Contents

List of figures

List of tables

CHAPTER 1

Methodology and implementation

1.1 Datasets

In this section we will describe the datasets That we used in our experiment and provide their characteristics. We will also describe the evaluation metric, and the hyperparameter spaces we used.

1.1.1 Univariate dataset

There are various existing labeled dataset that could be extracted from the Yahoo benchmark, the Numenta benchmark, the UCR benchmark and many others. Unfortunately Yahoo wasn't opened and we made a demand, but it has been rejected. UCR too wasn't available when we made our tests. So we focused on Numenta which is almost sufficient, and particularly interesting since it's highly used in the literature. Numenta provide 52 labeled datasets in 7 groups:

- **Datasets with artificial anomalies :** in this set of datasets, the complete streams present seasonalities (for some of them), with noises, but none of those datasets present trends.
- **Datasets extracted by the Amazon CloudWatch service :** those datasets record CPU utilization, network bytes, and disk read bytes. When visualizing the datasets, we remark that they present : cycle, seasonality, noises, drift and no trend.
- **Datasets of online advertisement clicking rate :** these datasets record the cost-per-click and the cost-per-thousand impression on clicking rate. The datasets also present seasonalities and trends, cycles, and noises.
- **Datasets on real known causes :** here, the causes were known and labeled as the event occurred. Those are mainly on known systems failure such as the ambient temperature or machine temperature failure, but there are also datasets on fraud detection. Here datasets are mostly showing cycles, noises.

- **Datasets on real traffic data from Minnesota :** those datasets are mostly showing noises and cycles.

  Datasets on real twitter mentions on a publicly traded company such as google or IBM : the metric value represents the number of mentions for a given stock symbol every 5 minutes. Here datasets are showing noises and cycles.

The datasets extracted from **real known causes** are interesting since they don't depend on NAB appreciation. And they permit a diverse form of data stream. **Amazon cloud watch datasets** and **Real advertisement clicking rate** are also interesting since they present various concept drift, and various components which can enable relevant conclusion from our study. Those are summarized in table 1.1.

Table 1.1: Univariate datasets chosen for abnormal point detection

Dataset	Domain	Dataset length	number of anomalies	Drift	Seasonality	Trend	Cycle
Real AWS Cloud watch on cpu utilization : 24ae8d	cpu utilization of AWS	4032	2	no	no	constant	yes
Real AWS Cloud watch on cpu utilization : 53ea38	cpu utilization of AWS	4032	2	no	no	constant	yes
Real AWS Cloud watch on cpu utilization : 5f5533	cpu utilization of AWS	4032	2	yes	no	constant	no
Real AWS Cloud watch on cpu utilization : 77c1ca	cpu utilization of AWS	4032	1	no	no	constant	no
Real AWS Cloud watch on cpu utilization : 825cc2	cpu utilization of AWS	4032	1	no	no	constant	no
Real AWS Cloud watch on cpu utilization : ac20cd	cpu utilization of AWS	4032	1	yes	no	constant	no
Real AWS Cloud watch on cpu utilization : fe7f93	cpu utilization of AWS	4032	3	yes	no	constant	no
Real AWS Cloud watch on disk written bytes : 1ef3de	disk utilization of AWS	4730	1	yes	no	constant	no

Real AWS Cloud watch on disk written bytes : c0d644	disk utilization of AWS	4032	3	yes	no	constant	no
Real AWS Cloud watch on network traffic : 257a54	network traffic of AWS	4032	1	yes	no	constant	no
Real AWS Cloud watch on network traffic : 5abac7	network traffic of AWS	4730	2	yes	no	constant	no
Real AWS Cloud watch on number of elb request : 8c0756	number of elb request of AWS	4032	2	yes	no	constant	no
Real AWS Cloud watch on grok asg anomaly : 8c0756	grok asg anomaly of AWS	4621	3	yes	no	constant	no
Real AWS Cloud watch on iio_us-east-1_i-a2eb1cd9_ NetworkIn	network traffic of AWS	1243	2	yes	yes	constant	no
Real AWS Cloud watch on rds_cpu _utilization_ cc0c53	cpu utilization of AWS	4032	2	yes	no	constant	no
Real AWS Cloud watch on rds_cpu_ utilization_e47b3b	cpu utilization of AWS	4032	2	yes	no	constant	no
Real advertisement exchange : exchange-2_cpc_results	Real advertisement exchange	1621	1	yes	yes	yes	no
Real advertisement exchange : exchange-2_cpm_results	Real advertisement exchange	1621	2	yes	yes	yes	no
Real advertisement exchange : exchange-3_cpc_results	Real advertisement exchange	1538	3	yes	yes	yes	no

Real advertisement exchange : exchange-3_cpm_results	Real advertisement exchange	1538	1	yes	yes	yes	no
Real advertisement exchange : exchange-4_cpc_results	Real advertisement exchange	1643	3	yes	yes	yes	no
Real advertisement exchange : exchange-4_cpm_results	Real advertisement exchange	1643	4	yes	yes	yes	no
Real known cause :ambient temperature failure	Real known cause	7267	2	yes	yes	yes	no
Real known cause :cpu_utilization_asg_misconfiguration	Real known cause	18050	1	yes	yes	yes	yes
Real known cause :ec2_request_latency_ system_failure	Real known cause	4032	3	no	no	yes	no
Real known cause :machine temperature system failure	Real Real known cause	22695	4	no	no	no	no
Real known cause :new York taxi	Real known cause	10320	5	no	yes	yes	yes
Real known cause :rogue agent key hold	Real known cause	1882	2	yes	no	yes	no
Real known cause :rogue agent key up-down	Real known cause	5315	2	yes	no	constant	no

1.1.2 Multivariate dataset

In the literature we found various available datasets of real life problems, we mainly focused on some of them as following :

- In the domain of fraud detection, the dataset **credit card detection** contains transactions made by credit cards in September 2013 by European cardholders. This dataset

shows the transactions that took place in two days, where we have 492 frauds out of 284,807 transactions. The dataset is very imbalanced, the positive class (frauds) accounts for 0.172% of all transactions. The original features are not provided due to privacy policy; and given dimensions are results of PCA applied on original features [1].

- In the IOT domain, SKAB proposes a benchmark of datasets extracted from an industrial testbed. Those datasets were extracted from experiments made by closing the valve at the outlet of the flow from the pump, by closing the valve at the flow inlet to the pump. Others from data obtained from other experiments:
  - Sharply behavior of rotor imbalance
  - Linear behavior of rotor imbalance
  - Step behavior of rotor imbalance
  - Two-phase flow supply to the pump inlet
  - Draining water from the tank until cavitation
  - etc [2].

We summarized those datasets in the following table.

Table 1.2: Multivariate chosen datasets for abnormal point detection

Dataset	Domain	Dataset length	number of anomalies	Number of dimensions	Short description	Drift	Seasonality	Trend	Cycle
other 13	Sharply behavior of rotor imbalance	968	4	7	3 dimensions have seasonality, each of different length. Anomalies are hard to detect, but it seems that each anomaly is related to a specific dimension. Nevertheless, it's difficult to be precise on the anomaly position.	no	yes	no	no
other 17	Exponential behavior of rotor imbalance	1147	4	7	The dataset presents a dimension with seasonality and the others with cycles.	no	yes	no	yes
other 20	Draining water from the tank until cavitation	1191	4	7	one dimension presents a concept drift, two others with season and trends, and the rest seems to be random fluctuation.	yes	yes	yes	no
other 9	Closing the valve at the flow inlet to the pump	751	2	7	Two dimensions show non-constant trend with cycles.3 others show only cycles and the two remaining are constants with some random fluctuations.	no	no	yes	yes

other 14	Linear behavior of rotor imbalance	1153	2	7	One dimension shows a concept drift from a constant trend to a trend with seasonality. 2 of the dimensions point out roughly one of the anomalies. There is one dimension with seasonality, 2 constant ones and the rest with trends and cycles.	yes	yes	yes	yes
other 11	Data obtained from the experiments with closing the valve at the flow inlet to the pump	665	4	7	3 constant trend dimensions roughly pointing out one of the anomalies. Another anomaly is a peak on one dimension with seasonality, and the rest of the anomalies are not easy to identify.	no	yes	no	no
other 22	Water supply of increased temperature	1079	4	7	There is one dimension with concept drift, two others with trend and seasonalities. Another with strong long period cycles, and another with seasonality and cycles on a constant trend; and two with a constant value and peaks.	yes	yes	yes	yes

other 15	Step behavior of rotor imbalance	1147	2	7	One dimension with a concept drift, one with seasonality and trend; and the others are difficult to categorize.	yes	yes	yes	no

1.2 Evaluation metrics

In the literature various metrics have been used to assess anomaly detection methods in streaming data. In the frame of point anomaly detection, classic metrics like the AUC, precision, recall,f1 score has been used [3].

But other methods rather used a metric giving the credit to the method when it's at + or - 1% near the abnormal point [4]. We think it is more relevant to check it.

Other's estimated that it's difficult to provide accurately the exact position of the anomaly in various cases. When working with streaming data, detecting anomalies as soon as possible is worthy. In this mood, NAB proposes a score which aims to reward anomalies detected around the ground truth label but also early detection. For constructing the neighbourhood around the ground truth, NAB authors propose to share 10% of the time series length among the existing anomalies in a data file. The length attributed to each ground truth anomaly will represent the length of the window around it. For this NAB consider an application profile **A** which is a matrix having weights for **false positive** (A_{FP}) , **true positive** (A_{TP}), **true negative** (A_{TN}) and **false negative detection** (A_{FN}). The reason for this matrix is the fact that for some domains, **false positive** labels have to be more penalized than **false negative** and the inverse for other domains. So the application profile permits to weigh the way the algorithm will be rewarded and penalized. By considering a given window around the ground truth label, an anomaly detected inside this window and **y** the relative position of the detected anomaly. NAB scores this detection by equation 1.1. The final score for a data file is given by equation 1.2.The score of all data files is given by 1.3 and the normalized score is given by 1.4.

$$\sigma^A(y) = (A_{TP} - A_{FP})(1/(1 + e^{5y})) - 1 \tag{1.1}$$

$$S_d^A = (\sum_{y \in Y_d} \sigma^A(y)) + A_{FN} f_d \tag{1.2}$$

$$S_d^A = \sum_{d \in D} S_d^A \tag{1.3}$$

$$S_{NAB}^A = 100 * (S^A - S_{null}^A)/(S_{perfect}^A - S_{null}^A)$$

(1.4)

[5]

An example of NAB scoring procedure could be find at 1

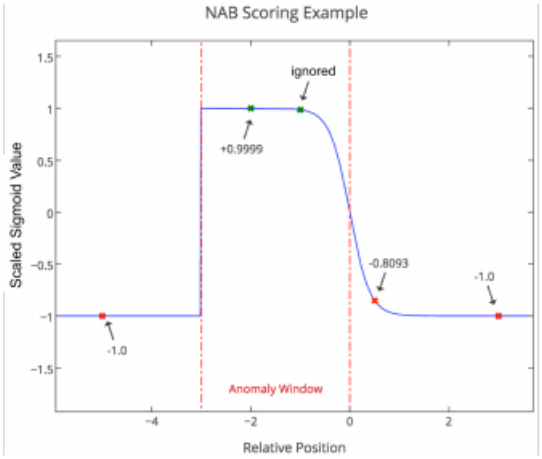

Figure 1: Example of NAB scoring procedure.
Example of NAB scoring procedure. In a given window, only earlier detection is valorized.

For **point anomaly detection**, the score of MERLIN and NAB are two interesting scores. What is more interesting is the zone around the anomaly given by NAB. NAB defined this score because in many more cases the anomaly is difficult to point out, we can even think the anomaly is a subsequence one. This impacted on the quality of labels given for the exact position of anomalies of much datasets used for abnormal point detection; those problems had been more highlighted in [6]. There are also many papers complaining on the exact position of anomalies given for NAB datasets. For these reasons we decided to **give the full point to a method if it finds an anomaly in the abnormal region provided for each dataset of NAB** as in [7] and in MERLIN. For each dataset we only have anomaly score on each point of the dataset, anomalies position with a gap (representing the abnormal region) that we defined as 1% of the time series length (MERLIN) . We processed the f1 score on all possible thresholds on the anomaly scores provided by the method. And finally kept the threshold providing the higher f1 score. A key element here is how we process the precision and the recall for the f1 score. For the recall we did as NAB authors, the true positive is given by the number of regions found (A region is found if the model identify an anomaly at at least one point in the abnormal region); and finally the recall is computed as the rate of abnormal region found by the method among all abnormal regions. We computed the precision as the rate of points pointing to a real abnormal region among all points detected. Finally the f1 score $= \frac{(2*\text{recall}*\text{precision})}{\text{recall}+\text{precision}}$.

In the next section we will briefly describe how we implemented all methods and give details of test procedures.

1.3 Implementation details

In this section we will provide the libraries used for the implementation of each method, and the search space provided for hyperparameters tuning procedure.

The language used for the tests is Python, and the library used for hyperparameters tuning is hyperopt in its basic configurations. Hyperopt will start by a random hyper parameter combination and will use $\frac{3}{4}$ as the quantile to obtain $y*$ as seen in the tree parzen estimation section. For each dataset we performed 30 iterations of parameter research.

For abnormal point detection, we compared MILOF, HStree, iforestASD, online ARIMA, and KitNet.

In the algorithm of **MILOF**, the parameters needed are: the maximal number of clusters **max-nbr-clusters**, the memory limit size **b**, and the number of nearest neighbor considered **K**. For each dataset we have taken :

- $5 \leq max - nbr - clusters \leq 15$,
- $5 \leq K \leq 15$,
- and we fixed $b = min(500, n)$ with n the size of the dataset.

MILOF was already implemented by the authors, so we used their implementations available at [8].

In the **iforestASD** algorithm, the parameters are the initial window size **init-W**, the window size **WS** inside which we provide score and monitor concept drift, the number of trees **ntrees**, the maximal number of features used for training each tree **features**, and the sampling size ϕ. Empirically the sampling training size is fixed to 256 as mentioned by tests made by iforest authors. For each dataset we have taken:

- $100 \leq init - W = WS \leq \frac{n}{4}$
- $15 \leq ntrees \leq 100$
- $1 \leq features \leq dim$ with **dim** maximal number of features of each instance of the dataset

Fortunately we found a library called Pysad which already has an implementation of iforestASD but without the concept drift implementation. Pysad proposed some function which permits us to train in batch and partially, and provide scores. The Pysad iforestASD class extends the iforest class of another library called Pyod. Pyod provides a library for

anomaly detection in batch data. Through the iforest class that extends iforestASD, we were able to use our hyperparameters. For the concept drift implementation, we prefered the non-parametric proposition of [9] using ADWIN, a concept drift detection algorithm.

In the **HS-tree** algorithm, the parameters are: the initial window size **init-W**, the window size **WS** inside which the model is train to test the next window size (so a window size balance from test window to training window once all its instances has been tested as we described on the section dedicated to half-space tree) , the number of trees **ntrees**, and the maximal **max-depth** of each tree. For each dataset we have taken hyperparameter in the following range:

- $100 \leq init - W = WS \leq \frac{n}{4}$
- $15 \leq ntrees \leq 100$
- $15 \leq max - depth \leq 25$

Similarly, as iforestASD Pysad provided us an implementation of Half space tree.

In the **Online ARIMA** algorithm, the parameters are: the autoregressive order **p**, and the differential order **d**.

For each dataset we have taken:

- $0 \leq d \leq 10$
- $0 \leq p \leq 200$

For the implementation of Online ARIMA we referred ourselves to its implementation in [10]. On their implementation we didn't find any possibility to add the differencing order, we contacted the author's on their GitHub, but we didn't get any response. Finally we differentiated the whole stream on our own, using recursively the **diff()** function provided by the pandas' library in order to have the **d** differencing order of the whole stream before providing it to the algorithm.

In the **Kitnet** algorithm, the parameters are **m** the maximal dimension that will have each subset of features, and **init-W** the number of data instances used to learn the feature mapping (it defines which features should be in each subset of dimension having at most **m** elements). **m** also defines the maximal dimension of each mini subset of similar features which would be studied together. For each dataset we have taken $1 \leq m \leq 30$. Fortunately, KitNet was already implemented on the GitHub of the author, and Pysad had already integrated it. So we directly used this implementation.

CHAPTER 2

Results

2.1 Univariate data stream

As said in the previous section, we assessed each method by giving the full point to the methods if it's around the anomaly. This surrounding zone is centered around the labelled anomaly. In table 2.4 we recorded the scores of MILOF, IforestASD, Online ARIMA and HS-tree with their respectives best parameters. KitNet is not compared because it's for multivariate datasets. Since some methods could have near `0` scores on some datasets, their best parameters shouldn't be considered in this case. In table 2.2 we gave the time cost in second. In those tables the best parameters are under the params column.

We will consider the following notation for our parameters:

- **For MILOF**: we will note **C** for the number of clusters to maintain, **K** for the number of nearest neighbors used to compute the local outlier factor. As said during the implementation, the memory size limit has been fixed to `500`.
- **For iforestASD**: we will note **WS** for the sliding window size and also the initial window size, **ntrees** for the number of trees of the model. Here we don't need to take into account the number of features since we have only one feature.
- **For Hs-tree** we will notice **max-depth** for the maximal depth of each tree, **ntrees** for the number of trees of the model, **WS** for the sliding window length which is also the size of **init-W**, the initial sliding window.
- **For online ARIMA** we will notice **p** for the auto-regressive order and **d** for the derivative order.

You can see on the following figures, the images of the datasets on which we are performing our tests.

Table 2.1: Scores for point anomaly detection on univariate datasets

Dataset	MILOF		iforestASD		HS-tree		Online ARIMA	
	Score	**Params**	**Score**	**Params**	**Score**	**Params**	**Score**	**Params**
Real known cause: ambient temperature failure	0.4	C=17,K=4	**0.67**	WS=1436, ntrees=23	0.3	WS=1223, ntrees=15, max-depth=11	**0.67**	p=31, d=0
Real known cause: cpu utilization	0.5	C=24,K=14	0.42	WS=2970, ntrees=44	0.45	WS=3885, ntrees=19, max-depth=11	**1**	p=29, d=0
Real known cause: ec2 system_failure	0.5	C=34,K=5	0.343	WS=366, ntrees=42	**0.94**	WS=200, ntrees=15, max-depth=10	0.8	p=32, d=1
Real known cause :machine temperature system failure	0.15	C=36, K=5	0.7825	WS=1957, ntrees=23	**0.88**	WS=3066, ntrees=23, max-depth=23	0.66	p=75, d=2
Real known cause: new York taxi	0.25	C=14,K=4	0.31	WS=756, ntrees=23	0.5	WS=1899, ntrees=17, max-depth=12	**0.6**	p=89, d=1
Real known cause: rogue agent key hold	0.136	C=25,K=23	**0.33**	WS=324, ntrees=22	0.079	WS=427, ntrees=26, max-depth=19	0.1	p=20, d=6
Real known cause: rogue agent key up-down	0.4	C=40,K=6	**0.67**	WS=221, ntrees=23	0.15	WS=777, WS=777, ntrees=29, max-depth=15	0.11	p=13, d=8

Table 2.2: Execution time for point anomaly detection on univariate datasets

Dataset	MILOF		iforestASD		HS-tree		Online ARIMA	
	Time	**Params**	**Time**	**Params**	**Time**	**Params**	**Time**	**Params**
Real known cause :ambient temperature failure	172	C=17,K=4	200.1	WS=1436, ntrees=23	212.35	WS=1223, ntrees=15, max-depth=11	**49.7**	p=31, d=0
Real known cause :cpu utilization	430	C=24,K=14	438	WS=2970, ntrees=44	738.17	WS=3885, ntrees=19, max-depth=11	**129**	p=29, d=0
Real known cause :ec2 system$_{f}ailure$	51	C=34,K=5	167	WS=366, ntrees=42	125	WS=200, ntrees=15, max-depth=10	**37.5**	p=32, d=1
Real known cause :machine temperature system failure	560	C=36, K=5	580	WS=1957, ntrees=23	9752	WS=3066, ntrees=23, max-depth=23	**108.9**	p=75, d=2
Real known cause :new York taxi	275.1	C=14,K=4	268.8	WS=756, ntrees=23	4776	WS=1899, ntrees=17, max-depth=12	**390.9**	p=89, d=1
Real known cause :rogue agent key hold	30.8	C=25,K=23	77.5	WS=324, ntrees=22	**15.51**	WS=427, ntrees=26, max-depth=19	16.38	p=20, d=6
Real known cause :rogue agent key up-down	26.3	C=40,K=6	202.7	WS=221, ntrees=23	**7.625**	WS=777, ntrees=29, max-depth=15	37	p=13, d=8

Something to note is the fact that real known cause datasets are datasets with a great amount of noise, nevertheless those methods are trying their best to find at least some anomalies.

- **MILOF** doesn't show the best score on any datasets, but its scores are not always too bad. We think the basic assumption that MILOF do on the anomaly is not always the right one. We think in the presence of seasonality it could be relevant to manage the temporal index as a contextual attribute with the help of the periodical length.
- **iforestASD** provides acceptable best scores on ambient temperature failure and rogue agent key up-down (we think it's not relevant to consider the score provided on rogue key hold). On ambient temperature failure, iforestASD accurately identifies one of the anomalies, that's why it's more rewarded than the others. IforestASD identifies this anomaly well because it's a spike even if there were a great amount of noises, it

was able to identify it. On rogue agent key up-down it's particularly difficult to identify the anomaly due to the presence of many small drifts, but iforestASD was able to understand this pattern and accurately identify some of the anomalies. On the rogue datasets, almost all methods are not performing well due to the high amount of noise present on those datasets. We can see that Online ARIMA, which performs well on all the other datasets, isn't performing well on those two last one. We think iforestASD luckily found anomaly on rogue agent key updown due to the random selection of splitting value and the presence of ADWIN to highly detect concept drift on this dataset. On ec2 system failure and rogue agent key hold, iforestASD was able to identify anomalies, but was inaccurate since it was also pointing out some other points as anomalies. For the rest of the datasets, the main problem of iforestASD was the precision, because it was difficult for iforestASD to identify the underlying pattern.

- **HS-tree** provides the best score on **machine temperature system failure**, and **ec2 system failure**. **machine temperature system failure** is a dataset having only spikes as anomalies. We can remark that HS-tree is not performing well on rogue datasets and it's providing a low execution time on those datasets. While on the other datasets it almost has an execution time greater than the one iforestASD. We think this low execution time is due to the fact that the tree of HS-tree has a high limit lower than the high limit of iforestASD which is able to isolate some anomalies. This lets us think HS-tree is stuck on its maximal depth, hence doesn't well isolate the anomalies. In the HS-tree paper authors stated that 15 as max-depth is almost enough, but we think that when there are noises, or data with diversified values, as on rogue datasets, users should carefully select the maximal depth. This maximal depth is a double-edged knife for HS-tree; it can permit HS-tree to improve its execution time, but can also block it on the isolation of anomalies.

- **Online ARIMA** provides the best results on cpu utilization, ambient temperature failure and New York taxi. On those datasets, Online ARIMA was able to find a model near the one shown by each of those datasets, and through this it was able to identify points that outfit the real model. We can remark that online ARIMA is performing well on almost all the datasets except the rogue datasets, where it's difficult to identify a pattern or even distinguish the presence of concept drift. On almost all the datasets Online ARIMA was showing the lower execution time, we think it's due to the online learning of online ARIMA, and the fact that there is no need to replace the model even in the presence of concept drift.

By those little observations on our results, we think that seasonality, noises and concept drift should be carefully tracked when working with machine learning and statistical based

anomaly detection methods. For HS-tree, the max-depth should be well managed, if not it can block the model in the isolation job.

As shown on table 2.6 Online ARIMA was providing the best scores on all specific components on the dataset. In the presence of a lot of drift and noises, as on rogue's datasets, we recommend to use iforestASD. As the results show, the assumption made by MILOF on the anomaly is not always justified.

Table 2.3: This board shows on how many datasets each method gave the best scores, and also precise on how much of those datasets there concept drift, seasonality, non constant trend and cycle.

Method	Number of best scores	Concept drift	Seasonality	Trend	Cycle
MILOF	0	0	0	0	0
HS-tree	2	0	0	1	0
iforestASD	3	2	1	2	0
Online ARIMA	3	2	3	3	2

2.2 Multivariate

In this section we will show the scores, and the best parameters we obtained for each method on our set of multivariate datasets. Treating a data stream, ask for taking care of the execution time. So we also added a board showing the execution time in seconds that made each method.

- **For MILOF**: we will note **C** for the number of clusters to maintain, **K** for the number of nearest neighbours used to compute the local outlier factor. As said during the implementation, the memory size limit has been fixed to **500**.
- **For iforestASD**: we will note **WS** for the sliding window size and also the initial window size, **ntrees** for the number of trees of the model. Here, each tree is trained on all features.
- **For Hs-tree** we will notice **max-depth** for the maximal depth of each tree, **ntrees** for the number of trees of the model, **WS** for the sliding window and initial window size.
- **For KitNet** we will notice **m** the maximal size of an encoder (the number of feature each encoder take as input) and **init-W** the training window size used to learn the feature mapping (which features should be in each subset of dimension having at most **m** elements) before starting to provide anomaly scores. Next the train of the model is pursued and is done at each time an instance is coming in a Stochastic gradient descent (SGD) mode.

You can see on the following figures, the images of the datasets on which we are performing our tests.

Table 2.4: Scores for point anomaly detection on multivariate datasets

Dataset	MILOF		iforestASD		KitNet		Hs-tree	
	Score	**Params**	**Score**	**Params**	**Score**	**Params**	**Score**	**Params**
other 9	**0.67**	C=13, K=9	0.25	WS=200, ntrees=19	0.248	init-W=178, m=3	0.285	WS=200, ntrees=27, max-depth=15
other 14	0.14	C=12, K=13	0.8	WS=258, ntrees=44	0.5	init-W=266, m=5	**1**	WS=203, ntree=31, max-depth=14
other 11	0.21	C=7, K=13	0.5	WS=200, ntrees=19	**0.6**	init-W=144, m=7	0.46	WS=200, ntrees=30, max-depth=12
other 13	0.167	C=12, K=11	0.4	WS=241, ntrees=21	**0.69**	init-W=223, m=3	0.6	WS=237, ntrees=19, max-depth=18
other 17	0.102	C=14, K=14	0.122	WS=245, ntrees=40	0.121	init-W=232, m=2	**0.125**	WS=201, ntrees=22, max-depth=16
other 20	0.15	C=13, K=4	0.29	WS=251, ntrees=49	**0**.278	init-W=278, m=3	**0.67**	WS=200, ntrees=15, max-depth=19
other 22	0.32	C=8, K=3	0.295	WS=228, ntrees=20	0.286	init-W=187, m=4	**0.37**	WS=211, ntrees=27, max-depth=15
other 15	0.167	C=9, K=7	0.5	WS=268, ntrees=46	0.292	init-W=199, m=1	**0.52**	WS=246, ntrees=20, max-depth=14

In table 2.5 we provide the execution time in second of the evaluation with the best parameters.

Table 2.5: Time taken for point anomaly detection on multivariate datasets

Dataset	MILOF		iforestASD		KitNet		Hs-tree	
	Time	**Params**	**Score**	**Params**	**Time**	**Params**	**Time**	**Params**

other 13	10.13	C=12, K=11	37.92	WS=225, ntrees=24	**0.53**	init-W=223, m=3	153.7	WS=237, ntrees=19, max-depth=18
other 17	11.92	C=14, K=14	31.77	WS=245, ntrees=40	**0.4**	init-W=232, m=2	47.87	WS=201, ntrees=22, max-depth=16
other 20	5.98	C=13, K=4	31.36	WS=207, ntrees=33	**0.23**	init-W=278, m=3	206.18	WS=200, ntrees=15, max-depth=19
other 9	9.12	C=13, K=9	27.25	WS=200, ntrees=37	**0.25**	init-W=178, m=3	26.62	WS=200, ntrees=27, max-depth=15
other 14	21.83	C=12, K=13	36.63	WS=210, ntrees=35	**0.48**	init-W=266, m=5	189.4	WS=203, ntree=31, max-depth=14
other 11	6.5	C=7, K=13	30.91	WS=200, ntrees=19	**0.17**	init-W=144, m=7	2.8	WS=200, ntrees=30, max-depth=12
other 22	4.8	C=8, K=3	30.6	WS=228, ntrees=20	**0.17**	init-W=187, m=4	19.8	WS=211, ntrees=27, max-depth=15
other 15	7.08	C=9, K=7	32.27	WS=202, ntrees=25	**0.39**	init-W=199, m=1	7.35	WS=246, ntrees=20, max-depth=14

Table 2.6: This board shows on how many datasets each method gave the best scores, and also precise on how much of those datasets there were: concept drift, seasonality, non constant trend and cycle.

Method	Number of best scores	Concept drift	Seasonality	Trend	Cycle
MILOF	1	0	0	1	1
HS-tree	5	5	4	4	3
iforestASD	0	0	0	0	0
KitNet	2	0	2	0	0

- **MILOF** provided the best result only for the dataset other 9 and for the rest it's performing very badly. It's surely because the anomalies detected were far from the oth-

ers in terms of euclidean distance, since even KitNet which is supposed to capture correlation perform badly on this dataset. We think MILOF is not performing well on the other datasets because it's not capturing the dependence between dimensions, and the fact that it uses the euclidean distance makes it lose information even on individual dimensions. As we can see on tabe 2.5, the execution time of MILOF is depending on the number of clusters and the number of neighbors used.

- **HS-tree** provided the best results for 5 datasets (other: 14, 15 17, 20, 22) with bad results on other 17 and 22 almost the same values as KitNet and **iforestASD** there. We can remark that HS-tree and iforestASD perform similarly; but HS-tree always provide better results. In the literature, the main difference between iforesteASD and HS-tree are the way they chose splitting node value, and the way the anomaly score is provided. HS-tree gives the anomaly score by taking into account the mass of each node through which the point passes to provide an anomaly score to this point; while iforestASD only looks at the height of the point on the tree. For the splitting value, HS-tree use the average of the min and max value of each attribute, while iforestASD choose it randomly. This let us think that the selection of node splitting values and anomaly score provide by HS-tree is more worthy. We can remark that the execution time of HS-tree is the highest in most case. We think this could be due to the maximal depth

- **KitNet** provided the best results on other 11. On this dataset the anomaly is difficult to identify. The dataset has dimensions with seasonalities and trends. We think KitNet is performing better here because it correctly identifies the linear correlation between dimensions with seasonalities and the one showing peaks. Another interesting remark is the fact that KitNet is low time-consuming, we think this is mostly due to the simplicity of the architecture of Kitnet, its online training approach (SGD, and train on an instance only one time; it can be viewed as a SGD of 1 epoch) and the presence of GPU that were used to train the model and process calculus on machine. We think the implementation of KitNet is well optimized, but the simplicity of its architecture is its higher strength.

Bibliography

[1] *Credit Card Fraud Detection | Kaggle* (cited on page 5).

[2] *SKAB/data at master · waico/SKAB* (cited on page 5).

[3] Redhwan Al-amri et al. "A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data". In: 11.12 (June 2021), p. 5320. DOI: 10.3390/APP11125320 (cited on page 8).

[4] Takaaki Nakamura et al. "MERLIN: Parameter-free discovery of arbitrary length anomalies in massive time series archives". In: 2020-November (Nov. 2020), pp. 1190–1195. DOI: 10.1109/ICDM50108.2020.00147 (cited on page 8).

[5] A Lavin and S Ahmad. "Evaluating Real-time Anomaly Detection Algorithms-the Numenta Anomaly Benchmark". In: (2015) (cited on page 9).

[6] Renjie Wu and Eamonn J. Keogh. "Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress". In: *arXiv:2009.13807 [cs, stat]* (Oct. 8, 2020). arXiv: 2009.13807 (cited on page 9).

[7] *hexagon-ml* (cited on page 9).

[8] *dingwentao/MILOF: Online Anomaly Detection for HPC Performance Data* (cited on page 10).

[9] Maurras Ulbricht Togbe et al. "Anomalies Detection Using Isolation in Concept-Drifting Data Streams". In: 10.1 (Jan. 2021), p. 13. DOI: 10.3390/COMPUTERS10010013 (cited on page 11).

[10] *waico/arimafd: Popular method ARIMA for outlier detection purposes* (cited on page 11).