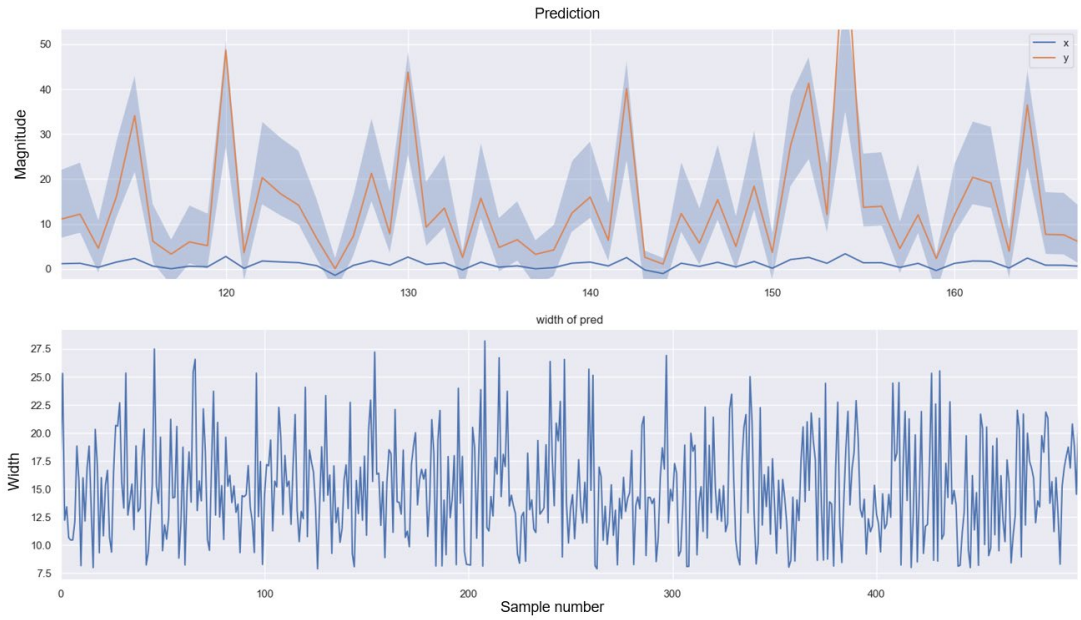
Drift detection treated as a neural network regression problem

Semester project



Author: Gardoni Maxime

Supervisor: Paul-Arthur Dreyfus
Teacher: Prof. Dimitrios Kyritsis

Winter 2020
ICT4SM Lab, EPFL

Contents

1 Introduction

1.1 Problem statement

This semester project investigates the use of neural networks to apply regression technics, which can then be used for drift detection, sometimes also called novelty detection or anomaly detection.

Novelty detection is the task of classifying test data that differ in some respect from the data that are available during training.
The novelty detection approach is typically used when the quantity of available “abnormal” data is insufficient to construct explicit models for non-normal classes. [1]

This problem is relevant in the Industry 4.0 field as a quality estimator during production.

1.2 Presented solution

The solution presented here make use of neural network to do a **probabilistic regression**, which means regression with an uncertainty quantification output. When this uncertainty grows beyond a given point, it can be used to trigger a drift detection.

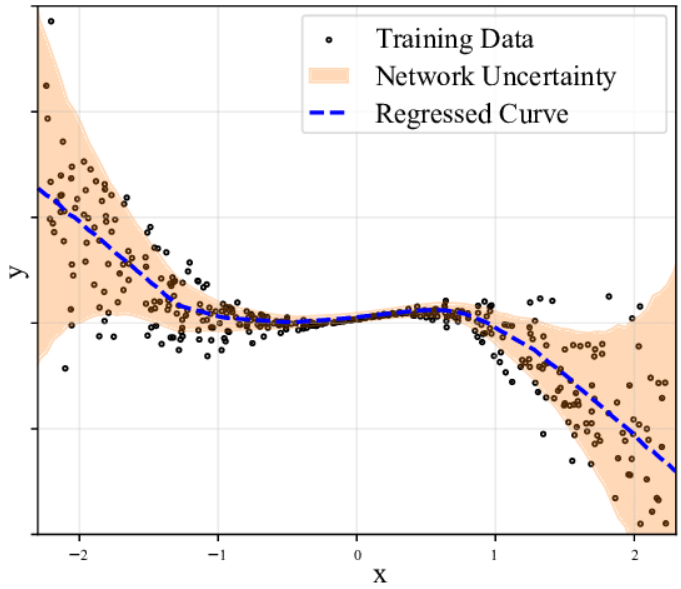


Figure 1: Visualization of a probabilistic regression [2]

The following method are explored:

- Boostrap Method
- Lower-Upper bound estimation (LUBE)
- LSTM Reconstruction error method

2 Problem definition in more details

The problem is formulated as following:

$$y = Ax^3 + Bx^2 + Cx + D$$

- A, B, C, D are constant.
- X is a variable sampled from a Gaussian distribution.
  The mean of this gaussian is constant under normal circumstances but will **move** when a drift happens. The goal of this exercise is to detect such drift.

Any drift is unknown during the training phase; the algorithm only has access to data classified as "normal". It is therefore called as a semi-supervised method in the anomaly detection field [3], [4]

2.1 Values that can be accessed

During the training phase, $\boldsymbol{X}$ value and the $\boldsymbol{Y}$ value are accessible for the algorithm (according to some observational noise, please refer to 3.1).

However, during the testing phase, the algorithm only has access to the $\boldsymbol{X}$ value.
This is an important fact; it means the algorithm will not be able to compare it's regressed absolute value (blue in Fig.1) to the $\boldsymbol{Y}$, and the algorithm has to rely on its uncertainty value (orange in Fig. 1).

3 Dataset

3.1 Observation Noise

The dataset is composed of 4 different observation noise datasets, each having 27 drift variants.

Dataset	Description
Noise_1	No observation noise on the X or the Y
Noise_2	Small observation noise on the X, no observation noise on the Y
Noise_3	Bigger observation noise on the X, no observation noise on the Y.
Noise_4	No observation noise on the X, observation noise on the Y

Figure 2: Noise description

Each one of those observation noise dataset have their own training dataset and must be considered as a separated experience.

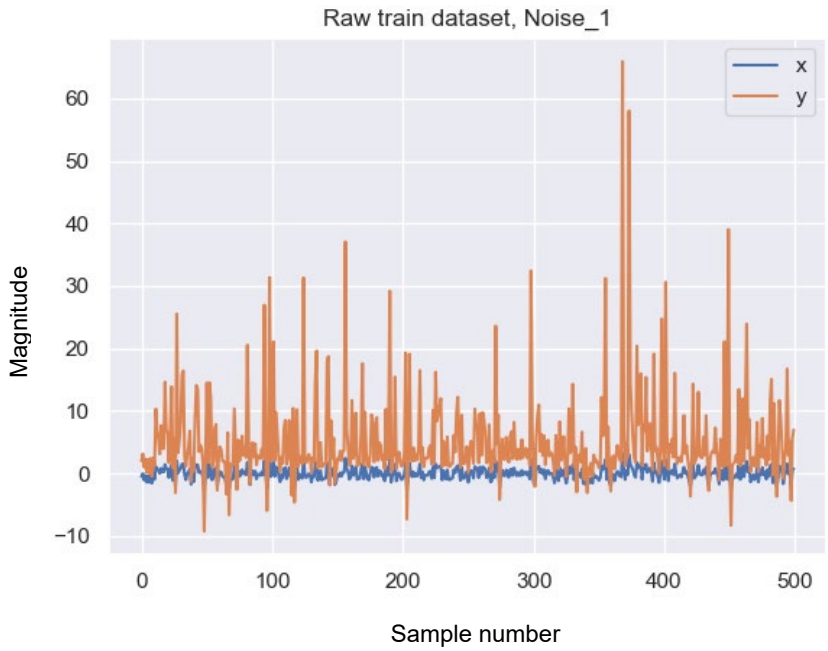


Figure 3: Plot of the Train_dataset of Noise_1, no drift

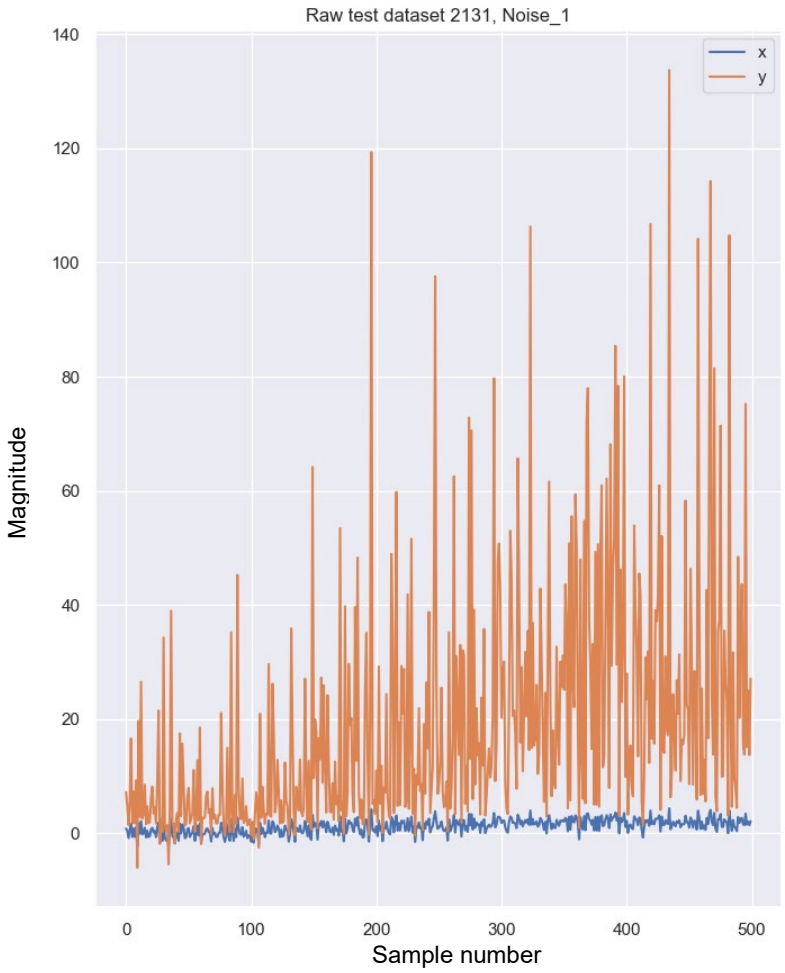


Figure 4: Plot of the dataset 2131, including drift (starting at sample 100)

3.2 Terminology

The Noise Variant names are characterised by 4 digits (example for example: “1234”).

Digits	Signification
1	Amplitude of the drift
2	Order of the drift polynome, i.e. his form
3	How much time the drift takes to come 0: Immediately 1: after 20 samples 2: after 200 samples 3: after 400 sample
4	Observation noise i.e. how well we can observe the x and y. 1: No observation noise on the X or the Y 2: Small observation noise on the X, no observation noise on the Y 3: Bigger observation noise on the X, no observation noise on the Y. 4: No observation noise on the X, observation noise on the Y

Table 1: definition of the metrics

Here is a table representing the different data:

NOISE_1	NOISE_2	NOISE_3	NOISE_4
1121	1122	1123	1124
1131	1132	1133	1134
1141	1142	1143	1144
1221	1222	1223	1224
1231	1232	1233	1234
1241	1242	1243	1244
1321	1322	1323	1324
1331	1332	1333	1334
1341	1342	1343	1344
2121	2122	2123	2124
2131	2132	2133	2134
2141	2142	2143	2144
2221	2222	2223	2224
2231	2232	2233	2234
2241	2242	2243	2244
2321	2322	2323	2324
2331	2332	2333	2334
2341	2342	2343	2344
3121	3122	3123	3124
3131	3132	3133	3134
3141	3142	3143	3144
3221	3222	3223	3224
3231	3232	3233	3234
3241	3242	3243	3244
3321	3322	3323	3324
3331	3332	3333	3334
3341	3342	3343	3344

Table 2: Drift variants representation

3.3 Pre-processing

In order to make the task of regression easier the Neural Networks architecture, the training data are pre-processed to have a mean of 0 and a standard deviation of 1.

$$X_{train} = \frac{X_{train} - \mu_{x_{Train}}}{\sigma_{x_{train}}}$$

$$Y_{train} = \frac{Y_{train} - \mu_{x_{Train}}}{\sigma_{y_{train}}}$$

The testing data are also pre-processed but using the mean and standard deviation of the trainset, because the test data are supposed unknown in advance, so we are not allowed to pre-compute their property.

$$X_{test} = \frac{X_t - \mu_{x_{train}}}{\sigma_{x_{train}}}$$

$$Y_{test} = \frac{Y_{test} - \mu_{x\_train}}{\sigma_{y\_train}}$$

4 Metrics used

For each of the noise variants, the drift appears the 100th sample. This value is supposed unknow and the algorithm must guess it as soon as possible.

Therefore, the following metrics are used:

Name	Definition
Lag of detection	The time taken by the algorithm to detect the drift after the 100th sample (to minimize). Can also be interpreted as the number of False Negative If the drift is not detected at all, we flag it.
False Positive	The amount of sample wrongly triggering a drift before the 100th sample (to minimize)
Drift not detected	Boolean value, True if the Drift was not detected at all

Table 3: Metrics definition

5 Solutions

The regression solutions presented here are inspired from the field of prediction intervals **(PI).**
The main inspiration of this work is the following paper, given in the beginning of the semester project:
"Neural Network-Based Uncertainty Quantification: A Survey of Methodologies and Applications" [5]

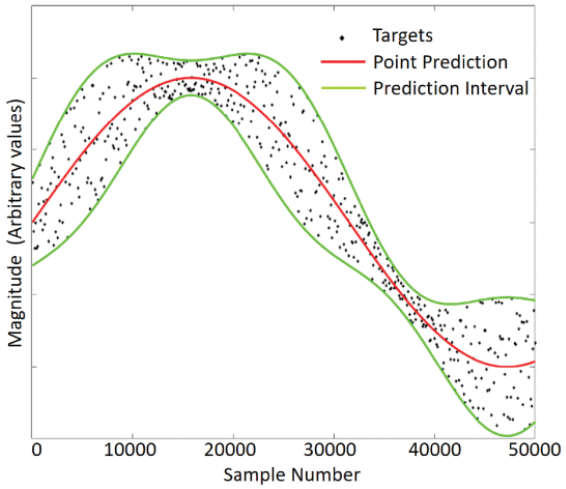


Figure 5: Graphic example of a predicition interval applied over the whole course of a signal [5]

Given a window of N sample (from sample $T - N$ to sample *T),* the algorithm tries to predict the sample $T + 1$ being present in some defined bounds.

These methods are reused to do a probabilistic regression, trying to guess $\boldsymbol{Y}$ when given $\boldsymbol{X}$, and the width of the prediction will be used as a measure of uncertainty. This measure of uncertainty can then be used to detect a drift.

It's important to note that since the problem doesn't allow access to the $\boldsymbol{Y}$ value during the test phase, the algorithm can only make use of its uncertainty measurements (i.e. the **green width** in Figure 5), and not directly it's absolute prediction (the red line in Figure 5).

5.1 Bootstrap method

According to [5], Bootstrap is one of the most popular method for building PI.
We implement here a variant of the algorithm cited in [6].

The global idea is to do a classic point regression with B neural network, then to do the uncertainty quantification via the standard deviation of all their predictions.

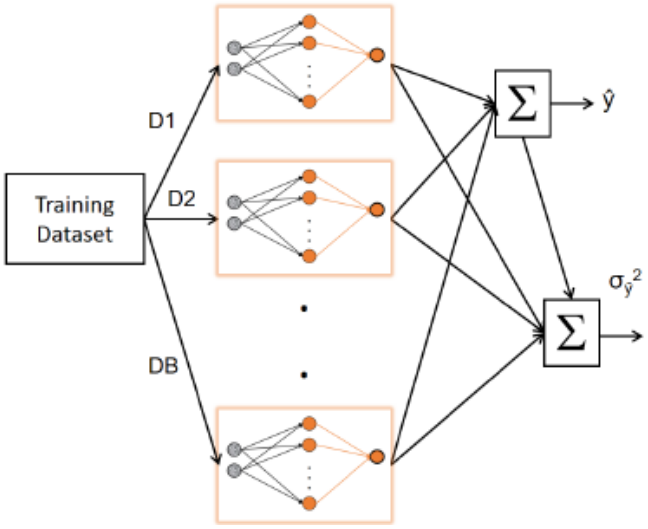


Figure 6: The composition of the B NN models applied in the bootstrap [5]

5.1.1 Building the prediction interval

Here are the steps used for the prediction interval:

1) A parent algorithm, composed of **B** children Neural Nets is defined, as seen in previous figure.
    - All the children are assigned different initial weight, according to a uniform distribution.
2) A **part** of the train dataset is assigned to each child, (resampling step).
3) Each child is trained to do a **point** regression. That is, given a sample **X**, they will try to infer the **Y** value.
   Since the children have different initial values and different dataset subpart, they will converge to different values.
4) The global mean regression can be computer by averaging all the children predictions.

$$\hat{y}_i = \frac{1}{B}\sum_{b=1}^{B}\hat{y}_i^b$$

5) The uncertainty bound can be computer by taking the standard deviation of the children prediction.

$$\sigma_{\hat{y}_i}^2 = \frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{y}_i^b - \hat{y}_i\right)^2$$

5.1.2 Building the threshold detection

1) The parent algorithm is run over all the **train** dataset, a prediction uncertainty vector is built:

$$\overrightarrow{\sigma_{\hat{y}\_train}{}^2} = \begin{bmatrix} \sigma_{\hat{y}_1}{}^2 \\ \vdots \\ \sigma_{\hat{y}_N}{}^2 \end{bmatrix}$$

2) $\overrightarrow{\sigma_{\hat{y}\_train}{}^2}$ is smoothed using a **moving average on the last W sample.**
   a. not here that the filter is FIR, which means it can be applied online.
3) A threshold is then defined as:

$$Threshold = SafetyFactor * max(\overrightarrow{\sigma_{\hat{y}\_train}{}^2})$$

5.1.3 Drift detection

1) The parent algorithm is run over all the **test** dataset, a prediction uncertainty vector is built:

$$\overrightarrow{\sigma_{\hat{y}\_test}{}^2} = \begin{bmatrix} \sigma_{\hat{y}_1}{}^2 \\ \vdots \\ \sigma_{\hat{y}_N}{}^2 \end{bmatrix}$$

2) $\overrightarrow{\sigma_{\hat{y}\_test}{}^2}$ is smoothed using the same moving average.
3) If $\overrightarrow{\sigma_{\hat{y}\_test}{}^2} > Threshold$, the sample is flagged as belonging to a drift.

5.2 Validation set

To find the best possible NN architecture for the children, a validation set need to be defined.
The approach chosen here for the validation set is to use the first 100 sample of one of the test set, since the 100 first sample are clear from any drift.

Another more rigorous approach would be to take 20% out of the train set and then run a K fold validation, which was not implemented here due to time constraints.

5.3 Training

The loss function used for the training is the built-in Pytorch MSE loss.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

And the network is trained with the Pytorch built-in stochastic gradient descent (SGD) optimizer.

5.4 Experiments

5.4.1 Defining the best architecture

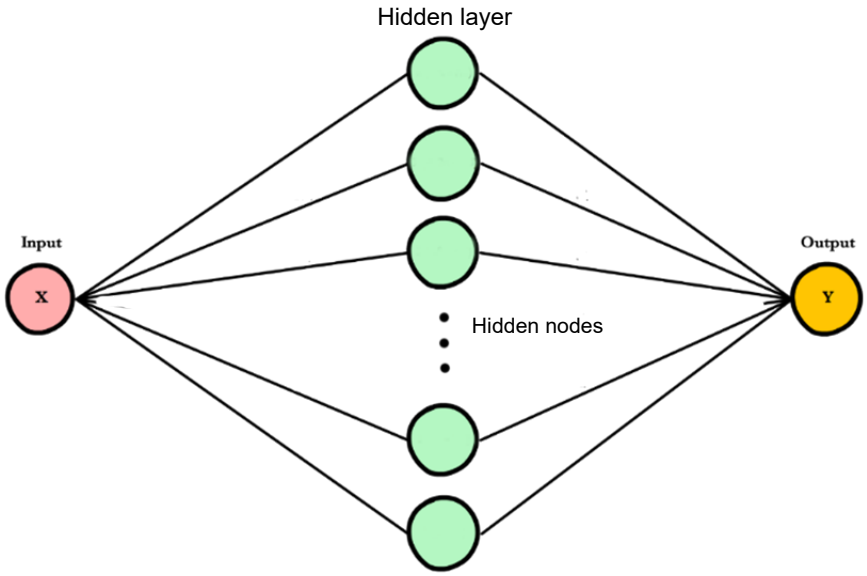


Figure 7: representation of a NN

As in [6], single hidden layer networks are considered, additionally, for this experiment, we consider a single input NN, because the function **Y**=f(**X**) depends directly on the last **X** sample

The hidden layer size is then changed between [5 and 35] and the one giving the **minimal validation loss** is then selected, which give 20 hidden nodes as the optimal one.

We then generate **B=100** children as in [6] and assign to each of them a random consecutive sequence of 50% from the dataset.

We then train then NN using a mean square error cost function (also called MSE loss) and Stochastic gradient descent (SGD) optimizer.

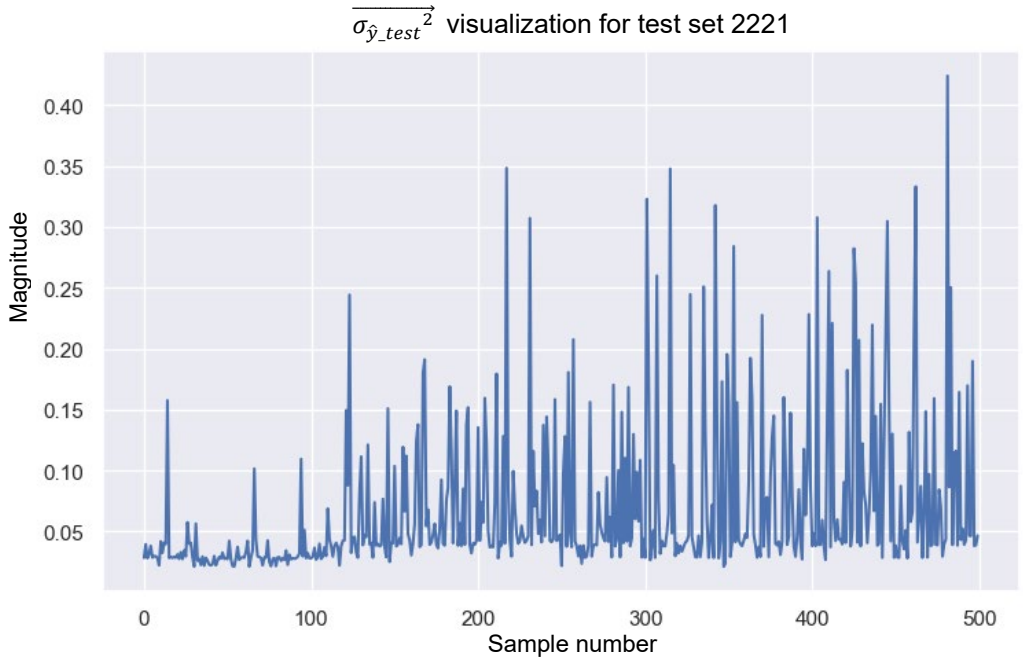


Figure 8: Plot of $\overrightarrow{\sigma_{\hat{y}\_test}^2}$. After the net is trained, we apply it onto test data set 2221, and plot here the $\overrightarrow{\sigma_{\hat{y}\_test}^2}$. We can see a clear visually an increase around sample 120, but the data is quite noisy, we therefore smooth it.

Afterward, the data is smoothed with $Window = 5$, and the threshold computed using a $SafetyFactor = 1.2$

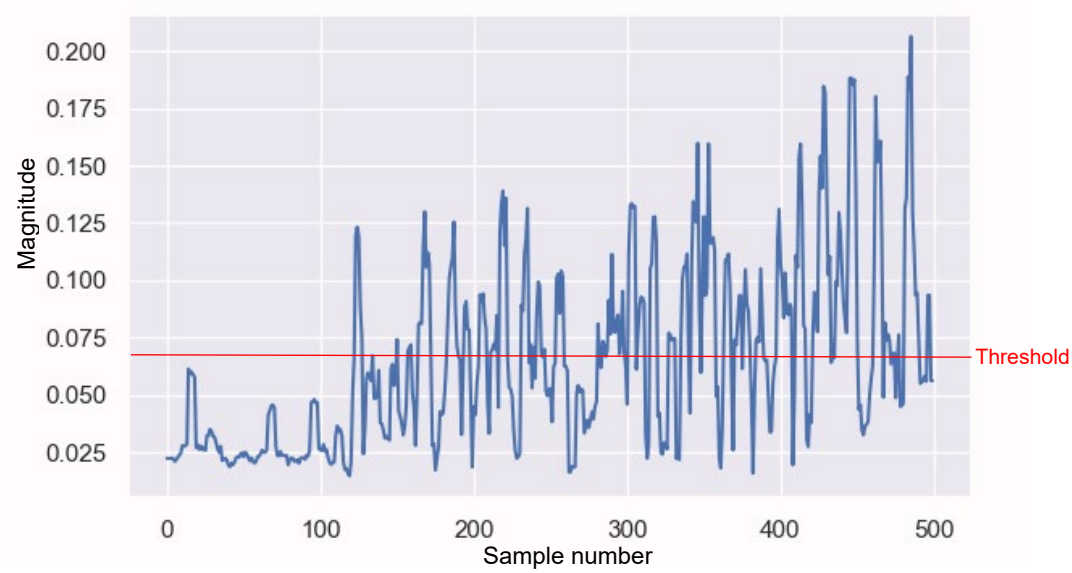


Figure 9: Representation of the smoothed $\overrightarrow{\sigma_{\hat{y}\_test}^2}$ with the Threshold

The parent algorithm is then applied onto the test dataset, and the metrics are computed.

For further analysis, a clean excel file was also made available.

	Noise_1	Noise_2	Noise_3	Noise_4
Tot. of false positive	0	2	17	0
Mean lag of detection	63.7	76.2	42.8	72.9
Median lag of detection	44	42	22	49
Tot. of non detected drift	0	0	0	4

Figure 10: Metrics analysis

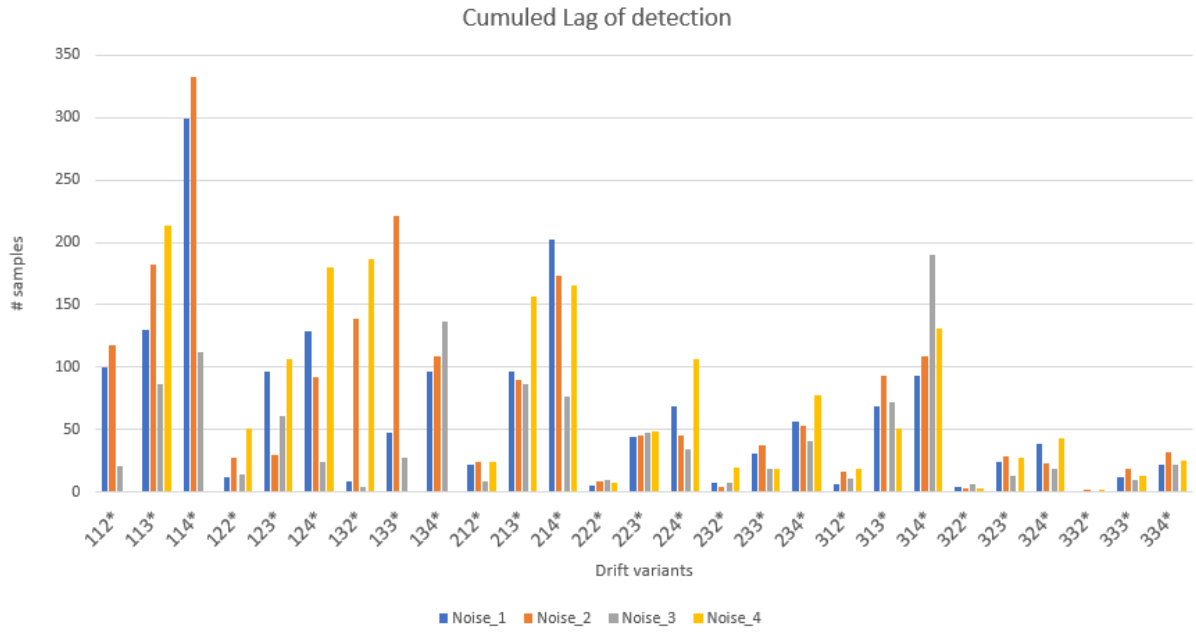


Figure 11: Representation of the cumuled lag of detection.

One can observe in fig 13 that the drift amplitude has a lot of influence on the detection speed (the bigger the amplitude the easier it is to detect), and that the drift of type 1 is the most difficult to detect.

Noise_1	false Pos.	Lag Of Detection
1121	0	100
1131	0	130
1141	0	299
1221	0	12
1231	0	97
1241	0	129
1321	0	9
1331	0	48
1341	0	96
2121	0	22
2131	0	96
2141	0	202
2221	0	5
2231	0	44
2241	0	69
2321	0	7
2331	0	31
2341	0	56
3121	0	6
3131	0	69
3141	0	93
3221	0	4
3231	0	24
3241	0	39
3321	0	1
3331	0	12
3341	0	22

Noise_2	false Pos.	Lag Of Detection
1122	0	118
1132	0	182
1142	0	333
1222	0	28
1232	0	30
1242	0	92
1322	1	139
1332	1	221
1342	0	109
2122	0	24
2132	0	90
2142	0	173
2222	0	8
2232	0	45
2242	0	45
2322	0	4
2332	0	38
2342	0	53
3122	0	16
3132	0	93
3142	0	109
3222	0	3
3232	0	29
3242	0	23
3322	0	2
3332	0	18
3342	0	32

Figure 12: Metrics for Noise_1 and Noise_2 (For further analysis purpose, a clean excel file was also made available)

Noise_2	false Pos.	Lag Of Detection
1123	4	21
1133	0	86
1143	0	112
1223	0	14
1233	0	61
1243	5	24
1323	0	4
1333	0	27
1343	5	137
2123	0	8
2133	0	86
2143	0	77
2223	0	10
2233	0	47
2243	0	34
2323	0	7
2333	0	19
2343	0	41
3123	3	11
3133	0	72
3143	0	190
3223	0	6
3233	0	13
3243	0	18
3323	0	0
3333	0	10
3343	0	22

Noise_3	false Pos.	Lag Of Detection
1124	0	Not detected
1134	0	213
1144	0	Not detected
1224	0	51
1234	0	106
1244	0	180
1324	0	187
1334	0	Not detected
1344	0	Not detected
2124	0	24
2134	0	157
2144	0	166
2224	0	7
2234	0	49
2244	0	107
2324	0	20
2334	0	19
2344	0	78
3124	0	18
3134	0	51
3144	0	131
3224	0	3
3234	0	28
3244	0	43
3324	0	2
3334	0	13
3344	0	25

Figure 13: Metrics for Noise_3 and Noise_4 (For further analysis purpose, a clean excel file was also made public)

5.5 LUBE (Lower Upper bound estimation)

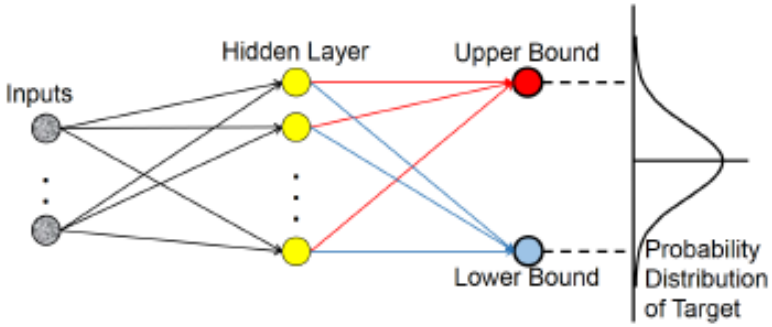


Figure 14: Neural net structure for the LUBE Method [5]

This method was originally developed in [7], and summarized in [5].
This method aims to give a lower and upper bound estimation of the data via a **direct neural network estimation.** The big strength of this method in the PI field is its ability to model asymmetric distribution.

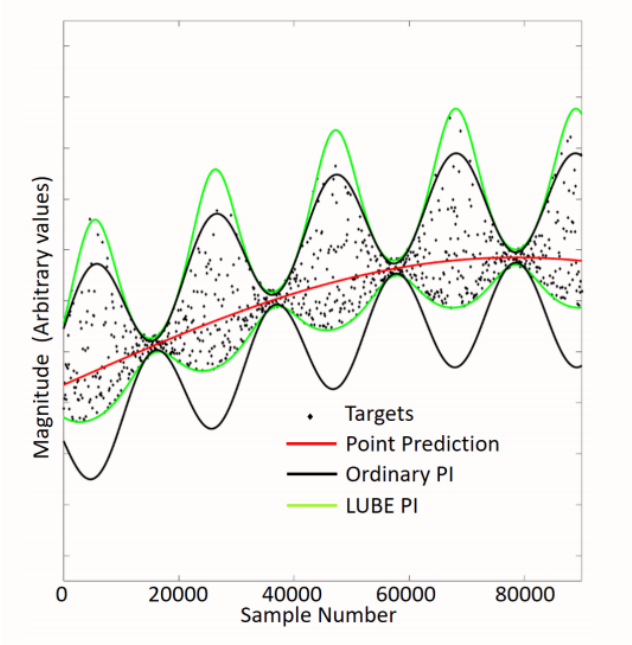


Figure 15: A rough sketch presenting the advantage of LUBE method. The green curve (LUBE) does a better job at predicting the non-gaussian and non-symmetric data.

A moving window is applied onto the **X** value **with N inputs,** and then the models will try to guess the **Y** value being within the bounds.

The 2 main goal for the bounds is to:

- contains as many datapoint within the bounds and little to no point outside (coverage).
- The prediction width should be as small as possible (in order to provide as much useful information as possible).

5.5.1 Coverage

Formally defined as the **PICP** (The PI coverage probability),
Provides the statistical probability of target values limited by the upper and the lower limits of the prediction interval.

$$PICP = \frac{1}{n}\sum_{j=1}^{n} c_j$$

Where:

$$c_j = \begin{cases} 1, & t_j \in [\underline{y_j}, \overline{y_j}] \\ 0, & t_j \notin [\underline{y_j}, \overline{y_j}]. \end{cases}$$

Here, t_j, $\overline{y_j}$ and $\underline{y_j}$ are the actual value, lower bound, and upper bound prediction of j_{th} sample respectively, n is the total number of samples.

5.5.2 Width

The formal definition of the width used in this work is the **Interval normalized root-mean-square width (PINRW),** and can be defined as:

$$PINRW = \frac{1}{R}\sqrt{\frac{1}{n}\sum_{j=1}^{n}(\overline{y_j} - \underline{y_j})^2}$$

Where $\boldsymbol{R}$ is the range covered by the $\boldsymbol{X}$ values of training dataset.

5.5.3 Optimising width and coverage at the same time

An infinitely big prediction interval will off course have a 100% coverage but will not provide any useful information to the user.

A metrics cleverly mixing the 2 therefore needs to be designed.

This metrics is formally defined as the Coverage width-based criterion (**CWC**).

$$CWC = PINAW\{1 + \gamma(PICP)e^{\eta(\mu - PICP)}\}$$

Where:

$$\gamma(PICP) = \begin{cases} 1, & PICP < \mu \\ 0, & PICP \geq \mu \end{cases}$$

Here, $\eta = 50$ and $\mu = 0.9$ are two hyperparameters.

Intuitive explanation of the metrics behaviour.

The default behaviour of the metrics is to penalize the width of the prediction via $\boldsymbol{PINAW}$.

However, when the coverage $\boldsymbol{PICP}$ drops below the threshold $\boldsymbol{\mu}$, the metrics also penalize that drops using a weighting factor $\boldsymbol{\eta}$.

5.6 Particle Swarm optimisation

We want to train our NN in order to minimize the CWC (i.e. use the CWC as a cost function).

$$\min_{NN} \; CWC = PINAW\{1 + \gamma(PICP)e^{\eta(\mu - PICP)}\}$$

Usually, Neural networks are trained using gradient descent or variants of the gradient descent (the most popular ones being stochastic gradient descent or Adam).

This approach is not possible in our case, since the given function is nonlinear, complex, discontinuous and nondifferentiable.

That's why [7] recommend using a particle swarm optimisation method (PSO) to train the NN.

5.7 Implementation details

Due to its flexibility and rising popularity among the academic field, the framework chosen for NN building is PyTorch.

Since PyTorch doesn't natively offer a PSO, it was manually interfaced with the popular library PySwarm [8], which a general-purpose optimisation library. This implementation with an external optimizer proved itself to be quite low level and time consuming.

Here is a description of the final solution: PySwarm takes care of simulating all the different particles (representing the NN weights and bias) at each iteration. Those weights are then manually injected inside the PyTorch NN in order to calculate the cost of the iteration, which is then transferred back to the PySwarm for the next iteration's particle computation.

After training of the Neural Net onto the train dataset, for each sample **X** of the train dataset, the prediction interval around **Y** is computed.

From this value we can compute the width of the prediction, which can be interpreted as the uncertainty:

$$width_j = \overline{y_j} - \underline{y_j}$$

This gives us a vector $width_{train}$ which is smoothed with a moving average of 5 samples.

A threshold is then defined as:

$$Threshold = SafetyFactor * max(width_{train})$$

The neural net is then applied onto the test dataset, the $width$ is computed for train sample, and if that width exceed the Threshold, the sample is being flagged as drift.

5.8 Experiments

Multiple architecture, taking either 1 sample **X** as input, or a window of **N** sample **X** as input, where tried (ranging from 1 to 15) were tried.

Single hidden layer as in [7] were considered, but also double hidden layers as exploration process. For each of those layers, hidden nodes number ranging between 3 and 25 were tested.

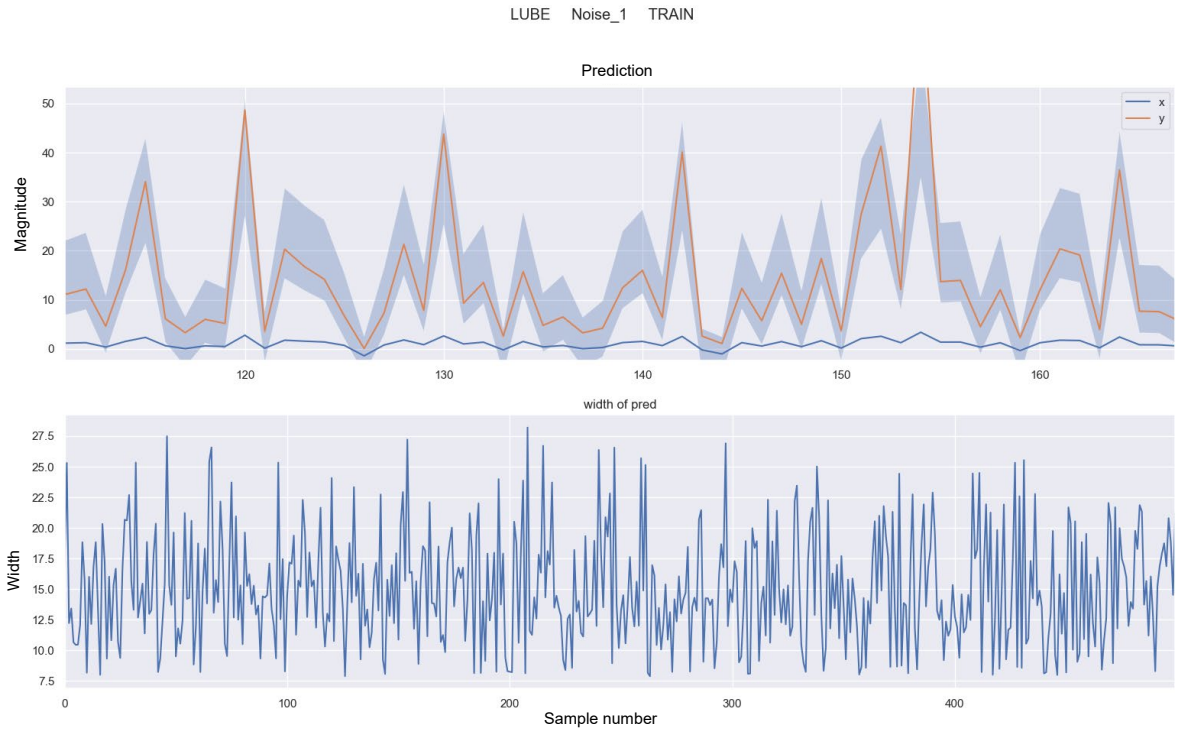


Figure 16: Visualization of the prediction interval on the train set, and the width evolution

However, they generalised quite badly when processing the anomaly sample.
The width would be expected to rise when going over drift, but instead it behaved in unexpected ways.

Depending on the architecture and the observation noise, the width would sometimes increase, sometime decrease, sometime stay the same. These results were also proved to be quite stochastic and not easily reproducible.

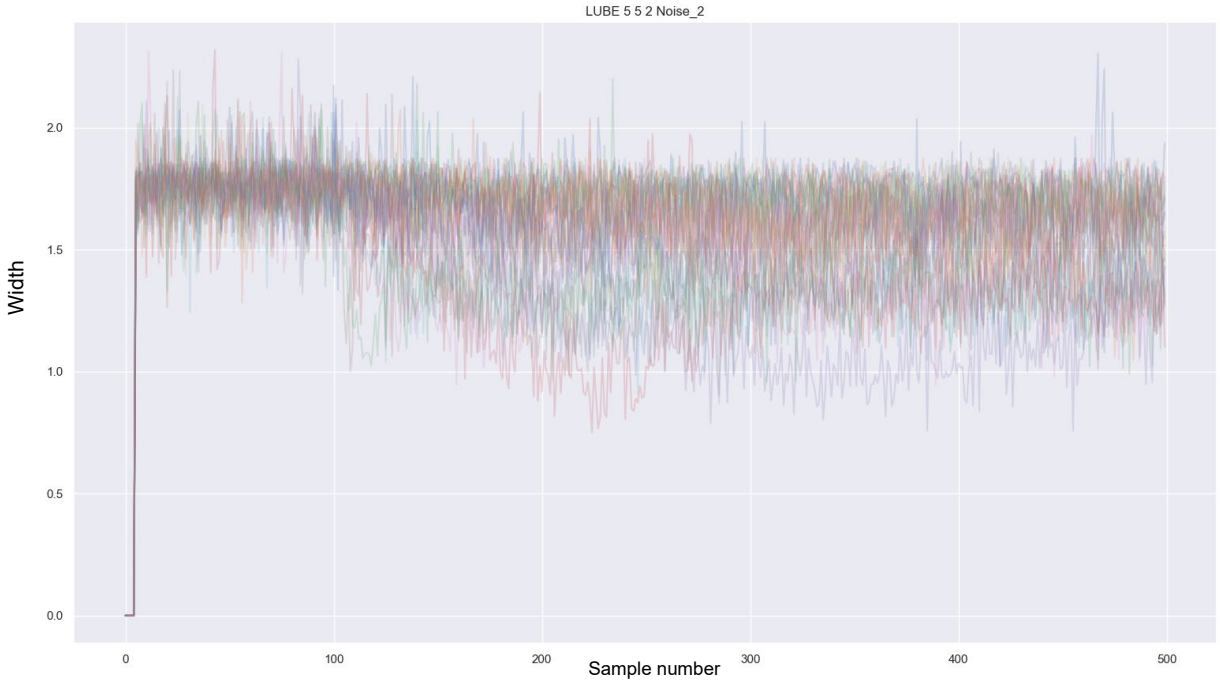


*Figure 17: Representation of the test regression width for all the drift variants. We can notice the width getting **smaller** when processing the drift. The NN architecture is the following: 5 inputs, 5 hidden node, 2 outputs, and applied on the Noise_2*

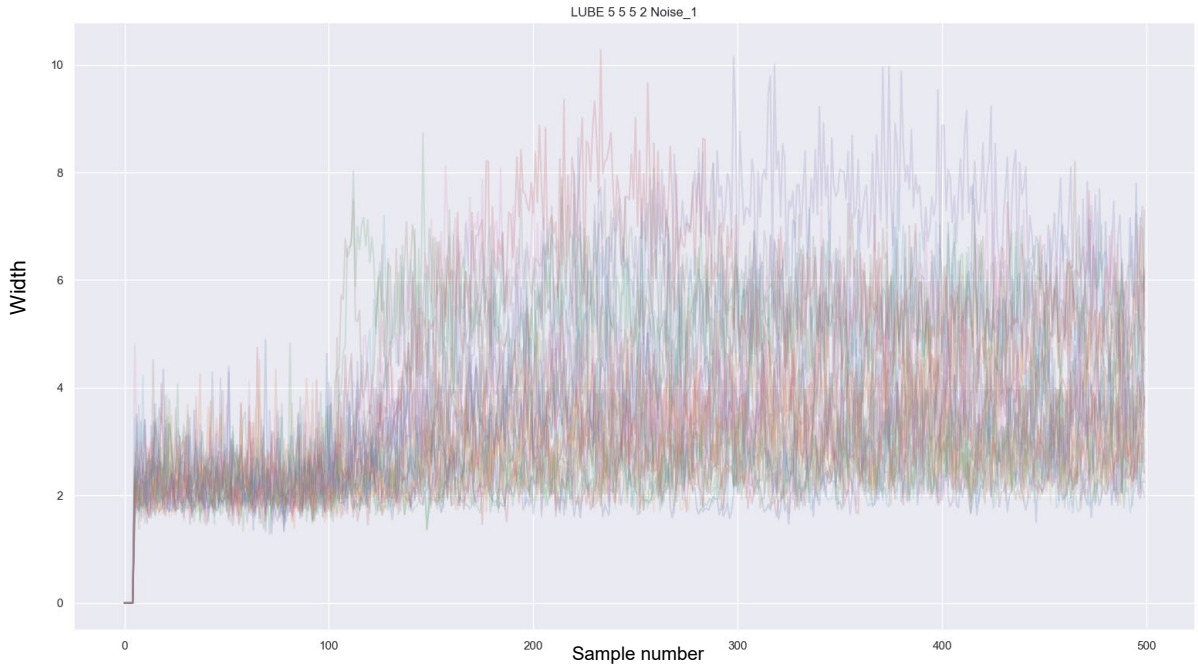


*Figure 18: Representation of the test regression width for all the drift variants, we can notice the width getting **bigger** when processing the drift. The NN architecture is the following: 5 inputs, 2 layers of 5 hidden nodes each, 2 outputs, and applied on the Noise_1*

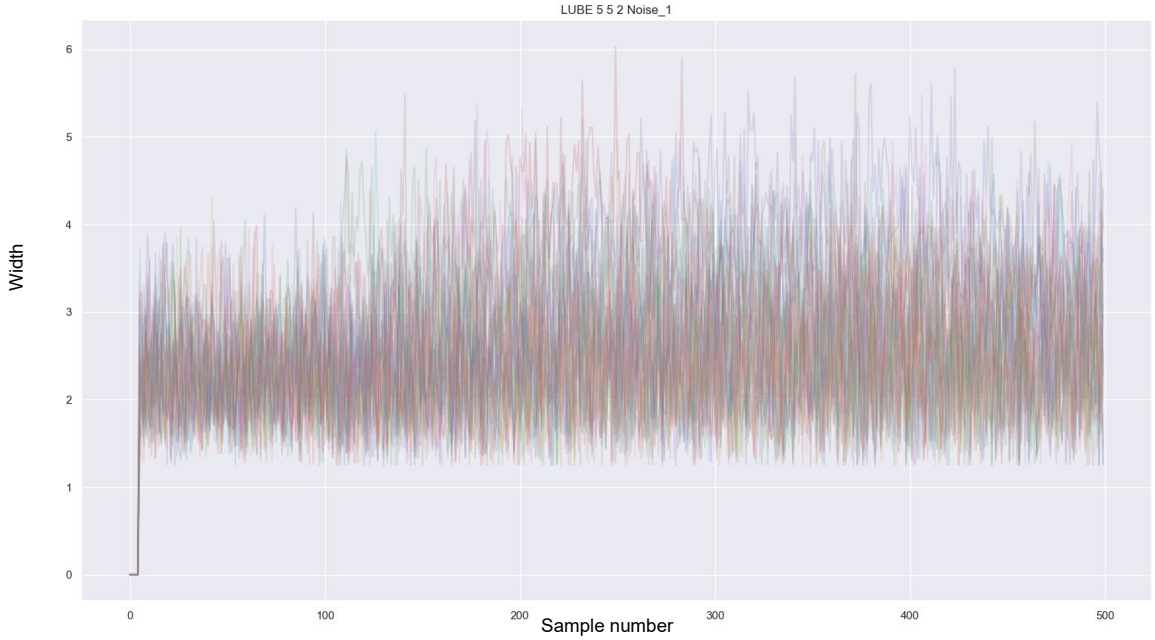


*Figure 19: Representation of the test regression width for all the drift variants, we can notice the width **staying constant** when processing the drift. The NN architecture is the following: 5 inputs, 2 layers of 5 hidden nodes each, 2 outputs, and applied on the Noise_1.*

5.9 Discussion

The method LUBE is considered "state of the art" and works well in the PI field but proved itself behaving quite erratic when applied to the problem of drift detection.

This is probably because the problems are quite different.
In the PI field, the test set is quite like the test set, there is no radical drift or anomaly happening. Typical application of PI stated in [9] are electricity load forecasting for a whole town, or demand forecast for the food industry, which do not have many anomalies.

In the drift detection problem, the neural net is shown radically different values during drift. Since the neural net only see very "normal" sample during the train phase, it is only good at making normal prediction interval, and cannot generalise very well to radically abnormal samples, and it therefore behave stochastically.

5.10 Parallel with the bootstrap method

One can note that it also is this "outlier stochastics behaviour" that is taken advantage of when applying the bootstrap method.
When processing drift data, all the B children network have a random different behaviour, thus the global standard deviation increase, and that is what is used to detect the drift.

5.11 Additional implemented methods (exploration)

5.11.1 Bootstrap with multiple inputs

As an additional work, the bootstrap method with multiple inputs was also implemented.
The idea is that the neural network could average the last few sample to do a better noise estimation.

In order to compare its performance in comparison to the unique input one, the following grid search was implemented:

Number of Inputs $\in [1, 5, 10, 15]$
Number Node in the Hidden Layer $\in [3, 5, 10, 15, 20]$

In order to compare the performance, a single child was trained, and the validation loss and train loss were observed.
For easier display in this report, the loss was averaged over the number of hidden nodes.

Validation loss	**1 input**	**5 inputs**	**10 inputs**	**15 inputs**
Obs. noise 1	0.09	0.15	0.07	0.22
Obs. noise 2	0.25	0.27	0.40	0.68
Obs. noise 3	0.59	0.74	1.00	1.25
Obs. noise 4	0.04	0.06	0.11	0.12

Table 4: Validation loss

Training loss	**1 input**	**5 inputs**	**10 inputs**	**15 inputs**
Obs. noise 1	0.026	0.054	0.010	0.014
Obs. noise 2	0.086	0.053	0.103	0.023
Obs. noise 3	0.189	0.154	0.106	0.053
Obs. noise 4	0.009	0.011	0.048	0.008

Table 5: Training loss

Interestingly, we can observe in Table 4 that the validation loss gets bigger with the number of inputs, especially with the **observation noise 3**.
At the same time, we can observe in table 5 that the Train loss get smaller with the number of inputs, especially with **observation noise 3.**

A possible hypothesis is that the neural network is “learning” locally relevant noise pattern in order to make better prediction, instead of “globally relevant” pattern such as the average. This then translate poorly to the validation set.

This can be interpreted as overfitting and reducing the number of parameters of the neural net should solve it, however even a small net having only 5 inputs and 3 hidden nodes also showed this behaviour.
A solution such a weight regularization is not possible here, since we typically want the weight of the latest input node to be higher than the weight of the others, neither is dropout regularization, due to the shallowness of our network.

A possible solution could be to have a bigger training dataset (so that the network avoid learning only locally relevant noise pattern), or to stick to a single input structure.

5.11.2 Delta method

The Delta method for producing PI was also implemented in the first part of the semester project. It basically does a point regression, then rely on a first order approximation of the function in order to the uncertainty quantification [5]
It was later dropped because the literature proved it's inferiority compared to Bootrap and Lube [5], and it would also suffer from a similar lack of generality as the LUBE method.

5.11.3 Reconstruction error via LSTM

Another exploration idea was to do drift detection only on the **X** values (ignoring the **Y**), via **reconstruction error monitoring.**

Here is the procedure:

1) Train an LSTM cell onto **normal data** to **reconstruct** the signal.
    a. That is, given a window of W sample (sample X_{t-W} to X_t), the LSTM is trained to reconstruct sample X_{t+1}
2) Apply this LSTM onto the Test dataset, and monitor the **reconstruction error.**

$$Reconstruction\ error = X_{t+1} - \widehat{X_{t+1}}$$

3) If the reconstruction error gets bigger than a certain threshold, trigger a drift detection.

More info detailed in this TowardDataScience article: Time Series of Price Anomaly Detection with LSTM [10]

This procedure was tried onto our dataset without success: The validation loss would not converge during the training:

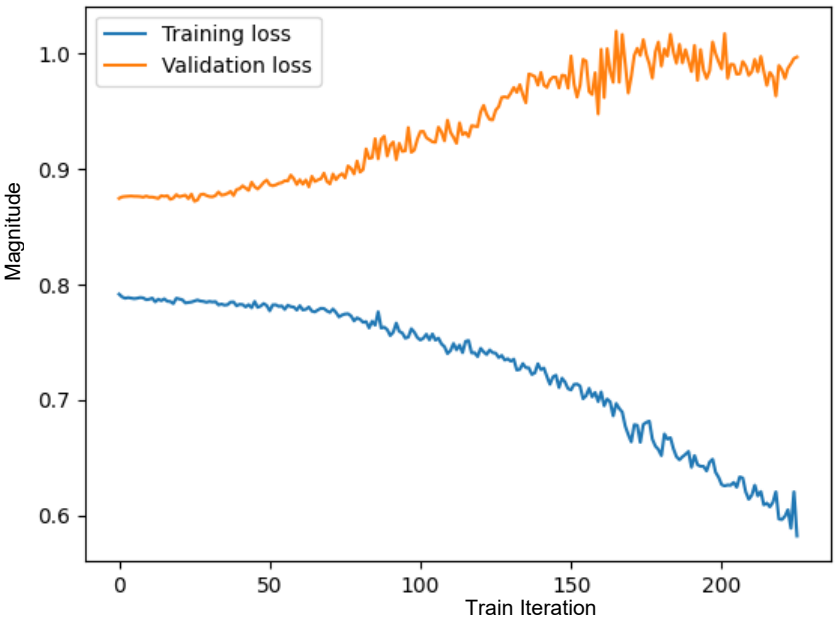


Figure 20: Training and validation loss during the training of the LSTM net

A first (wrong) hypothesis was that the net was overfitting, various attempt at reduce this overfit (such as diminishing the parameters of the net, adding dropout) did not succeed.

A more convincing explanation is that since the **X** sample are sampled from a gaussian with constant mean during training (Figure 21), the net is basically trying to predict Gaussian noise, which is not something possible. It therefore learns wrong “pattern” present only in the train set.

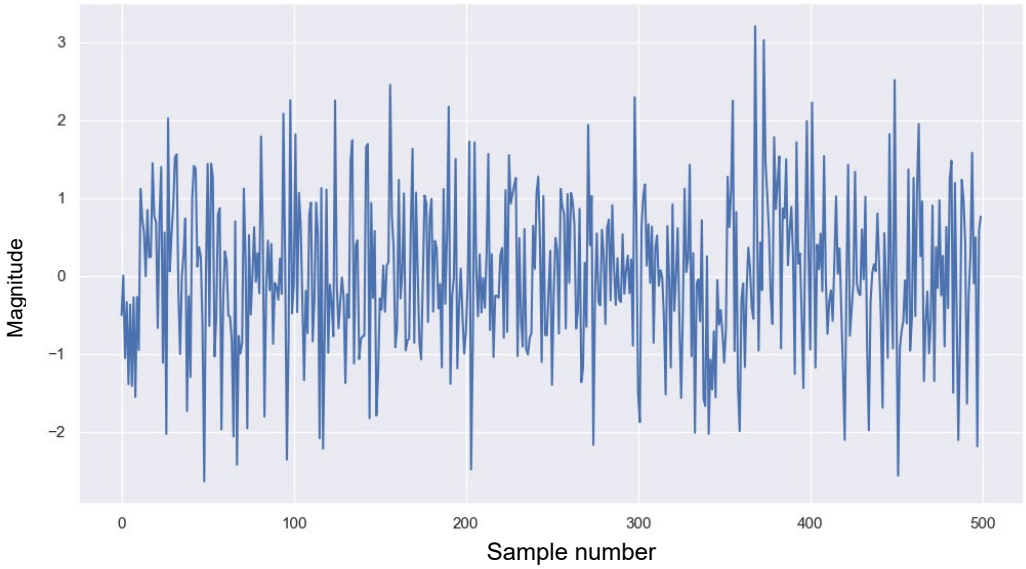


Figure 21: Representation of the X train data, one can notice the lack of identifiable structure.

6 Conclusion

In this work, we have applied regression methods from the Prediction Interval field to the problem of drift detection.

Due to the specificity of our problem, we have shown that some state-of-the-art method such a LUBE does not perform well on our dataset. We have understood and explained the theoretical reasons behind it.
Additionally, we have implemented the Bootstrap method, demonstrated it’s capability and analysed it’s results over our dataset.

Probabilistic regression technic is an interesting yet complex way to tackle the problem of anomaly detection, and the absence of the $\boldsymbol{Y}$ value during test might limit its full potential.

Also, the toy example presented here being quite simple, it would be interesting to compare their performance found in this report to a simpler standard machine learning approach and using only the $\boldsymbol{X}$ values.

6.1 Further direction

- In the literature, the keywords around “anomaly detection” or “outlier detection” lead to way more results than “drift detection” or “uncertainty quantification”, there could be some interesting paper and method to implement, especially deeper architecture (which would need a bigger training set).
- There is some popular python library for outlier detection, which allow an automated search with a lot of different algorithm and would avoid the lengthy process of algorithm reimplementation.

  However, having access to the $\boldsymbol{Y}$ value during train and not during test is quite specific to our problem, it is likely that these libraries do not support such feature, so one might have to do the classification only with the $\boldsymbol{X}$.

  Here are 2 popular libraries
  PyOD:
  Very popular and mature library for Outlier detection.
  https://github.com/yzhao062/pyod

  ToDS:
  Adaptation of PyOD specifically for time series, it is however a bit more recent and less mature:
  https://github.com/datamllab/tods

7 References

[1] Marco A.F. Pimentel, et al. "A review of novelty detection". Signal Processing 99. (2014): 215 - 249..

[2] Wang, Siqi & Zeng, Yijie & Liu, Xinwang & Zhou, Sihang & Zhu, En & Kloft, Marius & Yin, Jianping. (2020). Self-supervised Deep Outlier Removal with Network Uncertainty and Score Refinement..

[3] PyOD documentation, a popular outlier detection librairy, https://pyod.readthedocs.io/en/latest/relevant_knowledge.html.

[4] Amarbayasgalan, T.; Pham, V.H.; Theera-Umpon, N.; Ryu, K.H. Unsupervised Anomaly Detection Approach for Time-Series in Multi-Domains Using Deep Reconstruction Error. Symmetry 2020, 12, 1251..

[5] H. M. D. Kabir, et al. "Neural Network-Based Uncertainty Quantification: A Survey of Methodologies and Applications". IEEE Access 6. (2018): 36218-36234..

[6] A. Khosravi, S. Nahavandi, D. Srinivasan, & R. Khosravi (2015). Constructing Optimal Prediction Intervals by Using Neural Networks and Bootstrap MethodIEEE Transactions on Neural Networks and Learning Systems, 26(8), 1810-1815..

[7] Hao Quan, et al. "Particle swarm optimization for construction of neural network-based prediction intervals". Neurocomputing 127. (2014): 172 - 180..

[8] PySwarm Documenttion homepage, https://pyswarms.readthedocs.io/en/latest/.

[9] Abbas Khosravi, et al. "Quantifying uncertainties of neural network-based electricity price forecasts". Applied Energy 112. (2013): 120 - 129..

[10] TowardDataScience article about LSTM : https://towardsdatascience.com/time-series-of-price-anomaly-detection-with-lstm-11a12ba4f6d9.