considered.
In theory, the decision tree can split the space until each leaf is pure, i.e it contains only points of the same class. This would give us the optimal clustering. However, in practice we will allow the tree to split until having at minimum 10 points in each leaf. Mixed clusters are then handled similarly as in Exstream and given the maximum segmentation score. This is done for efficiency purposes. It is also important to note that this limitation does not impact significantly the results. Reaching this limit would mean that the points are very

segmented and that the entropy would be high in any case.
Now let us go over the properties of this method:

- The results are the same as Exstream when run on a single feature if we do not take into consideration the 10 point approximation when the points are too segmented.
- Points are classified using Hypercubes. Therefore two points being classified in the same cluster means there are no points of the other class in between them. At the same time, two normal clusters with anomalous points in between cannot be classified together as the hypercube containing them would be impure.
- As most models, decision trees suffere from the curse of dimensionality. However in our case the number of dimensions considered is low and the method would work fine.
- A decision tree can fit any distribution. In our case we do not care about overfitting as we will not make predictions with it and only use it to cluster points.
- As stated previously, Decision trees create hypercubes. This in effect gives us intervals.
- We fit a decision tree on the data. This is done in $O(mn\log(n))$ with m being the number of dimensions and n being the number of points. In practice, the algorithm runs slightly faster than the nearest neighbor one. Overall the time complexity isn't too high.

This clustering method checks all of the properties we needed making it a good candidate for our use. The construction of the decision tree case be done using Gini impurity or entropy. We will use entropy as this creates a link between the creation of the tree and the measure we are trying to compute. At each node of the tree, the best split is created by minimizing the entropy of the two leaves created. This suits us as we are trying to minimize the entropy of the clusters.
The use of hypercubes for clustering provides intervals for our explanation. However, this implies that some points that are adjacent will fall in multiple clusters. For example in Figure 2, the normal points will be split into two clusters. This can either be seen as a limitation or as a strength of this method. This penalizes the use of multiple dimensions when explaining the anomaly. Higher dimensions implie more cuts. This will prevent the incorporation of irrelevant features as these would either decrease the entropy or not change it. A byproduct of this is that getting a score of 1 for n features implies that there is a feature alone in that set that gets a score of 1. i.e let $g_i$ be the entropy function on segment $i$. $f_j$ is a feature for $j \in [1, n]$.:

$$g_i([f_1, f_2, ..., f_n]) = 1 \Rightarrow \exists j \in [1, n], g_i([f_j]) = 1$$

This is due to the fact that getting an entropy score of 1 implies that the points are perfectly segmented, thus finding 2 clusters. Using decision trees this implies a single cut along one dimension. Therefore the feature on which this cut was done is enough on its own to perfectly segment the points.
It is also possible to overcome this limitation by merging neighbouring leaves into one cluster. The intervals would still be given separately but the clusters would be considered as one, lowering the overall entropy. This would be done by checking the neighbouring leaves and merging them if they contain points of the same class. This would be done iteratively until no more merges are possible.
Decision trees also do not have any garantees of finding the best clusters. The method is greedy and will only split the space along the best feature at each node. This can lead to suboptimal clustering as for example in Figure 6, the normal points on the right are split off early and are separated from the rest whereas they could have been clustered together. However, in practice the method works well and the results are satisfactory.

### 6.3 Incorporation into Exstream

We will now see how to incorporate any method mentioned above for computing entropy into Exstream. The intuitive way would be finding the set of features that obtains the best score through our method. However, as shown previously, finding such a set of features isn't a submodular optimization problem. We therefore do not have an efficient way of finding the best set of features. There might be other ways of finding the best set of features using properties of the different functions. However, in this paper we will not explore these. And will limit ourselves to brute forcing the problem in two dimensions.
Another cause to motivate this decision of keeping the number of features considered to two is the complexity of our dataset and of the anomaly types. There are many traces where a single feature alone has perfect separation between the normal and anomalous points getting an entropy score of 1. Moreover, some traces contain multiple such features that are not heavilly correlated. In these cases it is useless to search for a combination of feautres as a single feature alone has a perfect score. It is also above the scope of this work to improve the selection of features to be returned in the above case. We will therefore rely on the same method as Exstream and when a feature gets a score above a certain threshold, we will not search in two dimensions and return all features with a score above that threshold subject that they are not correlated and that they are not false positives.
The returned format for the anomalous intervals will be the same as in the original exstream with $I_j^i$ intervals:

$$((f_1 \in I_1^1 \vee f_1 \in I_2^1 ...) \wedge (f_2 \in I_1^2 \vee f_2 \in I_2^2 ...))...$$

The case where no single feature is enough to perfectly segment the normal and anomalous points, we will brute force the problem in two dimensions. After removing false positive feature, we will compute the entropy for all pairs of features and return the pair that has the best score. Considering there are initially 237 features in the dataset, in the worst case (when there aren't any false positive features), we will be computing the entropy for $\binom{237}{2} = \frac{237 \times 236}{2} = 27966$ pairs of features. This is a lot of computation especially considering the complexity for each pair is $O(n \log(n))$.
When using the decision tree method, we will obtain hypercubes as intervals. These hypercubes can be defined as the conjunction of intervals in each feature. Therefore the return format will be the following:

$$(f_1 \in I_1^1 \wedge f_2 \in I_1^2) \vee (f_1 \in I_2^1 \wedge f_2 \in I_2^2)...$$

This inversion between $\wedge$ and $\vee$ provides much more information about specific intervals where points are anomalous. It is also possible to return multiple pairs as explanation. These would be linked using $\wedge$ clauses.

## 7 Conclusion

# 8 References

[1] Vincent Jacob and Yanlei Diao. Unsupervised anomaly detection in multivariate time series across heterogeneous domains. *PVLDB*, 14(1):XXX–XXX, 2025.

[2] Vincent Jacob, Fei Song, Arnaud Stiegler, Bijan Rad, Yanlei Diao, and Nesime Tatbul. Exathlon: A benchmark for explainable anomaly detection over time series. *PVLDB*, 14(11):2613–2626, 2021.

[3] Clement Martineau. Advanced algorithm design for explainable anomaly detection on data streams. Internship report, Laboratoire d'Informatique de l'École Polytechnique (LIX), Team CEDAR (LIX - INRIA), July 2024.

[4] Rad Niazadeh, Tim Roughgarden, and Joshua R. Wang. Optimal algorithms for continuous non-monotone submodular and dr-submodular maximization. *Journal of Machine Learning Research*, 21:1–31, 2020.

[5] Mija Pilkaite. Algorithm design for explainable anomaly detection for data streams, 2023/2024.

[6] Bharath Sankaran, Marjan Ghazvininejad, Xinran He, David Kale, and Liron Cohen. Learning and optimization with submodular functions. *arXiv preprint arXiv:1505.01576*, 2015.

[7] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: A comprehensive evaluation. *PVLDB*, 15(9):1779–1797, 2022.

[8] Haopeng Zhang, Yanlei Diao, and Alexandra Meliou. Exstream: Explaining anomalies in event stream monitoring. pages 156–167, 2017.

# A Appendix

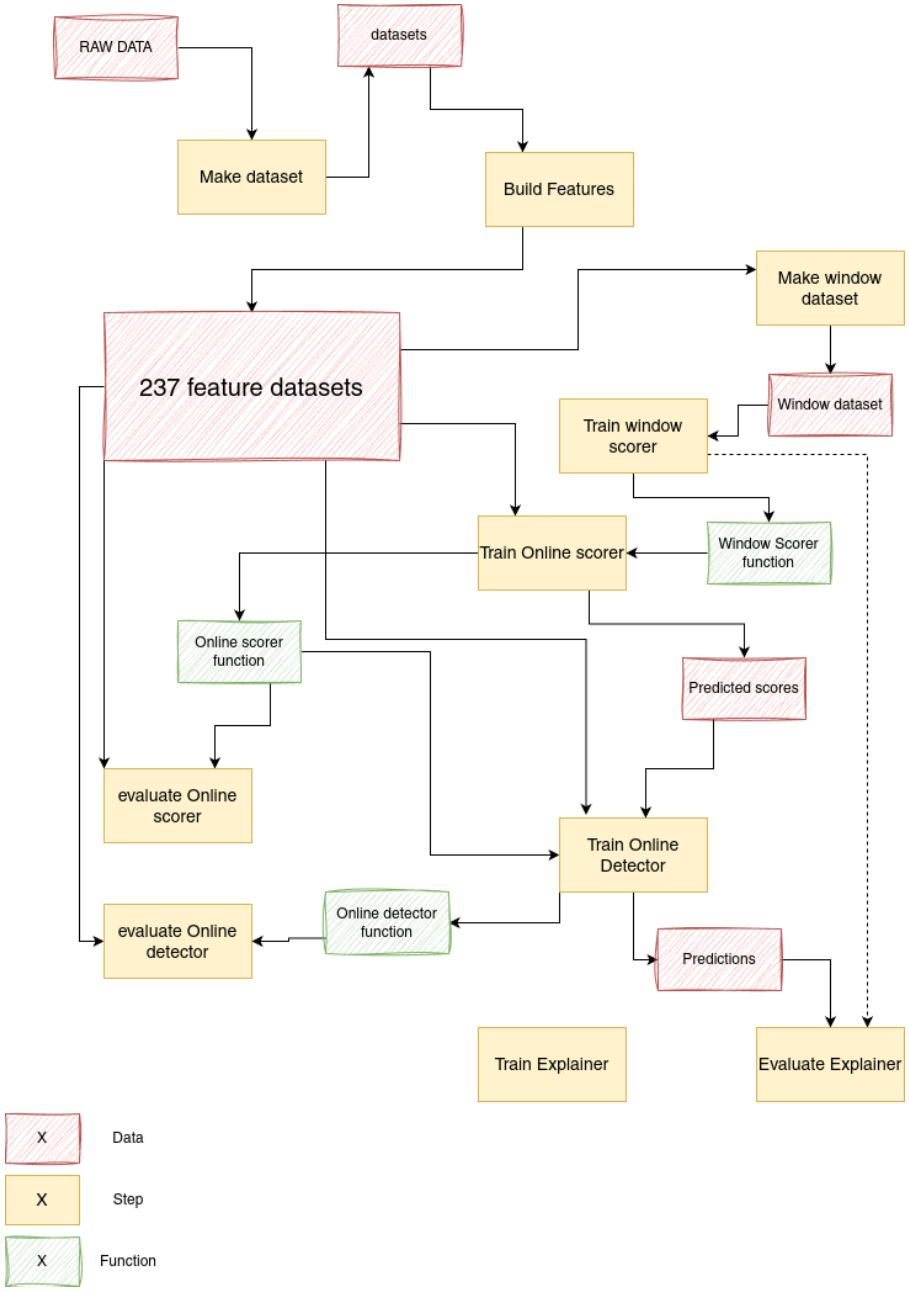



Figure 7: Pipeline of Exathlon