Lina BEN-YOUNES

Supervised by Dr Renson

Project Final Report

Automatic Detection of Mode Interactions Using Machine Learning

ME4 Individual Project

Imperial College London

Department of Mechanical Engineering

Word count: 9,157

Word limit: 10,000

June 8th 2023

Abstract

This report examines how coupled vibration modes could be automatically detected using well-established Machine Learning techniques. Combining the use of wavelet transforms and support vector machines, the presented method detects interactions between the fundamental frequency and harmonics up to five, with an accuracy of over 95%.

In order to train the support vector machines, scalogram databases were computed using the Morse wavelet transform from the MATLAB Wavelet Toolbox. The time series analysed were obtained by integrating beam equations using Python's scipy.integrate module.

The report also presents a method which does not rely on Machine Learning techniques but rather on hard-coded logic intended to mimic the reasoning physicists would use to discern interactions. That method is less computationally demanding and allows for more flexibility in defining the threshold for detecting interaction. Still, it fails to exploit the large amount of information the scalogram gives.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr Ludovic Renson, for the opportunity to work on this project. I am grateful for his invaluable patience, support, and feedback, for his continued guidance throughout the project and for providing his knowledge and expertise, while encouraging me to take initiative.

Table of contents

Introduction and Objectives

Context

Confronted with complex nonlinear structures, engineers analyse various nonlinear responses, one being coupled vibration modes. In a nonlinear system, vibrational modes may influence each other, affecting individual behaviour and characteristics. These mode interactions can lead to changes in modal frequencies and energy transfer between modes.*[1] [1]*

When subjected to high forcing, they pose a challenging threat to the integrity of the concerned structures. Indeed, energy transfers from a local mode of low effective mass components to a global mode with high effective mass might critically jeopardise the structure. *[2]*

Nowadays, these interactions are detected by trained-eyed physicists and engineers. They conduct frequency sweeps and inspect scalograms of the structure's response. This time-consuming verification could be automated using well-established Machine Learning techniques. It is the aim of this project.

Objectives

Firstly, it is necessary to learn how to recognise mode interactions and understand the conditions and parameters that favour them.

Physicists usually rely on scalogram analysis to detect interactions. The algorithms will have to be trained with those scalograms, which entails creating scalogram datasets and classifying them according to whether they present modal interactions. We will also have to write scaling and pre-processing algorithms to obtain clean data and increase the accuracy and efficiency of the machine learning models.

We will then train and test different Machine Learning models and techniques in order to select those best suited for this classification problem. Finally, the algorithms will be tested on real-life data and examples.

To summarize, the project goals are:

- Identifying the patterns, parameters, and favourable conditions of modal interactions.

- Creating scalogram datasets of nonlinear models with and without model interactions.

- Creating a procedure to pre-process and standardise the data format on which the support vector machines (SVM) are trained and used.
- Training and adjusting the parameters of the SVM.
- Testing them on real-life data.

Literature Review and General Concepts

Nonlinear Normal Modes and Mode Interaction

Mode interactions are a prominent feature of Nonlinear Normal Modes (NNMs). If the system's natural frequencies are commensurate, an energy exchange between the different modes may be observed during the internal resonance. *[3]*

When exciting one mode, one may produce a large-amplitude response on another. Such interactions are identified as n:m interactions, meaning the n-th harmonic of a mode matches the m-th of another.

Such interaction may therefore be observed when the system:

- Is non-linear: The resonances are decoupled if the system is linear.
- Is subject to high forcing: for low forcing, the equations governing the system may be approximated by their linearisation.
- Its frequencies f_i and f_j are approximately such that: $k.f_i = p.f_j$, for low integer values of k and p. We may observe k:p interactions.

In such cases, mode interaction would be identified by high amplitude coefficients for frequency f_j when the system is subject to excitation of frequency f_i. So, in the case of $1:i$ interactions, the scalogram would present high amplitude coefficients for the i-th harmonic. An example can be found in *[4]*.

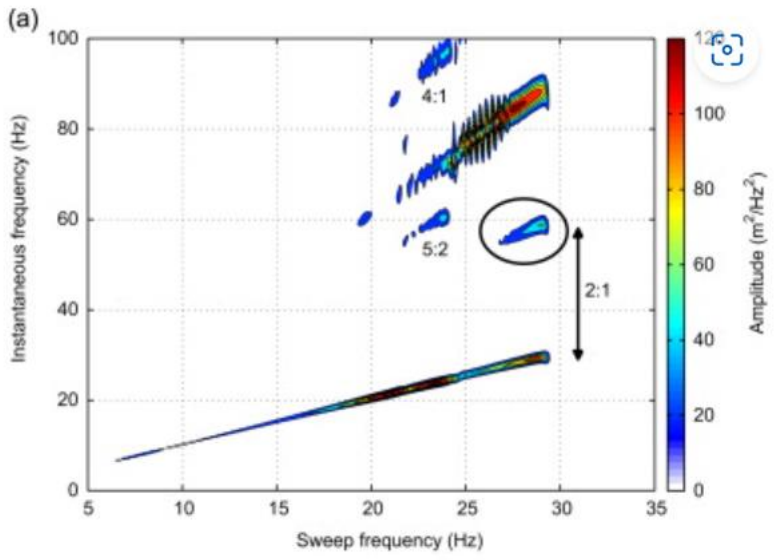


Figure [1] Scalogram presenting mode interaction from *[4]*

The scalogram in *Figure [1]* displays the response of a small satellite to a frequency sweep. One identifies the 2:1 interaction, hereby circled, between two internally resonant modes of the structure. The second harmonic of mode 3 matches the frequency of mode 7. Mode 3 involves an out-of-phase motion of the inertia wheel, and mode 7 consists of an axial motion of the telescope supporting panel. *[4]*

Continuous Wavelet Transform & Scalograms.

Wavelet transforms are mathematical tools used to analyse changing features in data. As the response to a frequency sweep is studied, the feature analysed is frequency.

Wavelet transforms address the limitations of the Fourier transform. While Fourier analysis quantifies the contributions of sine waves of specific frequencies over the full length of the signal, wavelet analysis decomposes signals into wavelets shifted in time and scaled in frequency. A wavelet is a rapidly decaying, wave-like oscillation. This enables wavelets to represent data across multiple scales. *[5]*

Wavelet transforms can be classified into continuous wavelet transforms (CWT) and discrete wavelet transforms (DWT). Unlike DWT, discrete approximations of CWT are redundant in information. However, DTW only allows for a restrictive choice of wavelets and is not shift-invariant, the wavelet analysis method used throughout the project is CWT and therefore, this section will cover continuous wavelet transforms. *[6]*

We should think of the graph of ψ as a single 'cycle', the wavelet ψ is assumed such that:

- It is compactly supported.
- $\psi \in L^2(\mathbb{R})$
- $\int_{\mathbb{R}} \psi(t)dt = 0$

But it may be chosen such that it has additional properties such as smoothness. *[7]* The mother wavelet ψ is dilated by the scale $a > 0$ and shifted to the position $b \in \mathbb{R}$. We define:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)$$

The scale parameter a is inversely proportional to the instantaneous frequency at time b. We shall therefore define the continuous wavelet transform $W(f)(a,b)$ such that we get the relation:

$$f(t) = \frac{1}{c_\psi}\int_{(a,b)\in(0,\infty)\times\mathbb{R}} W(f)(a,b)\,\psi_{a,b}(t)\frac{da\,db}{a^2}$$

Where c_ψ is a constant. *[7]*

The continuous wavelet transform of $f \in L^2(\mathbb{R})$ corresponding to a choice of wavelet ψ is:

$$W(f)(a,b) = \int_{\mathbb{R}} f(t)\,\overline{\psi_{a,b}(t)}\,dt, \qquad a > 0, \qquad b \in \mathbb{R}$$

$$c_\psi = \int_{\mathbb{R}^+} \frac{|\mathcal{F}(\psi)(\xi)|^2}{\xi} d\xi$$

As mentioned in the abstract, physicists use scalograms to characterise mode interaction. Scalograms are graphs displaying the absolute value of the CWT of a signal, plotted as a function of time and frequency, as seen in *Figure [1]*.

Support vector machines

The scope of the project was limited to $1{:}\,i$ interactions for values of i< 6. The detection of mode interactions corresponds to a classification problem. The different classes are:

- No interaction
- 1:2 interaction
- 1:3 interaction
- 1:4 interaction
- 1:5 interaction

The common Machine Learning classification techniques are Naïve Bayes, Random forests, Support Vector Machines (SVMs), and K-Nearest-Neighbours (KNN). The choice of SVMs for this project is justified by the format of the data used in this project.

Scalograms were transformed into a set of vectors each corresponding to a time sample of the scalogram, each of these vectors must be attributed a class. The dimension of the vectorial space is 120.

- Naïve Bayes relies on strong independence assumptions between the features of the data, which are not valid on the constructed dataset.
- K-Nearest-Neighbours: KNN is very computationally expensive, not only when trained but also when used to predict a class. Therefore, it would not be suited to classify data in a vectorial space of dimension 120.
- Random forests are a collection of decision trees. This method is best suited for data that are a mixture of numerical and categorical features, which is not the case in this project.

This section will therefore cover SVM algorithms only. The aim of Support Vector classification is to find optimal separating hyperplanes in a high-dimensional feature

space. There are several ways to define the optimality of the hyperplane, in this project, the maximal margin classifier was used. *[8]*

This means that SVM considers the training points at the extreme of their classes, i.e., the support vectors, and places the decision boundary, or separating hyperplanes, as far from these as possible. The margin is the shortest distance between the support vectors and the boundary. The SVM algorithm searches for a decision boundary that would maximise the margin.

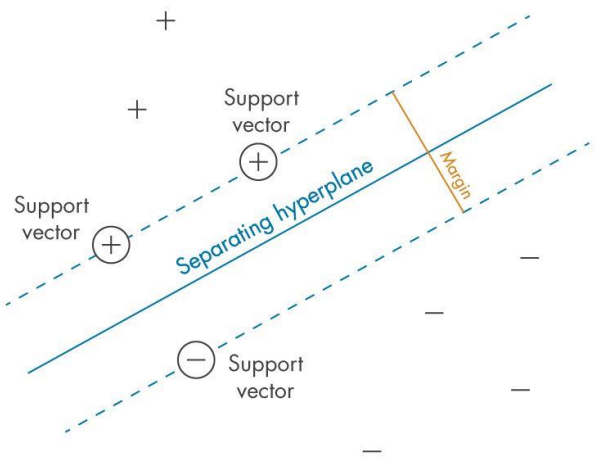


Figure [2] 2D example of Support Vector Machine *[9]*

Linear SVM

Let us first focus on linear decision boundaries. The decision hyperplane is such that the linear discriminant function cancels. The linear discriminant functions is:

$$g(x) = \omega^t\, x + \omega_0$$

Where ω is the weight vector and ω_0 is the bias or threshold weight. The weight vector is orthogonal to the decision hyperplane. The class of point x is determined by the sign of $g(x)$. And the margin is the smallest value of $\{g(x),\ x \in training\ set\}$.

In the case of multi-category classification, the usual approach is to define one linear discriminant function g_i for each class i. The class of a point x is then j such that $g_j(x) = max\{g_i(x),\ i \in possible\ classes\}$.

Kernels

Kernels are used to learn non-linear relations with a linear machine. Non-linear machines are built in two steps: a fixed non-linear mapping transforms the data into a feature space F, and a linear machine classifies them in F. *[10]*

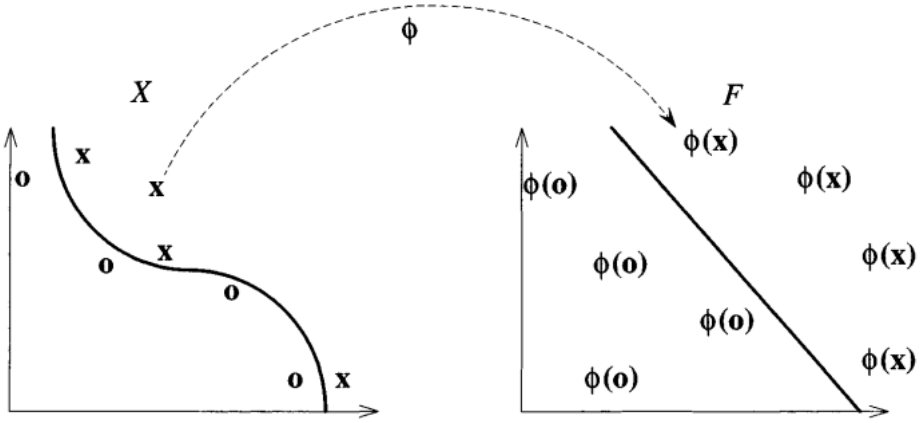


Figure [3] A feature map for kernel ϕ from *[10]*

The common kernel functions are polynomial, Radial Basis Function (RBF) and sigmoid. Those kernels necessitate the introduction of parameter is γ, which defines how much influence a training sample has on the hyper-plane position. Parameter C influences how close the data is fitted if soft margin SVMs.

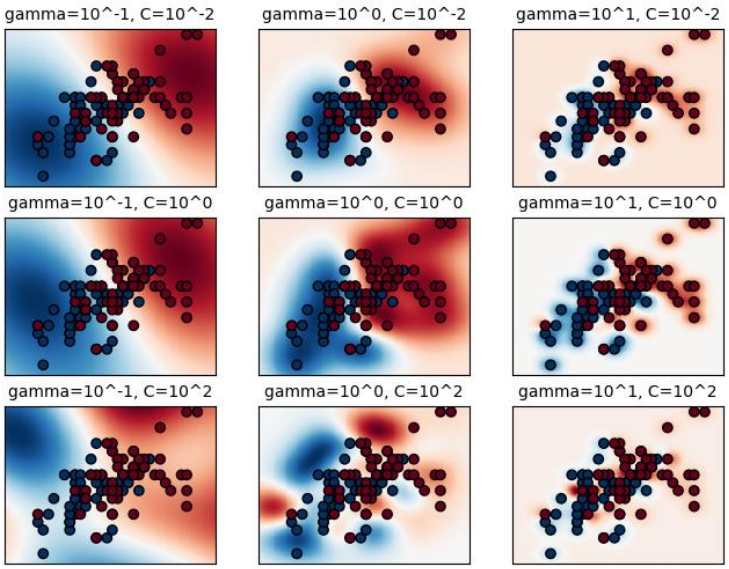


Figure [4] Influence of parameters γ and C on an RBF model *[11]*

Soft/Hard Margin

In the case of a hard margin, no support vector can be on the wrong side of the boundary, this is at the expense of a more complex decision boundary. A soft-margin SVM gives a smoother decision boundary by misclassifying some training points. *[12]*

(a) Hard Margin SVM

(b) Soft margin SVM

Figure [5] Comparison between Hard and Soft Margin SVMs *[12]*

Soft margin consists in introducing a parameter C called cost or penalty, which controls the trade-off between the size of the margin and the slack variable penalty. A set of values ϵ_i are introduced. They correspond to the error being made of sample i. x_i is the sampled point and $y_i \in \{-1,1\}$ is the class. *[12]*

$$y_i(\omega^t x_i + \omega_0) \geq M(1 - \epsilon_i)$$

$\epsilon = 0$ means the point x_i lies on the correct side of the margin. $\epsilon > 0$ means the point x_i lies on the wrong side of the margin, and $\epsilon > 1$ means the point x_i lies on the wrong side of the classification hyperplane. *[12]*

Finding the parameters of the SVM is then an optimisation problem:

$$\min\left(\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n}\epsilon_i\right)$$

Increasing the value of C will therefore increase the accuracy as well as the complexity of the boundary hyperplane, which might result in overfitting the data.

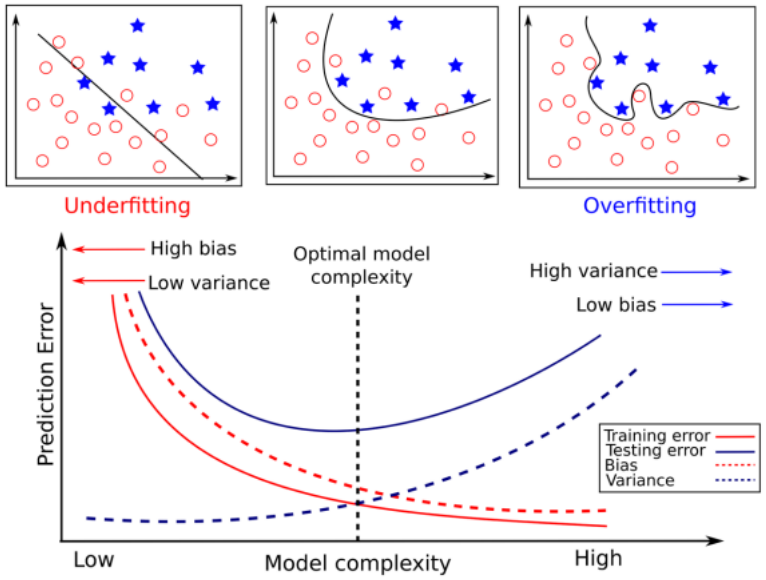


Figure [6] Model complexity and accuracy *[13]*

Performance evaluation

K-fold cross-validation

K-fold cross-validation is used to assess performance by averaging across several divisions of a training set. The steps are:

1. Split all the training data in K equally sized random subsets.
2. Use K-1 of the subsets to train the model.
3. Use the last remaining subset to check the performance of the model.
4. Discard the model and repeat the process another K-1 times, until the model has been tested on every subset.

K-fold is used to confirm that the model is suitable, in particular, to check that it avoids overfitting, which would be noticeable by a significant gap in performance between the training dataset and the testing set.

Receiver operating characteristic

Receiver operating characteristics (ROC) graphs are useful for organizing classifiers and visualizing their performance. Let us first introduce a few metrics. In the case of a binary classification, one may use the confusion matrix:

		True class	
		Positive	**Negative**
Hypothesized class	**Positive**	True Positive (TP)	False Positive (FP) Type I error
	Negative	False Negative (FN) Type II error	True Negative (TN)
Column Totals		P	N

The following performance metrics are then defined:

- Precision quantifies the ability to correctly identify positive cases without including false positives. *Precision* = $\frac{TP}{TP+FP}$
- Sensitivity quantifies the ability to identify the true positives while minimizing false negatives. *Sensitivity* = $\frac{TP}{P}$
- Specificity measures the ability to identify negative cases correctly out of all the actual negative cases. *Specificity* = $\frac{TN}{TN+FP}$
- Accuracy measures the proportion of correct predictions or classifications out of all the cases. *Accuracy* = $\frac{TP+TN}{P+N}$

The ROC graph then illustrates the trade-off between sensitivity (true positive rate) and specificity (false positive rate) at different classification thresholds.

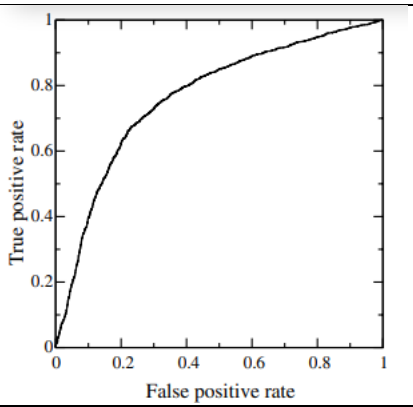


Figure [7] Example of a ROC graph *[14]*

The False Positive Rate (FPR) and True Positive Rate (TPR) from the ROC graph are not that of the confusion matrix. They are weighted by the number of actual positive cases for TPR and the number of actual negative cases for FPR.

TPR = $\frac{TP}{TP+FN}$, *FPR* = $\frac{FP}{FP+TN}$ *[15]*

Class Imbalance

Class imbalance refers to a situation where the distribution of classes in a dataset is significantly skewed, meaning that the number of instances belonging to one class is much higher or lower than the number of instances belonging to the other class.

In class-imbalanced scenarios, accuracy alone may not accurately assess a model's performance. A classifier that predicts only the majority class can achieve high accuracy, even if it fails to identify instances from the minority class correctly. In such cases, one uses the F-measure to assess the performance of a model. *[16]*

$$F\text{-}measure = \frac{2}{1/precision+1/sensitivity}$$

The F-measure accounts for the trade-off between precision and sensitivity, making it suitable for evaluating models in imbalanced datasets. However, the selection of that appropriate trade-off depends on the specific task and the relative importance of false positives and false negatives. *[16]*

In some cases, a type II error might have worse consequences than a type I error. Typically, when it comes to the detection of mode interactions, which could have security consequences, it might be preferable to have a higher rate of False Positives than False Negatives, meaning sensitivity is more important than precision, and maximizing the F-measure is not always desirable. *[16]*

Multi-class ROC analysis

Managing the entire space becomes more complex when dealing with more than two classes in classification problems. Instead of being a 2x2 matrix, the confusion matrix becomes a nxn matrix, where n represents the number of classes. This matrix contains n correct classifications along the major diagonal and n^2-n possible errors in the off-diagonal entries.

One of the common approaches to handling n classes is to generate n different ROC graphs, one for each class. This approach is known as the class reference formulation. ROC graph i plots the classification performance using class c_i as the positive class and all other classes as the negative class.

However, this formulation has a limitation, it compromises the insensitivity of ROC to class imbalance. For example, if a certain class becomes more prevalent, it might influence the performance of the classifier for another class. Despite this caveat, the class reference formulation can still be practical and provide reasonable flexibility in evaluation. *[16]*

Implementation

To automatise the detection of mode interactions, a Python package was created. Like most ML projects, the package was implemented in 3 phases: dataset building, pre-processing, and testing.

The Python Package.

The goal was to create a package containing the following:

- Three Python modules:
  - A module that implements the method relying on support vector machines. It will be referred to as the ML method or ML module throughout this report.
  - A module that implements the method by comparing the amplitude of harmonics. It will be referred to as the No_ML method or No_ML module throughout this report.
  - A module with functions to test the performance of the ML method. It will be referred to as the Model_Test.
- A demo Jupyter notebook that illustrates the applications of the modules and compares the results of the ML and No_ML methods.
- A parameters_test Jupyter notebook that tests the performance of the model for various parameters such as the threshold probability of interaction, the threshold definition of interaction, kernel and hyper-parameter of the Support Vector Machine.
- The dataset used to train the models in format .csv.
- The trained Support Vector Machine in format .pkl.

First Module: No_ML

The No_ML package is meant to address the issue of automation with no Machine Learning techniques. It would mimic the reasoning physicists would use to discern interactions. Its functions are:

Preprocessing(scalogram, frequencies, f0, fend) which returns the pre-processed scalogram and takes the following arguments:

- *scalogram* is the scalogram (matrix of size [time array size] x [frequency array size])
- *frequencies* is the list of corresponding frequencies.
- *f0* and *fend* are the initial and final frequencies in the conducted frequency sweep.

interaction2(scalogram, preprocessed =True, threshold=2, frequencies=None, f0=None, fend=None) returns a dictionary, the keys correspond to the interaction and the values are the interaction score for each interaction. Its parameters are:

- *scalogram* is the scalogram to analyse, it is assumed to be pre-processed, unless one specifies *preprocessed =False* and therefore must give values to the pre-processing parameters frequencies, *f0* and *fend*.

norm(scalogram, i, preprocessed =True, frequencies=None, f0=None, fend=None) returns the relative amplitude for the i-th harmonic, with the same arguments as previously.

Second Module: ML_method

interaction(scalogram, model=default, preprocessed =True, frequencies=None, f0=None, fend=None) has the same purpose as *interaction2* but relies on SVM.
model2(threshold=2, Xy='default', chosen_kernel='rbf', chosen_C=100, chosen_gamma='auto') creates and trains the SVM for the wished threshold. One can specify different kernels but the default is linear.
model(threshold=2, Xy='default', chosen_kernel='rbf', chosen_C=100, chosen_gamma='auto') serves the same purpose as *model2* but addresses the issue of class imbalance.

Third Module: Model_Test

Cross_validate(model, threshold=2, k=5) returns a Python dictionary containing the accuracy as well as the confusion matrix, precision, specificity and sensitivity for each class.
method_compare(Sets, ax, columns) plots the scores of the scalograms in the *Sets* for each method.

calculate_bias_variance(model, X, y, k=5) computes an estimate of bias and variance of the chosen *model* relying on K-Fold cross-validation.

ROC_probability(model, X, y, probas, k=5) plots the ROC graphs of the chosen *model*, and with the threshold probabilities in the list *probas*.

Time series: Beam equation

The most time-consuming part of the project has been the creation of the training dataset. The difficulties encountered were:

- Finding the right parameters to observe interactions.

- Finding the right libraries to compute scalograms.

I tried to capture the 1:3 interaction described in *[17]* using duffing equations with two degrees of freedom. *Figure [8]* displays the obtained scalogram using Python's PyWavelets library.

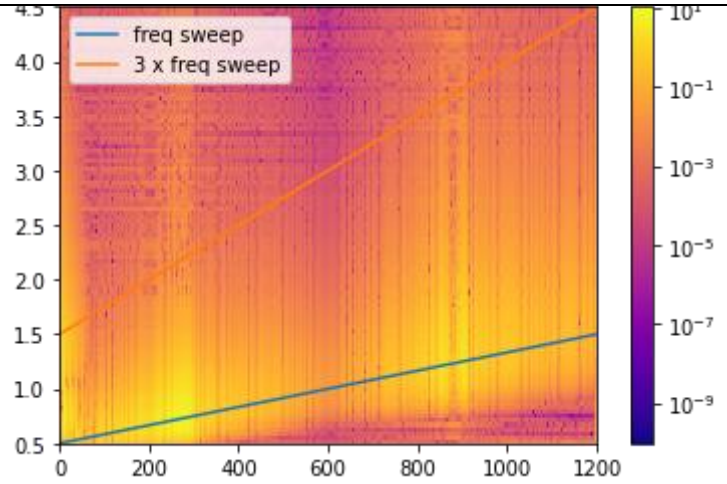
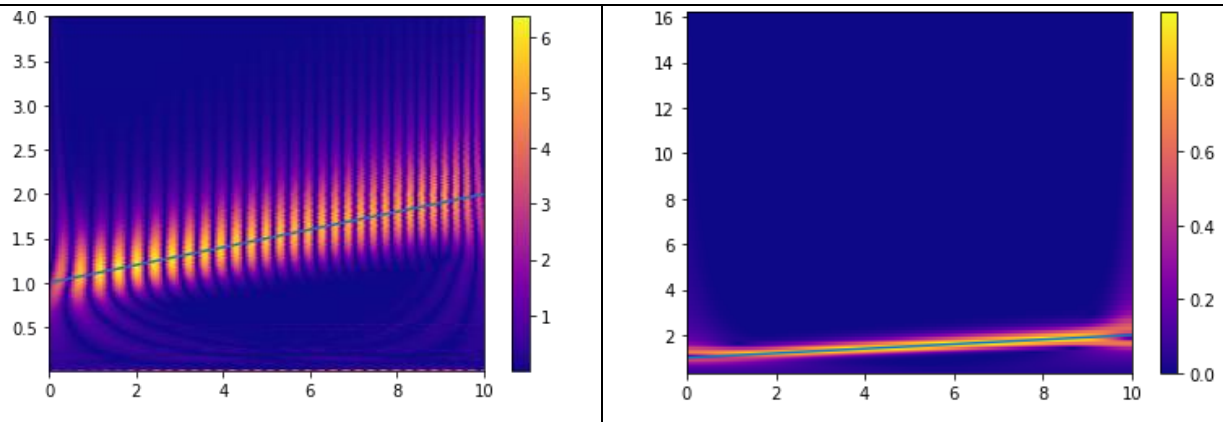


Figure [8] 1:3 interaction derived from the duffing equation in *[17]*

After experimenting with different parameters of the equation, I managed to obtain strong third harmonics as described in the paper. However, they were not as high as the ones I obtained with the beam equations in *[18]* (in some cases over 20 times stronger than the response amplitude of the excitation frequency).

As for computing scalograms, those computed with Python's PyWavelets fluctuated. The scipy.signal module gave better results but not as good as the continuous wavelet transforms computed by MATLAB's Wavelet Toolbox.



(a) Scalogram computed using Python's PyWavelets.

(b) Scalogram computed using MATLAB's Wavelet Toolbox.

Figure [9] Scalograms of frequency sweeps.

The final datasets were created using equations adapted from that of a cantilever beam with a nonlinear spring at the tip in *[18]*. In the paper, 1:3 interactions were deliberately introduced. The equations governing the system are:

$$\ddot{Q} + \omega^2 \dot{Q} + \Phi^T \, \mathrm{f}_{nl}\left(\Phi_{tip} \, \mathrm{Q}\right) + \delta \, \dot{Q} = excitation(t)$$

$$Y = \Phi \mathrm{Q}$$

With the following notations:

- $Y(t)$ is the parameter of interest, it is the static force–displacement of the beam resulting from the excitation computed at three different axial coordinates, and therefore a 3x1 matrix.
- $\mathrm{Q(t)}$ is a 3x1 matrix where Q_i is the i-th modal response of the system.

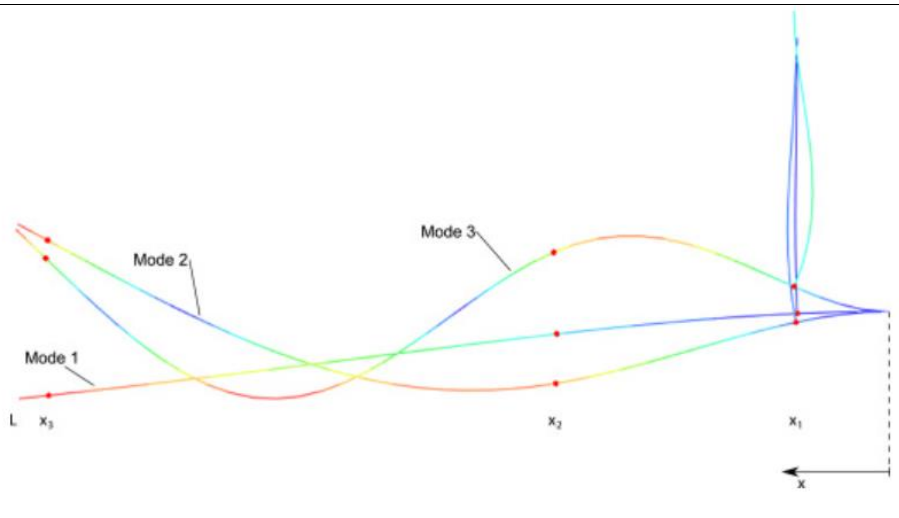


Figure [10] Mode shapes of beam and stinger. The red dots indicate accelerometer locations. From *[18]*

- Φ is the mode shape matrix. It is a 3x3 matrix such that $\Phi_{i,j}$ is the mode shape of the i-th mode at axial coordinate $\mathrm{x} = \mathrm{x}_j$
- Φ_{tip} is the mode shape vector at the tip of the cantilever.
- $\omega = \begin{bmatrix} \omega_1 & 0 & 0 \\ 0 & \omega_2 & 0 \\ 0 & 0 & \omega_3 \end{bmatrix}$, where ω_i is the resonant response frequency of the n-th mode.
- δ is the damping coefficient. $\delta = 2 \times damping\ ratio \times \omega$
- $excitation(t)$ was a frequency sweep.
- $f_{nl}(y) = k_1 \times y + k_2 \times y^2 + k_3 \times y^3$ is the non-linear function that introduces the mode interaction.

Various values of δ and k_i were used to create the database. However, the order of magnitude of k_i was between $10^{-2} \times k_{30} \times L^{3-i}$ and $10^{2} \times k_{30} \times L^{3-i}$, where $L = \sqrt{\frac{k_{10}}{k_{30}}}$ and k_{10} and k_{30} are the values of k_1 and k_3 respectively in the paper. This was done in order to add some variety to the dataset while maintaining a certain physical plausibility.

The resonant response frequencies were chosen such that $\omega_2 = k\ \omega_1$, for $\mathrm{k} \in \{1,2,3,4,5\}$.

The time series were computed using the fourth-order Runge-Kutta function of Python's SciPy library. The wavelet transforms were computed using MATLAB's cwt function in the Wavelet Toolbox.

Producing clean data: format and pre-processing

A scalogram corresponds to a list of three-dimensional data, a time value, a frequency, and an amplitude. The format of the scalograms computed by MATLAB is an array containing the values of frequency, and a matrix containing ($a_{i,j}$), where $a_{i,j}$ is the amplitude of frequency indexed i in the frequency array, at time t_j.

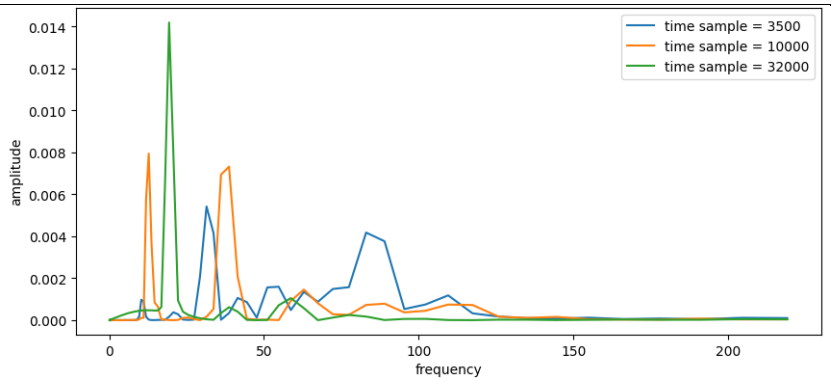
It makes no difference that the scalogram displays an interaction at time t_1 rather than t_2. We are interested in whether the scalogram presents interaction at all. The time parameter is, therefore, irrelevant to the identification problem. If the time array was of length n, the scalogram is split into n data points. Each data point is a frequency array and an amplitude array. *Figure [11]* (a) presents three of these data points.

The scalogram is then scaled along its frequency axis. The amplitude of the excitation frequency should be the same coefficient in the amplitude array. This is generalised to all the amplitude coefficients. The n-th coefficient in the amplitude array is the amplitude of frequency $\mathrm{f} = \mathrm{n}/20 \times \mathrm{f}_{\mathrm{excitation}}$, for $\mathrm{n} \in \{0, \dots, 119\}$. The standard frequency array is now $\mathrm{f}_{\mathrm{norm}} = (\frac{i}{20})_{i<120}$

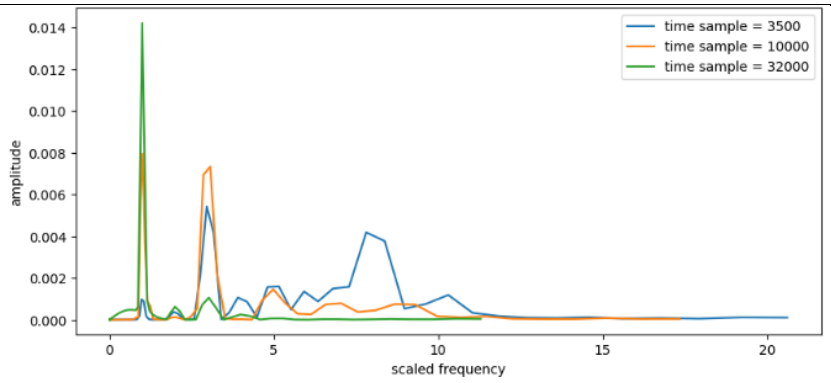
We chose to stop at $\mathrm{f} = 6 \times \mathrm{f}_{\mathrm{excitation}}$ because we limited the scope of the project to detecting interactions up to 1:5.

Using the frequency array (f_i) and the amplitude array ($a_{i,j}$), the data points are created by dividing (f_i) by the excitation frequency at time t_j. We then get $\mathrm{f}_{\mathrm{scaled}}$ and the corresponding amplitudes. Using interpolation, we compute the amplitudes corresponding to $\mathrm{f}_{\mathrm{norm}}$, a default value can be used to avoid extrapolation, as seen in *Figure [11]*. This is

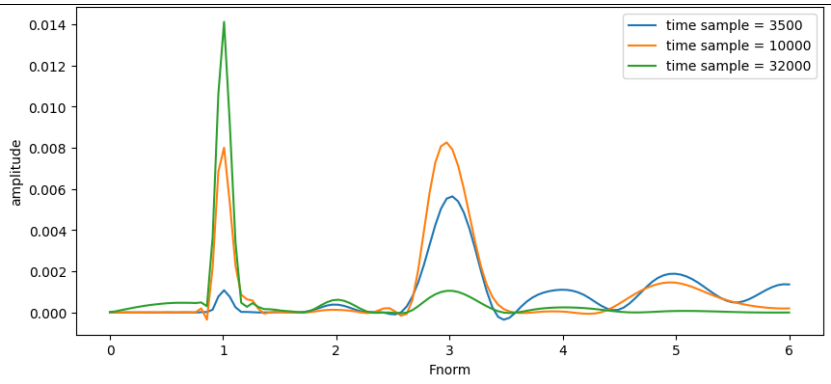
essentially what the function *preprocessing(scalogram, frequencies, f0, fend)* does in the Python package.



(a) data points before pre-processing



(b) Normalised frequency

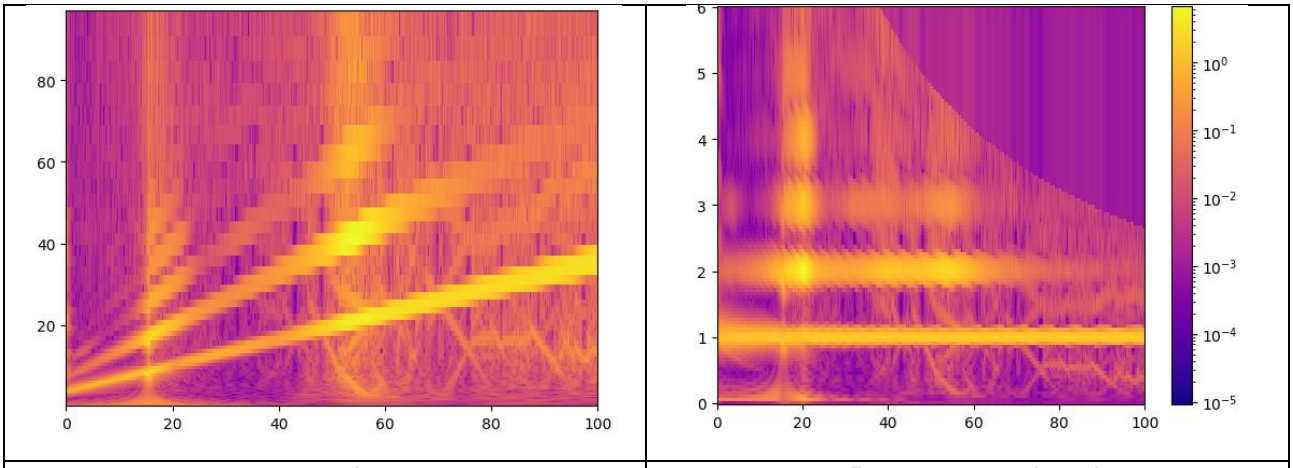


(c) Interpolated

Figure [11] 3 data points through the steps of pre-processing.

The last part of the pre-processing was to normalise the amplitudes, such that the amplitude of the excitation frequency would always be 1. The integral of amplitude over the excitation frequency was computed and the amplitude array was then divided by it.

An example of a pre-processed scalograms is presented in *Figure [12]* (b).



(a) Scalogram | (b) Pre-processed scalogram

Figure [12] Dataset presenting 1:2 interactions.

The top right part of (b) correspond to information that is not present in the original scalogram and is therefore replaced with a default value. That value is chosen to be the 25th percentile of the coefficient in the data point.

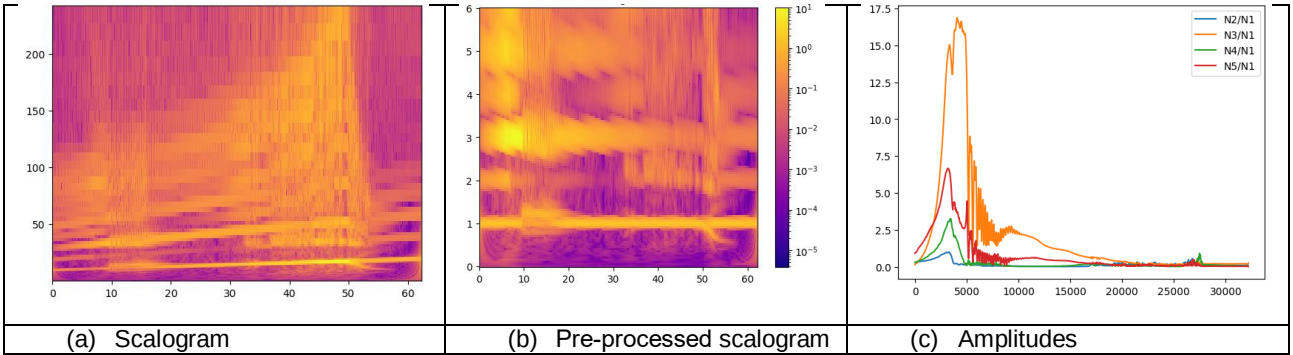
Class attribution for the training dataset

Class attribution

In the vectorial space of possible inputs, we define five functions:

- $n_i = \frac{1}{2\varepsilon}\int_{i-\varepsilon}^{i+\varepsilon} amplitude(f)\, df$, for $i \in \{2,3,4,5\}$

  n_i is the amplitude of the i-th harmonic. For each vector of the scalogram, $\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$ was computed, as seen in *Figure [13]* (c).



(a) Scalogram | (b) Pre-processed scalogram | (c) Amplitudes

Figure [13] Dataset presenting 1:3 interactions

The set of possible classes is {0,2,3,4,5}. 0 if there is no interaction, 2 if there is a 1:2 interaction, 3 if there is a 1:3 interaction, etc. However, when forming the dataset, if the

time series was obtained integrating a beam equation with $\omega_2 = 2\,\omega_1$, the possible classes for the vectors of the computed scalogram are 0 or 2.

So when forming the training set, if a scalogram should display a $1{:}\,i$ interaction, the vectors were classified with the following criteria:

- If $\frac{n_i}{n_1} > threshold$, the class of the vector is i.
- Otherwise, the class of the vector is 0.

The choice of the threshold will be a parameter of interest in the next section, and it is a parameter one can choose in the Python package. The No_ML module uses a similar logic to determine whether a scalogram displays an interaction, it checks if $\frac{n_i}{n_1} > threshold$ for all $i \in \{2,3,4,5\}$, which is unnecessary when it come to the training dataset as we already know which interaction the scalogram should be displaying.

The structure of the dataset is therefore:

- A file *X.csv* which contains 275077 vectors. Each vector corresponds to a time sample from the computed scalogram. The dimension of each vector is 120, accordingly with the pre-processing method.
- A *y.csv* file which contains information on the class of each vector. For each of the 275077 vectors from X, *y.csv* stores i, and $\frac{n_i}{n_1}$, with the previous notations. The shape of the y matrix is therefore 275077 x 2.

The dataset is structured that way so that the class of each vector is determined when the user chooses the threshold, meaning when the user defines what constitutes mode interaction.

Class imbalance

The last issue concerning the training set is class imbalance. As the class of the vectors from the training dataset is only computed once the user chooses a threshold, it is impossible to make the representation of each class in the set balanced prior to choosing said threshold.

Therefore, the function *model(threshold=2, Xy='default', chosen_kernel='rbf', chosen_C=100, chosen_gamma=0.1)* computes a new X, y such that all classes are represented equally, by getting rid of random samples in overrepresented classes.

Another approach could have been to use Cost-Sensitive Learning. In Machine Learning, the common cost-insensitive approach aims to minimize the probability of error,

assuming all misclassifications have the same cost. In cases of significant class imbalance, assigning higher misclassification costs for underrepresented classes makes the SVM's optimisation process biased towards reducing misclassifications on the minority class. *[19]*

Although the cost-sensitive learning approach doesn't include a loss of information, unlike under-sampling majority classes, it requires more computational resources, which is why I chose the former.

Format of the results

Two functions are used to assess the scalogram are *interaction(scalogram)* and *interaction2(scalogram, threshold)*. They both return a dictionary, with the interaction number as the key and the corresponding interaction score as the value.

The first function takes a scalogram as input and predicts the probabilities of different interaction classes using a Support Vector Machines. It then computes the interaction scores by summing the probabilities.

interaction2(scalogram, threshold) calculates interaction scores based on a given threshold and does not use Support Vector Machines. For each time sample, the amplitude of each harmonic is compared to the threshold. The interaction score for each harmonic is computed by summing the instances where the amplitude is superior to the threshold.

For both functions, the scores are then scaled by the number of time samples. The scores do not correspond to a probability of presence of interaction in the scalogram, but rather to the fraction of the scalogram presenting interaction.

It is moreover likely that the second function will give a higher score than the former as for each time sample, *interaction2* will add 1 to the score if it meets the threshold while *interaction* will return a probability inferior or equal to 1.

Support Vector Machine Parameters

The last step of implementation was to test the differences in accuracy, precision, sensitivity, and specificity for various parameters:

- Parameters C and γ.
- Choice of the kernel.

- Choice of the threshold.
- Does the automation of detection without ML techniques provide sufficient results.
- Does under-sampling majority classes improve performance metrics significantly?

And adjust the functions and parameters of the Python package accordingly.

Results and Discussion.

C and γ

For different values of C and γ, a 5-Fold cross-validation was run. The chosen threshold was 2, and the kernel was RBF. The cross-validation function coded for this project returns the accuracy, as well as the precision, sensitivity, and specificity for each class. The per-class results can be found in Appendix 3: performance per class for C and γ.

Table [1]: Accuracy results of Support Vector Machine hyperparameters

	C	0.1	1	10	100
γ					
0.1		98.76%	99.52%	99.72%	99.87%
1		97.35%	99.43%	99.69%	99.75%
10		86.29%	98.21%	98.02%	98.38%
100		66.95%	92.40%	92.83%	93.23%

Augmenting parameter C slightly improves the accuracy results, which is to be expected as a higher value of C increases the penalty for misclassifying samples. However, it does not seem to have a big influence on the performance of the algorithm, this usually indicates that the data is well-separated and easily classified, which is confirmed by the fact that the model performs better for low values of γ.

Lower values of γ allow for a smoother decision boundary that is less affected by individual data points. It indicates that the data points are well-separated or have distinct patterns that can be captured by a simpler decision boundary. *[20]*

In order to avoid overfitting, the data when choosing the parameter C, we will look at the variance and bias trade-off for different values of C. Using 5-fold cross-validation, an estimate of the variance and the bias as computed for increasing values of hyperparameter C.

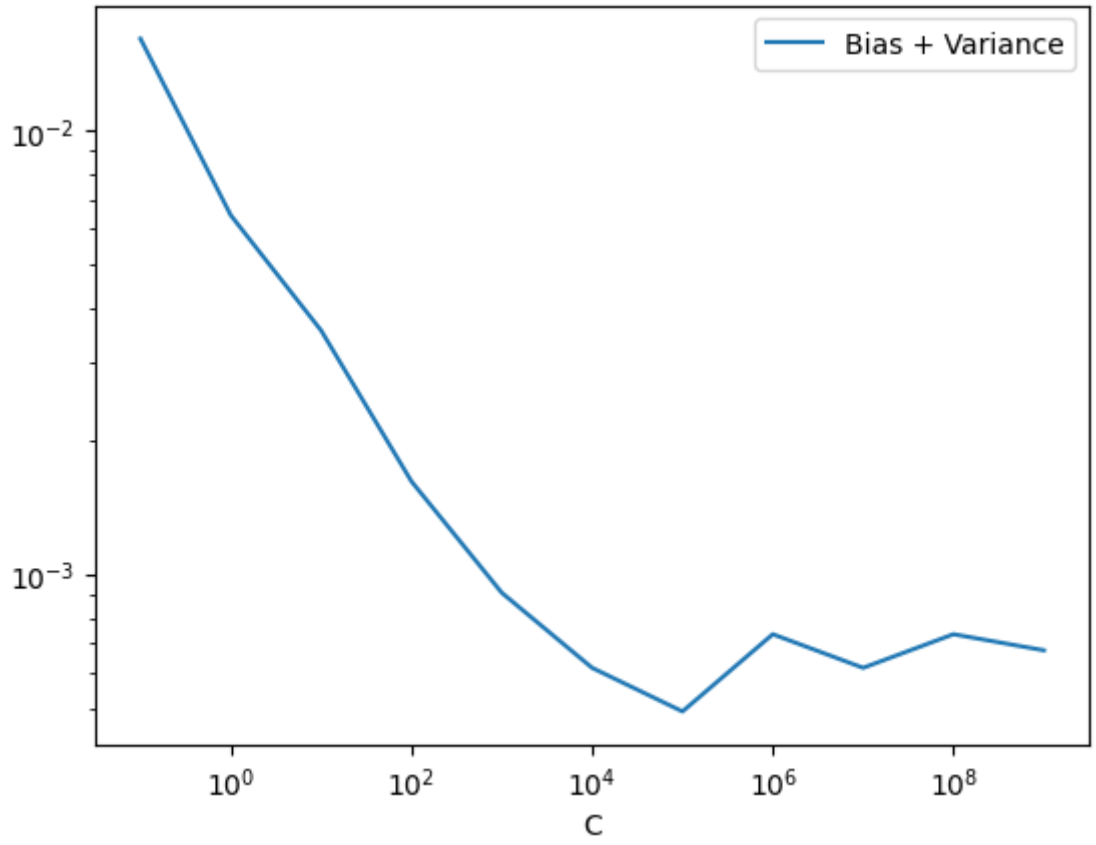


Figure [14] Bias variance trade-off for different values of C.

The rest of the testing was be conducted using C=100 and γ=0.1

Choice of kernel

The accuracy results for the RBF, linear and polynomial kernels are very similar.

Table [2] Accuracy results for Support Vector Machine kernels

Kernels	Accuracy
Linear	99.87%
Polynomial	99.46%
RBF	99.79%
Sigmoid	72.60%

It makes sense that the linear kernel would have a good accuracy as each sub-section i of the training set was classified using the following criteria:

$$n_i(vector) > threshold * n_1(vector)$$

Where n_i and n_1 are linear functions. The precision, sensitivity and specificity were computed for the RBF and linear kernels. They do not significantly differ. The RBF has a slightly better sensitivity for interaction classes. The RBF kernel was the one used for further testing.

Table [3] Performance metrics of Linear and RBF kernels

Kernel	Linear	RBF
Class 0: No interaction	precision: 99.93% sensitivity: 99.49% specificity: 99.98%	precision: 100% sensitivity: 98.98% specificity: 100%
Class 2: 1:2 interaction	precision: 99.93% sensitivity: 100% specificity: 99.98%	precision: 99.96% sensitivity: 100% specificity: 99.99%
Class 3: 1:3 interaction	precision: 99.78% sensitivity: 99.96% specificity: 99.95%	precision: 99.82% sensitivity: 100% specificity: 99.95%
Class 4: 1:4 interaction	precision: 99.85% sensitivity: 100% specificity: 99.96%	precision': 99.53% sensitivity': 100% specificity': 99.88%

Class 5: 1:5 interaction	precision: 99.93% sensitivity: 99.96% specificity: 99.98%	precision: 99.67% sensitivity: 100% specificity: 99.92%

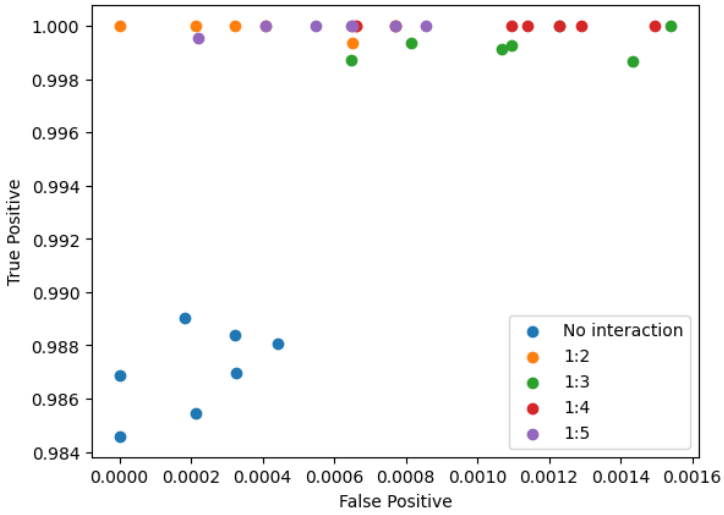
Choice of threshold

Threshold on the training data

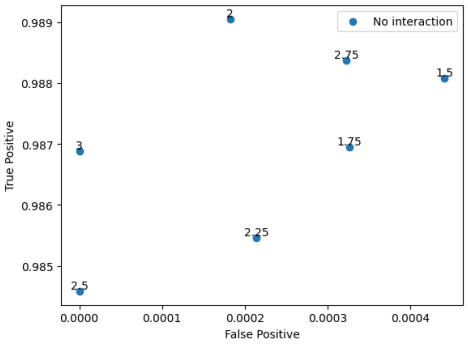
One must remark that the following graphs are an analogy to ROC graphs and not exactly ROC graphs. In actual ROC testing, X and y are known and do not change with the change of threshold. The same test is conducted on X and predicted class y_{pred} depends on whether the test meets the threshold.

In this case, the Support Vector Machines do not produce a result that must be compared with the threshold but rather they produce y_{pred}. The threshold intervenes in classifying the training data, i.e., in computing y.

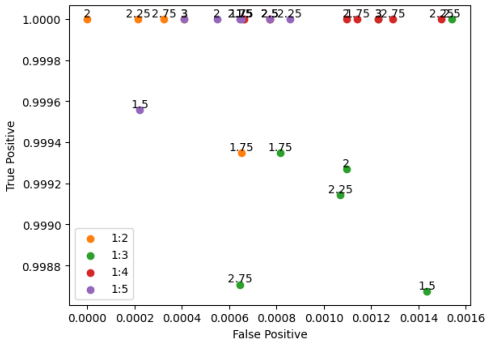
Therefore, the choice of threshold should not significantly impact the overall performance of the model. The interest is in comparing the relative impact of the threshold on the classes, as choosing a lower threshold means increasing the size of the dataset for interaction classes.



(a) Overall view



(b) Zoom on ‘No interaction’ class



(c) Zoom on ‘No interaction’ class

Figure [15] False Positives and True Positives for various thresholds and classes.

As expected, the choice of threshold does not significantly change the rates of True Positives and False Positives. One can also remark that the True Positive rate for the interaction classes is higher than that of the ‘No Interaction’ class for all thresholds, it could be explained by the fact that its vectors are more dispersed vectorial space of possible inputs. They were computed using a more diverse set of values for the resonance frequencies and other constants in the beam equation as explained in the Time series: Beam equation section.

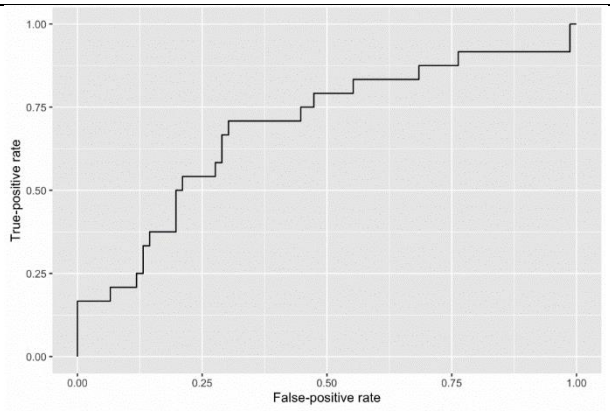
Threshold on probabilities

One way to implement actual ROC testing with Support Vector Machines, or other classification algorithms, is to have it compute, for each sample, the probability of it belonging to each class rather than simply give the class with the highest probability. The

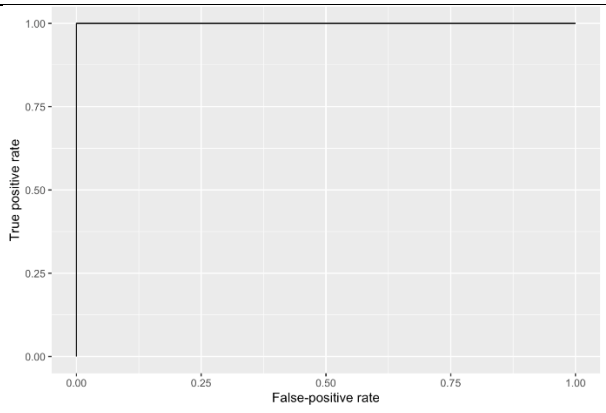
threshold would then be 0<p<1 such that if the probability of the sample belonging to one class is superior to p, the sample belongs to said class. *[21]*

Choosing a low threshold for p could be interesting in a detection problem as it could have more cost to miss interactions than it would be to have a 'No interaction' sample misclassified.

For comparison, ROC curves typically resemble that of *Figure [16]*(a). *Figure [16]* (b) displays the ROC graph of a model perfectly representing the data.



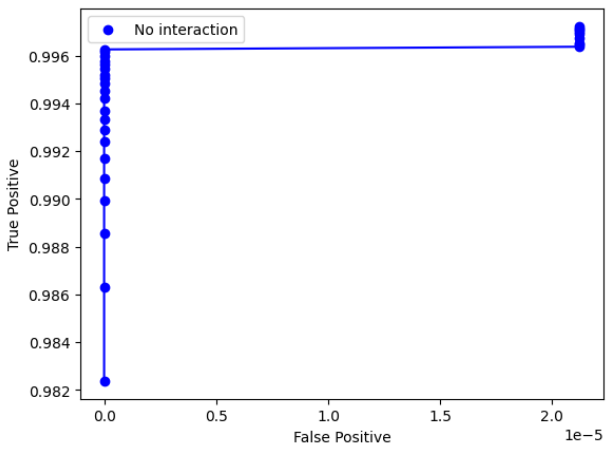
(a) Simulated ROC curve



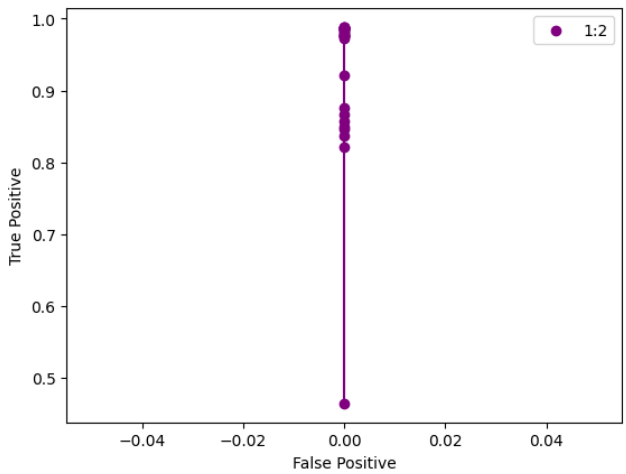
(b) ROC curve for a perfect model

Figure [16] ROC graphs from *[22]*

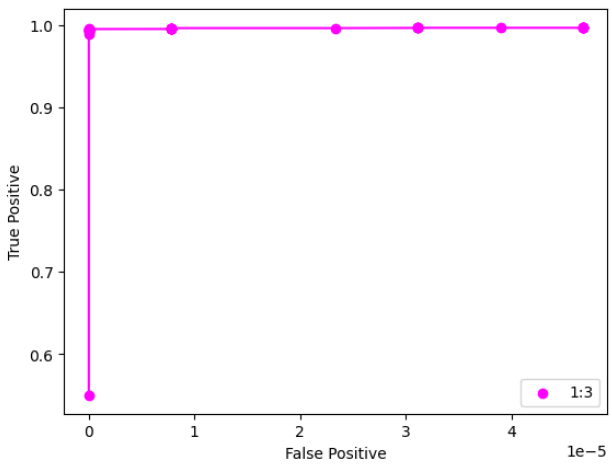
ROC curves were plotted for each class with probability thresholds varying from 0.2 to 0.999 and the following curves were obtained.



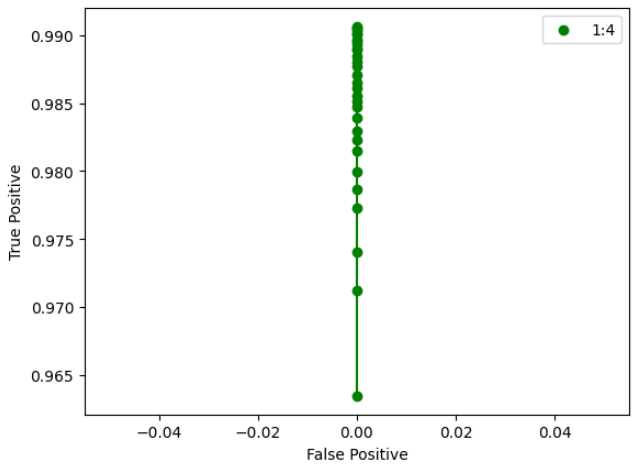
(a) Class: No Interaction



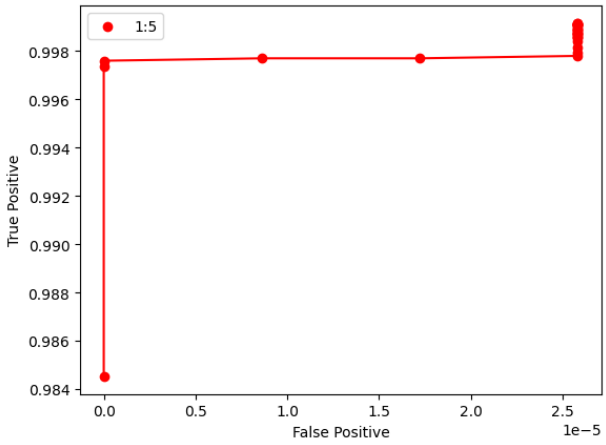
(b) Class: 1:2 interaction



(c) Class: 1:3 interaction



(d) Class: 1:4 interaction



(e) Class: 1:5 interaction

Figure [17] ROC graph. Threshold on probabilities

The curves resemble that of *Figure [16]* (b). The model performance in cross-validation is close to that of a perfect model. That is due to a very well-separated dataset. It could be explained by:

- a lack of variety in the training dataset

- the absence of noise

- a pre-processing which simplified the data to the extent that a non-machine learning approach can give satisfactory results.

Under-sampling majority classes

Two cross-validations were run for the same model as described previously, the kernel is RBF with C=100 and γ=0.1.

Table [4]: Comparison of performance metrics with and without class imbalance

Performance metrics		Under-sampling	Class Imbalance
Accuracy		99.80%	99.95%
Class 0 No Interaction	Precision	100%	99.96%
	Sensitivity	98.98%	99.96%
	Specificity	100%	99.90%
Class 2 1:2 interaction	Precision	99.89%	99.64%
	Sensitivity	100%	99.93%
	Specificity	99.97%	100%
Class 3 1:3 interaction	Precision	99.75%	99.85%
	Sensitivity	100%	99.87%
	Specificity	99.94%	99.99%
Class 4 1:4 interaction	Precision	99.64%	99.93%
	Sensitivity	100%	99.85%
	Specificity	99.91%	99.99%
Class 5 1:5 interaction	Precision	99.71%	99.95%
	Sensitivity	100%	99.94%
	Specificity	99.93%	99.99%

Under-sampling does not improve the model's accuracy. However, in class-imbalanced scenarios, accuracy alone does not accurately assess the model's performance. A classifier that predicts only the majority class can achieve high accuracy, even if it fails to identify instances from the minority class correctly.

Under-sampling did slightly improve the sensitivity to interactions, which is the aim of this project. Moreover, it has significantly reduced the time necessary to run the training and testing algorithms used throughout this project.

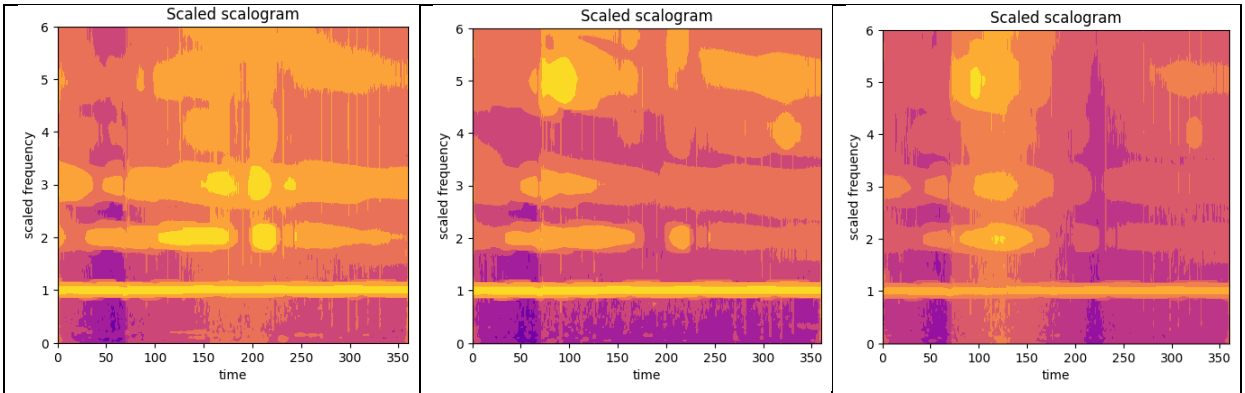
Experimental data & Comparison between the ML and No_ML methods

The models were used on experimental data. The proposed method is applied to a wing-engine structure with nonlinear connections. The time series exploited are that from *[23]*. They are provided by accelerometers mounted on the top of the wing plate measuring vertical acceleration and on two pylons measuring horizontal acceleration.

Figure [18] Wing-engine structure from *[23]*.

All the results and graphs can be found in Appendix 5: Pre-processed Scalogram of Wing Engine. The ones exploited here correspond to the sensors:

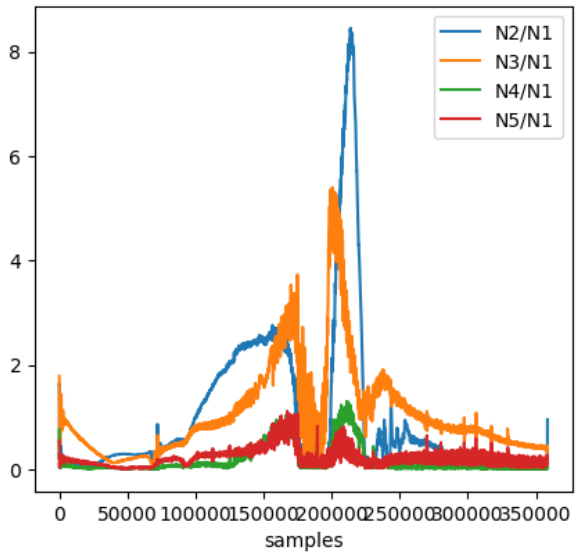
- Sensor 4 measures acceleration in the Z direction.
- Sensor 7 measures acceleration in the Z direction where the single-point shaker is located.
- Sensor 11 measures acceleration in the X direction.



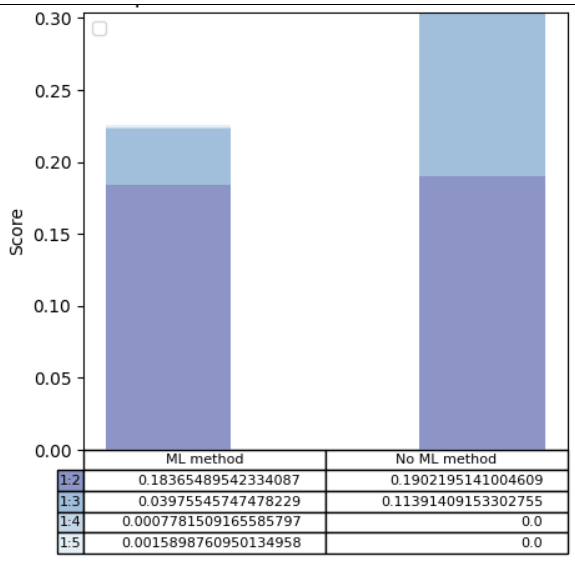
(a) Sensor 4 (b) Sensor 7 (c) Sensor 11

Figure [19] Pre-processed scalograms

It seems like Sensor 4 exhibits signs of 1:2 and 1:3 interactions, Sensor 7 exhibits signs of 1:5 interactions and Sensor 11 exhibits signs of 1:2 and 1:5 interactions. For each of these scalograms, $\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$ was plotted throughout time as well as the results provided by the No_ML and ML methods, they can be found in Appendix 6: $\frac{n_i}{n_1}$ and algorithm results of Wing Engine.



$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$

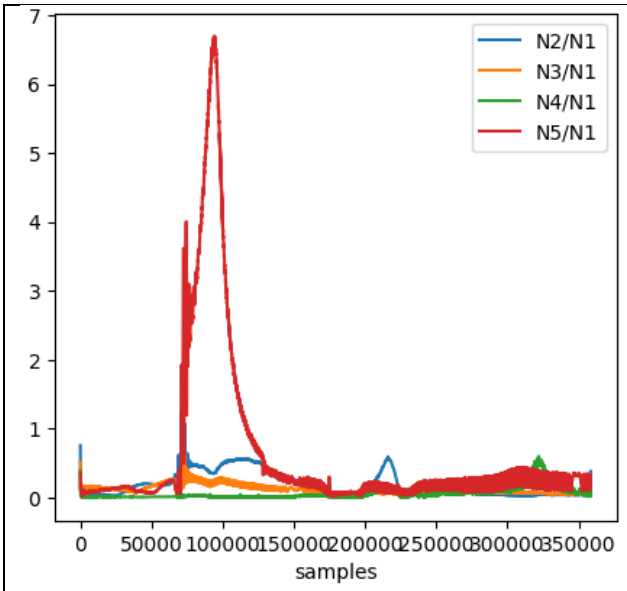


	ML method	No ML method
1:2	0.18365489542334087	0.1902195141004609
1:3	0.03975545747478229	0.11391409153302755
1:4	0.0007781509165585797	0.0
1:5	0.0015898760950134958	0.0

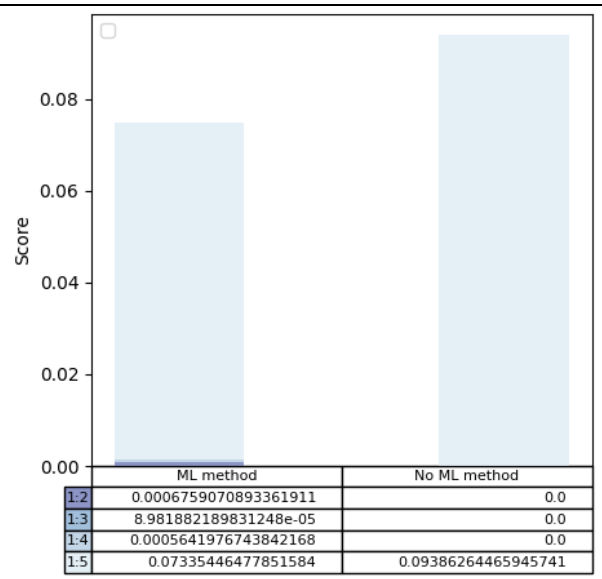
Comparison between ML and No_ML methods

Figure [20] Sensor 4 results

Both the ML and No_ML methods detect the presence of 1:2 and 1:3 interactions. However, there is a discrepancy in the scores of the ML method and No_ML methods. It can be attributed to the different computational approaches employed for calculating these scores, as explained in the section Format of the results.



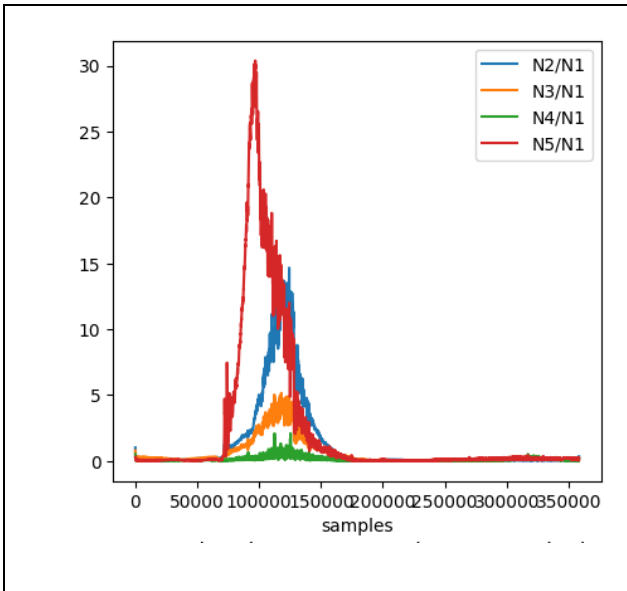
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$



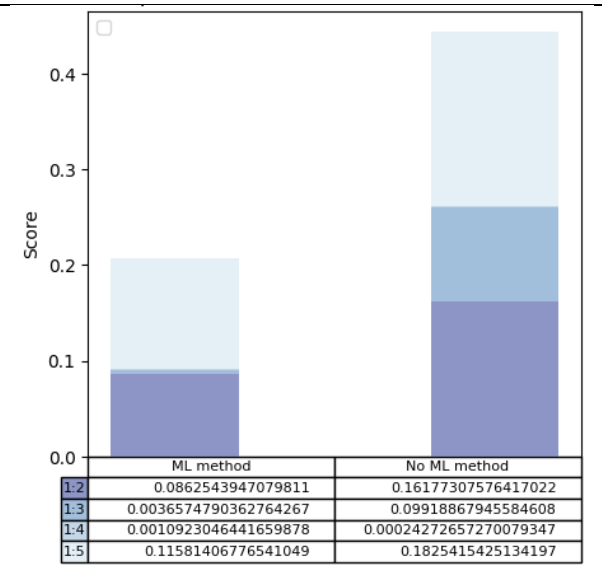
Comparison between ML and No_ML methods

Figure [21] Sensor 7 results

Both the ML and No_ML methods identify the 1:5 interaction. The relative low values of those scores (around 8%) does not correspond to a probability of presence of interaction, but rather to the fraction of the scalogram presenting interaction. As seen in the scalogram, most of the samples only shows the presence of harmonics (before sample 50,000 and after sample 150,000).



$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$

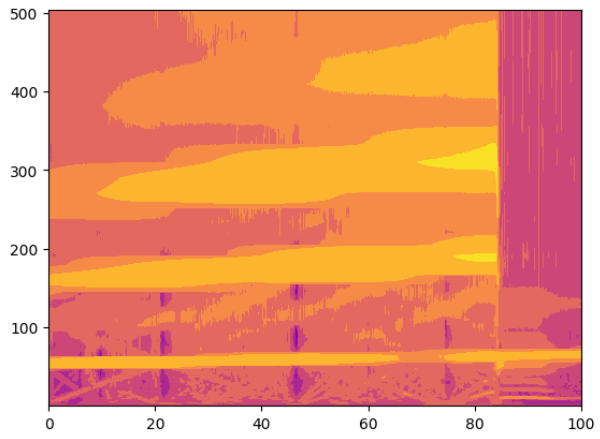


Comparison between ML and No_ML methods

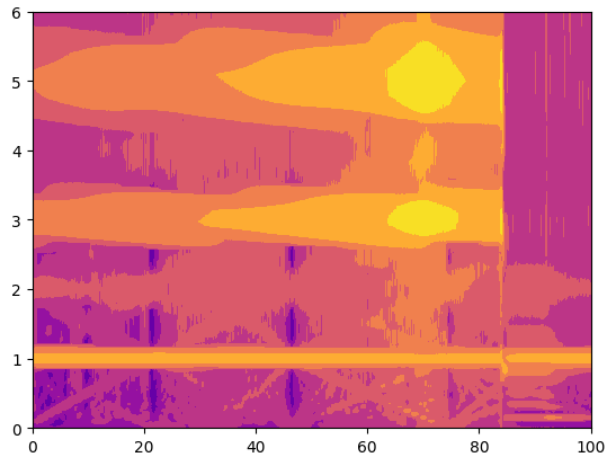
Figure [22] Sensor 11 results

The ML and No_ML methods give different results for Sensor 11. The interactions displayed on the scalogram in *Figure [22]* are probably 1:2 and 1:5. As identified by the ML_method. The No_ML method identifies a 1:3 interaction due to the high amplitude of the third harmonic. However, a high third harmonic is also to be expected in a 1:5 interaction scalogram.

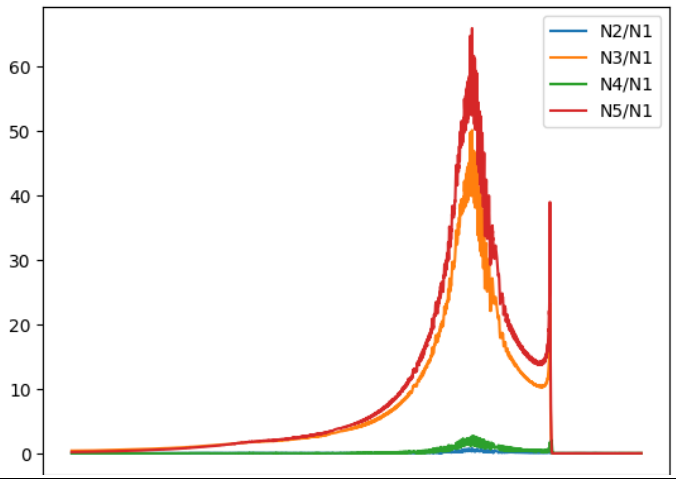
As explained in the section Format of the results, the logic used in No_ML algorithm is quite simplified in that it considers the amplitudes of the different harmonics independently. Therefore, it interprets a high amplitude of the third harmonic as a 1:3 interaction. For comparison, *Figure [23] Figure [21]* displays one of the 1:5 scalograms used to train the model.



(a) Scalogram



(b) Pre-processed scalogram



(c) $\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$

Figure [23] 1:5 interaction

Conclusions and Future Work

The methodology used in the project was designed to be adaptable to different datasets, different thresholds of what constitutes modal interaction, and different types of interaction. The test_parameters file enables the user to finetune the parameters for the chosen dataset.

The good performance of the models is due to a pre-processing that simplified the data to the extent that a non-machine learning approach gave satisfactory results, the absence of noise, as well as a lack of variety in the training dataset. Although different parameters were used (damping, forcing, polynomial coefficients of the non-linear function), it would be interesting in future work to derive time series from a more diverse set of equations.

Surprisingly, the model's performance did not improve significantly when addressing class imbalance. However, it revealed that the model was sensitive to the size of the training data set. The project would have benefitted of a larger training data set l, although it would require more computational power.

The No_ML module seems to give good results as well. It was able to flag that there is an interaction but was less good at discriminating which interaction occurs. The logic used in this algorithm is quite simplified in that it considers the amplitudes of the different harmonics independently. The algorithm could be refined so as to give results as satisfactory as that of the Support Vector Machines.

Future work could involve:

1. Creating a larger and more diverse training dataset.
2. Enlarging the scope of the project to incorporate more interactions, such as 2:3, 2:5, 3:4, etc.
3. Adapting the scaling function to be able to handle different frequency sweeps, such as logarithmic, step and chirp sweeps.
4. Reducing the dimension of the dataset using unsupervised learning techniques.
5. Complexifying the No_ML method to enable it to discriminate the interactions. It should compare harmonic amplitudes, not only with that of the excitation frequency but also with each other.

References

[1] Piccirillo, V., Tusset, A.M., Balthazar, J.M., Martinez, A.G. (2022). Nonlinear Modal Analysis of Vibrating Systems with Limited Power Supply. In: Balthazar, J.M. (eds) Nonlinear Vibrations Excited by Limited Power Sources. Mechanisms and Machine Science, vol 116. Springer, Cham. doi: 10.1007/978-3-030-96603-4_8

[2] J.P. Noël n , L. Renson, G. Kerschen, (2014). Complex dynamics of a nonlinear aerospace structure: Experimental identification and modal interactions. *Journal of Sound and Vibration, 333*(12), 2588-2607. doi: 10.1016/j.jsv.2014.01.024

[3] Kerschen, G., Peeters, M., Golinval, J., & Vakakis, A. (2009). Nonlinear normal modes, Part I: A useful framework for the structural dynamicist. *Mechanical Systems and Signal Processing, 23*(1), 170-194. doi: 10.1016/j.ymssp.2008.04.002

[4] Noël, J., Renson, L., & Kerschen, G. (2014). Complex dynamics of a nonlinear aerospace structure: Experimental identification and modal interactions. *Journal of Sound and Vibration, 333*(12), 2588-2607. doi: 10.1016/j.jsv.2014.01.024

[5] Inc. The MathWorks. (nd) *Wavelet Transforms in MATLAB.* https://fr.mathworks.com/discovery/wavelet-transforms.html4 [Accessed 20th December 2022]

[6] Inc. The MathWorks. (nd) *Continuous and Discrete Wavelet Transforms* https://uk.mathworks.com/help/wavelet/gs/continuous-and-discrete-wavelet-transforms.html [Accessed 22nd December 2023]

[7] Meyer, Y. Wavelets and spaces of functions and distribution. In: *Wavelets and Operators*. Cambridge, UK: Cambridge University Press; 1992. p. 163-207

[8] Cristianini, N., Shawe-Taylor, J. (2005). *Support Vector Machines*. In Cambridge University Press eBooks (pp. 93–124). doi: 10.1017/cbo9780511801389.008

[9] Inc. The MathWorks. (nd) *Support Vector Machines* https://fr.mathworks.com/discovery/support-vector-machine.html [Accessed 10th March 2023]

[10] Cristianini, N., & Shawe-Taylor, J. (2000). Kernel-Induced Feature Spaces. In *Support Vector Machines*. doi: 10.1017/cbo9780511801389.005

[11] Scikit-learn. (n.d.) *RBF SVM parameters*.

https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html [Accessed March 28th 2023]

[12] Cristianini, N., Shawe-Taylor, J. (2005). *Support Vector Machines*. In Cambridge University Press eBooks (pp. 93–124). doi: 10.1017/CBO9780511801389.008

[13] Tharwat, A. Parameter investigation of support vector machine classifier with kernel functions. *Knowl Inf Syst* **61**, 1269–1302 (2019). doi: 10.1007/s10115-019-01335-4

[14] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. doi: 10.1016/j.patrec.2005.10.010

[15] Feng, Y., Shen, X., Chen, H., & Zhang, X. (2016). A weighted-ROC graph based metric for image segmentation evaluation. *Signal Processing*, *119*, 43-55. doi: 10.1016/j.sigpro.2015.07.010

[16] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. doi: 10.1016/j.patrec.2005.10.010

[17] Kerschen, G., Peeters, M., Golinval, J., & Vakakis, A. (2009). Nonlinear normal modes, Part I: A useful framework for the structural dynamicist. *Mechanical Systems and Signal Processing*, *23*(1), 170-194. https://doi.org/10.1016/j.ymssp.2008.04.002

[18] Shaw, A. D., Hill, T. R., Neild, S. A., & Friswell, M. I. (2016). Periodic responses of a structure with 3:1 internal resonance. *Mechanical Systems and Signal Processing*, 81, 19–34. doi: 10.1016/j.ymssp.2016.03.008

[19] Iranmehr, A., Masnadi-Shirazi, H., & Vasconcelos, N. (2019). Cost-sensitive support vector machines. *Neurocomputing*, *343*, 50-64. doi: 10.1016/j.neucom.2018.11.099

[20] Wainer, J., Fonseca, P. How to tune the RBF SVM hyperparameters? An empirical evaluation of 18 search algorithms. *Artif Intell Rev* **54**, 4771–4797 (2021). doi: 10.1007/s10462-021-10011-5

[21] Omar, L., & Ivrissimtzis, I. (2019). Using theoretical ROC curves for analysing machine learning binary classifiers. *Pattern Recognition Letters*, *128*, 447-451. doi: 10.1016/j.patrec.2019.10.004

[22] University of Virginia Library Research Data Services. (n.d.). *ROC Curves and AUC for Models Used for Binary Classification*

https://data.library.virginia.edu/roc-curves-and-auc-for-models-used-for-binary-classification/#:~:text=ROC%20curves%20are%20graphs%20that%20plot%20a%20model%E2%80%99s,radar%20blips%20were%20genuine%20signals%E2%80%94e.g.%2C%20fighter%20planes%E2%80%94or%20noise.%29 [Accessed June 7th 2023]

[23] Song, M., Renson, L., Moaveni, B., & Kerschen, G. (2022). Bayesian model updating and class selection of a wing-engine structure with nonlinear connections using nonlinear normal modes. *Mechanical Systems and Signal Processing*, *165*, 108337. doi: 10.1016/j.ymssp.2021.108337

AI-Assisted Writing Declaration

I used Grammarly to refine the writing style of this report.

I used ChatGPT to help me translate some ideas in Results and Discussion from French to English

Appendices

Appendix 1: Main code

```
def preprocessing(scalogram, frequencies, f0, fend):
    import numpy as np
    from scipy.interpolate import interp1d
    Npoint = len(scalogram[0])
    exc_finst = np.linspace(f0, fend, Npoint) # instantaneous excitation frequency

    fnorm = np.linspace(0, 6, 120) # Vector of frequency ratios
    wt_amp_scaled = np.zeros((len(fnorm),Npoint))
    for i in range(Npoint):
      fscaled = (frequencies/exc_finst[i]).reshape(len(scalogram))
      scaling = interp1d(fscaled, scalogram[:,i], kind = 'linear',
              bounds_error = False, fill_value = np.percentile(scalogram[:,i], 25))
      wt_amp_scaled[:,i] = scaling(fnorm)

    wt_amp_scaled2 = np.zeros(np.shape(wt_amp_scaled))
    for i in range(Npoint):
      norm = np.sum(wt_amp_scaled[17:28,i])/5
      wt_amp_scaled2[:,i] = wt_amp_scaled[:,i] / norm
    return wt_amp_scaled2
```

```
# returns a dictionary, the keys correspond to the interaction and the values are
the interaction score for each interaction.
with open(r"balanced-svm-thr2", 'rb') as file:
    default = pickle.load(file)

def interaction(scalogram, model = default, preprocessed = True,
       frequencies = None, f0 = None, fend = None):
    if not preprocessed:
        scalogram = preprocessing(scalogram, frequencies, f0, fend)
    n = np.shape(scalogram)[1]
    proba = model.predict_proba(scalogram.T)
    score = {}
    for i in range(2,6):
          score[i] = np.sum(proba[:,i-1])/n
    return score
```

```
# Returns the relative amplitude for the i-th harmonic (relative because scaled by
the amplitude of the fundamental).
def norm(scalogram, i, preprocessed = True, frequencies = None, f0 = None,
      fend = None):
  import numpy as np
  if not preprocessed:
    scaledwt = preprocessing(scalogram, frequencies, f0, fend)
  else:
    scaledwt = scalogram
  l = []
  Npoint = len(scaledwt[0])
  for j in range(Npoint):
    norm1 = np.sum(scaledwt[15:26,j])
    normi = np.sum(scaledwt[20*i-5:20*i+6,j])
    l.append(normi/norm1)
  l = np.array(l)
  return l
```

```
# Returns a dictionary, the keys correspond to the interaction and the values are
the interaction score for each interaction. Without the use of ML
def interaction2(scalogram, preprocessed = True, threshold = 2, frequencies = None,
      f0 = None, fend = None):
  if not preprocessed:
    scalogram = preprocessing(scalogram, frequencies, f0, fend)
  n = np.shape(scalogram)[1]
  score = {}
  for i in range(2,6):
    l = norm(scalogram, i, scaled = True, frequencies = None, f0 = None,
      fend = None) > threshold
    score[i] = np.sum(l)/n
  return score
```

Appendix 2: Numerical values of the dataset.

Mode n	$\Phi_{n,1}$ (m)	$\Phi_{n,2}$ (m)	$\Phi_{n,3}$ (m)	$\Phi_{n,L}$ (m)
1	0.125	1.35	5.13	5.34
2	−0.575	−3.86	3.80	4.67
3	1.330	3.21	2.86	4.41

Numerical values of system where $\Phi_{n,i}$ is the mode shape of the *n*th mode at $x=x_i$, and $\Phi_{n,L}$ gives the *n*th mode shape at $x=L$.

Appendix 3: performance per class for C and γ

Class 0	**C**	0.1	1	10	100
γ					
0.1		precision = 100% specificity = 100% sensitivity = 93.80%	precision = 100% specificity = 100% sensitivity = 97.59%	precision = 100% specificity = 100% sensitivity = 98.61%	precision = 99.93% specificity = 99.98% sensitivity = 99.42%
1		precision = 100% specificity = 100% sensitivity = 94.46%	precision = 99.93% specificity = 99.98% sensitivity = 97.34%	precision = 100% specificity = 100% sensitivity = 98.54%	precision = 99.96% specificity = 100% sensitivity = 98.83%
10		precision = 100% specificity = 100% sensitivity = 88.44%	precision = 100% specificity = 100% sensitivity = 95.84%	precision = 99.92% specificity = 99.98% sensitivity = 96.14%	precision = 99.89% specificity = 99.97% sensitivity =96.35%
100		precision = 100% specificity = 100% sensitivity = 40.17%	precision = 100% specificity = 100% sensitivity = 85.93%	precision = 99.92% specificity = 99.98% sensitivity = 88.44%	precision = 100% specificity = 100% sensitivity = 88.30%

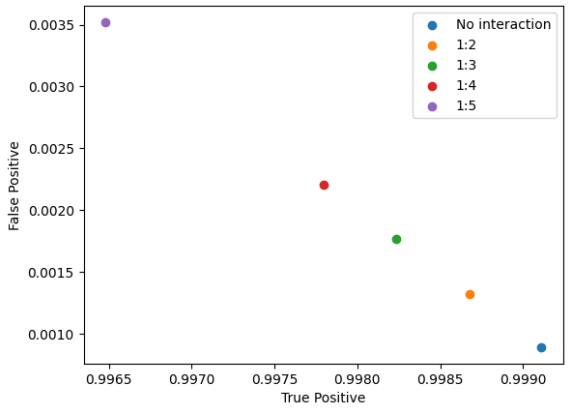
Class 2	C	0.1	1	10	100
γ					
0.1		precision = 99.17% specificity = 99.79% sensitivity = 100%	precision = 99.82% specificity = 99.95% sensitivity = 100%	precision = 99.96% specificity = 99.99% sensitivity = 100%	precision = 99.93% specificity = 99.98% sensitivity = 100%
1		precision = 99.71% specificity = 99.93% sensitivity = 100%	precision = 99.82% specificity = 99.95% sensitivity = 100%	precision = 99.96% specificity = 99.99% sensitivity = 100%	precision = 99.96% specificity = 99.99% sensitivity = 100%
10		precision = 99.96% specificity = 99.99% sensitivity = 98.10%	precision = 100% specificity = 100% sensitivity = 100%	precision = 99.89% specificity = 99.97% sensitivity = 100%	precision = 100% specificity = 100% sensitivity = 100%
100		precision = 100% specificity = 100% sensitivity = 74.81%	precision = 100% specificity = 100% sensitivity = 99.93%	precision = 99.96% specificity = 99.99% sensitivity = 100%	precision = 99.93% specificity = 99.98% sensitivity = 99.93%

Class 3	C	0.1	1	10	100
γ					
0.1		precision = 98.67% specificity = 99.66% sensitivity = 100%	precision = 99.53% specificity = 99.88% sensitivity = 100%	precision = 99.49% specificity = 99.87% sensitivity = 100%	precision = 99.71% specificity = 99.27% sensitivity = 99.93%
1		precision = 99.71% specificity = 99.93% sensitivity = 100%	precision = 99.20% specificity = 99.80% sensitivity = 99.82%	precision = 99.71% specificity = 99.93% sensitivity = 99.93%	precision = 99.82% specificity = 99.95% sensitivity = 99.93%
10		precision = 59.63% specificity = 83.08% sensitivity = 100%	precision = 99.92% specificity = 99.98% sensitivity = 95.77%	precision = 99.88% specificity = 99.97% sensitivity = 94.35%	precision = 99.92% specificity = 99.98% sensitivity = 95.81%
100		precision = 37.73% specificity = 58.74% sensitivity = 100%	precision = 99.91% specificity = 99.98% sensitivity = 78.27%	precision = 99.86% specificity = 99.97% sensitivity = 77.73%	precision = 100% specificity = 100% sensitivity = 79.88%

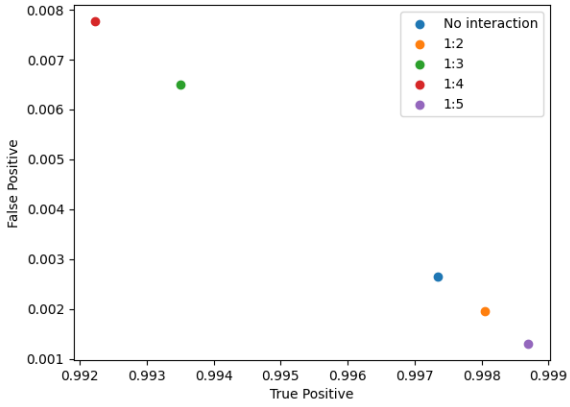
Class 4	C	0.1	1	10	100
γ					
0.1		precision = 97.48% specificity = 99.35% sensitivity = 100%	precision = 98.74% specificity = 99.68% sensitivity = 100%	precision = 99.53% specificity = 99.88% sensitivity = 100%	precision = 99.82% specificity = 99.95% sensitivity = 100%
1		precision = 98.79% specificity = 99.70% sensitivity = 98.29%	precision = 99.49% specificity = 99.87% sensitivity = 100	precision = 99.64% specificity = 99.91% sensitivity = 100%	precision = 99.75% specificity = 99.94% sensitivity = 100%
10		precision = 99.58% specificity = 99.90% sensitivity = 96.10%	precision = 99.71% specificity = 99.93% sensitivity = 99.45%	precision = 997.81% specificity = 99.95% sensitivity = 99.60%	precision = 99.89% specificity = 99.97% sensitivity = 99.78%
100		precision = 99.92% specificity = 99.98% sensitivity = 90.52%	precision = 99.93% specificity = 99.98% sensitivity = 97.89%	precision = 99.93% specificity = 99.98% sensitivity = 97.99%	precision = 99.96% specificity = 99.99% sensitivity = 98.07%

Class 5	C	0.1	1	10	100
γ					
0.1		precision = 98.60% specificity = 99.64% sensitivity = 100%	precision = 99.53% specificity = 99.88% sensitivity = 100%	precision = 99.64% specificity = 99.91% sensitivity = 100%	precision = 99.96% specificity = 99.99% sensitivity = 100%
1		precision = 99.35% specificity = 99.85% sensitivity = 94.02%	precision = 98.74% specificity = 99.68% sensitivity = 100%	precision = 99.17% specificity = 99.79% sensitivity = 100%	precision = 99.28% specificity = 99.82% sensitivity = 100%
10		precision = 99.19% specificity = 99.90% sensitivity = 48.82%	precision = 92.11% specificity = 97.86% sensitivity = 100%	precision = 91.40% specificity = 97.65% sensitivity = 100%	precision = 92.76% specificity = 98.05% sensitivity = 99.96%
100		precision = 99.50% specificity = 99.96% sensitivity = 29.24%	precision = 72.55% specificity = 90.54% sensitivity = 100%	precision = 73.78% specificity = 91.11% sensitivity = 100%	precision = 74.78% specificity = 91.56% sensitivity = 100%

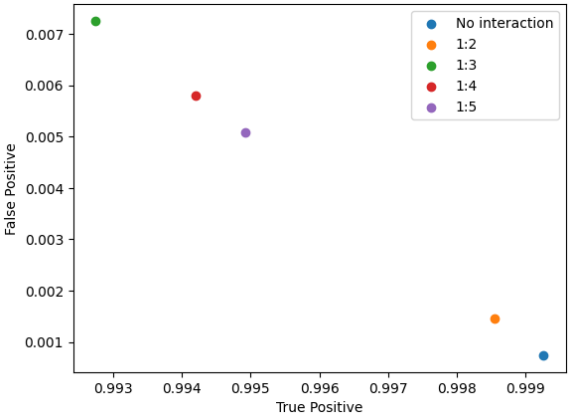
Appendix 4: ROC graph threshold on probabilities



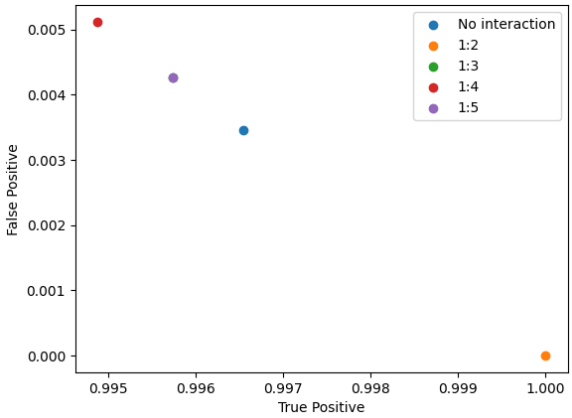
Threshold = 1.5



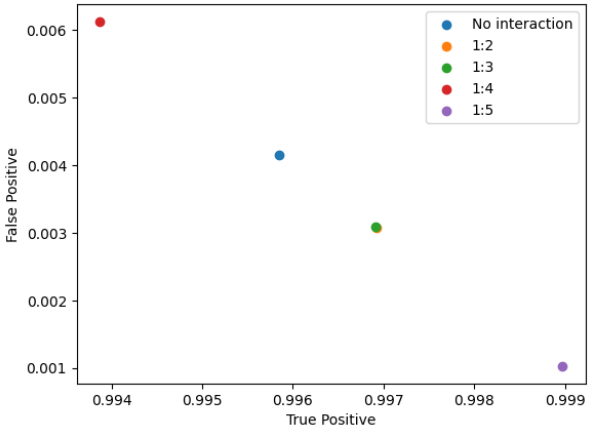
Threshold = 1.75



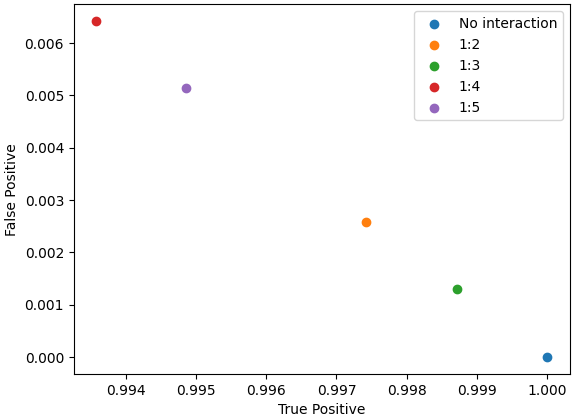
Threshold = 2



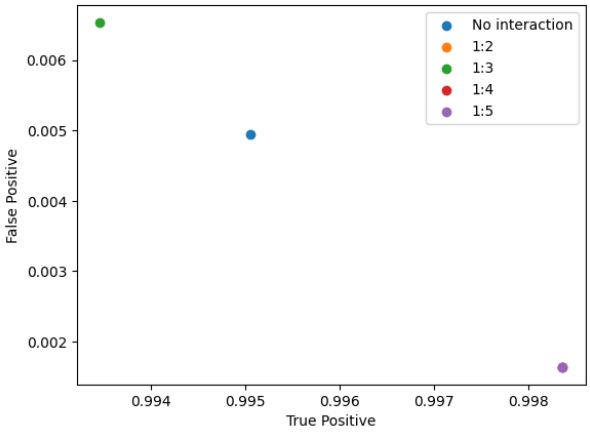
Threshold = 2.25



Threshold = 2.5

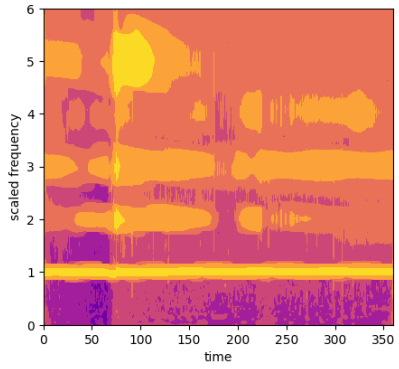


Threshold = 2.75

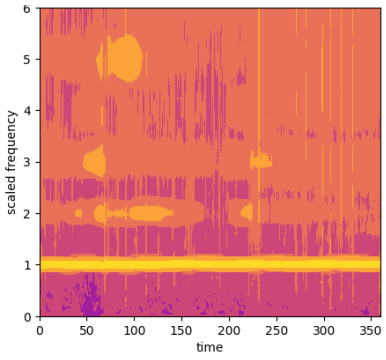

No interaction
1:2
1:3
1:4
1:5
0.006
0.005
0.004
0.003
0.002
False Positive
0.994
0.995
0.996
0.997
0.998
True Positive

Threshold = 3

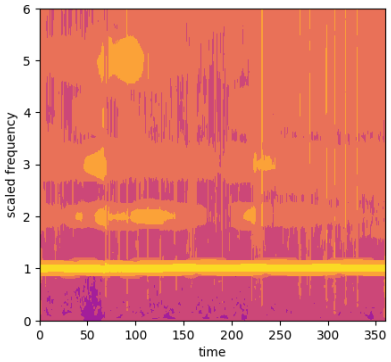
Appendix 5: Pre-processed Scalogram of Wing Engine



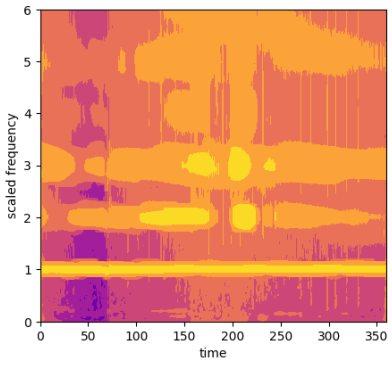
Sensor 1



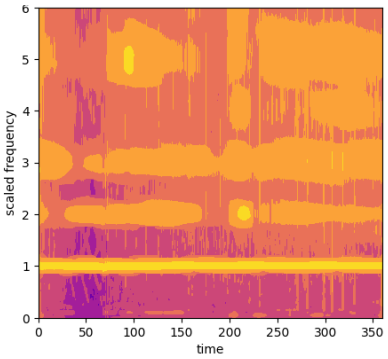
Sensor 2



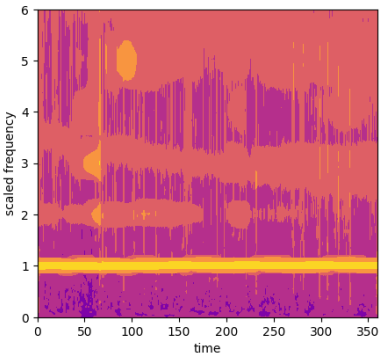
Sensor 3



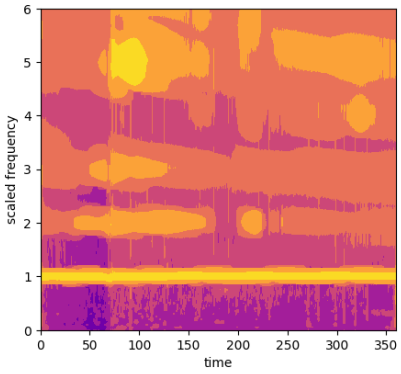
Sensor 4



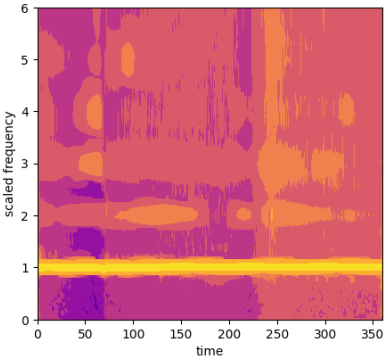
Sensor 5



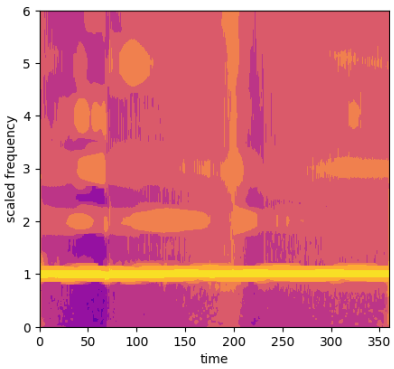
Sensor 6



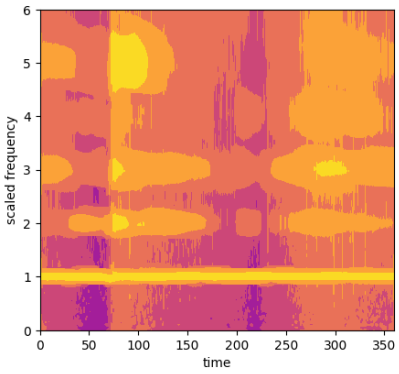
Sensor 7



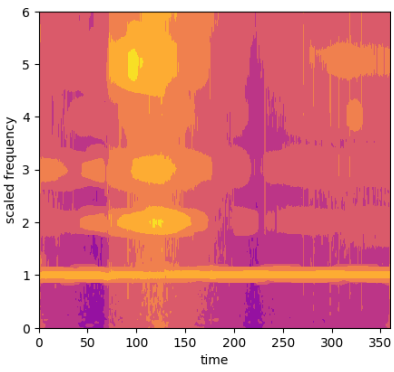
Sensor 8



Sensor 9

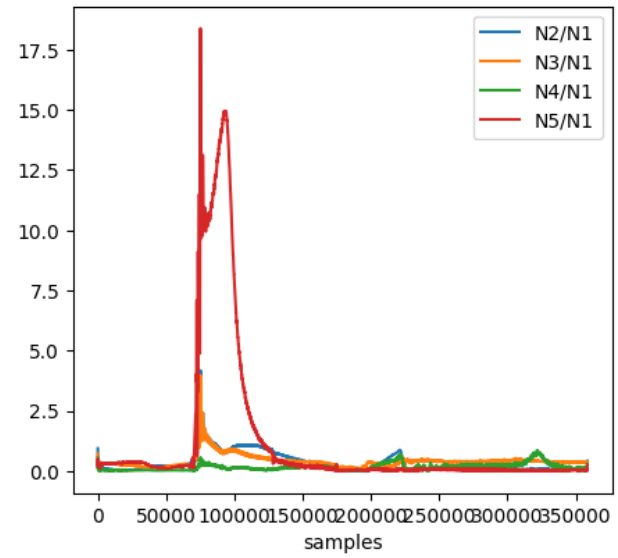


Sensor 10

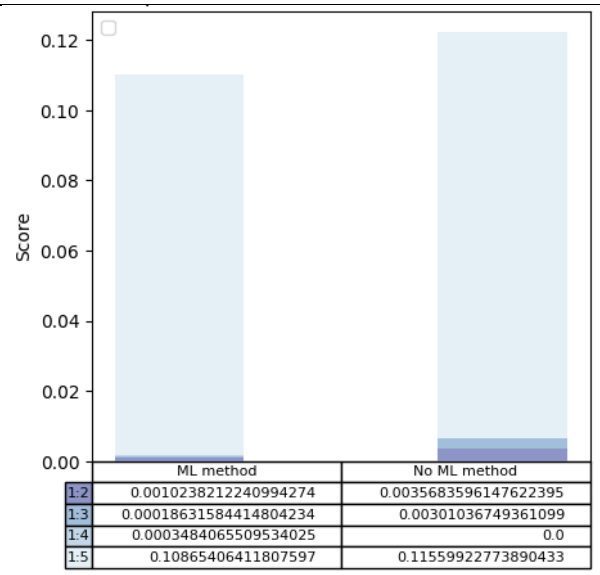


Sensor 11

Appendix 6: $\frac{n_i}{n_1}$ and algorithm results of Wing Engine



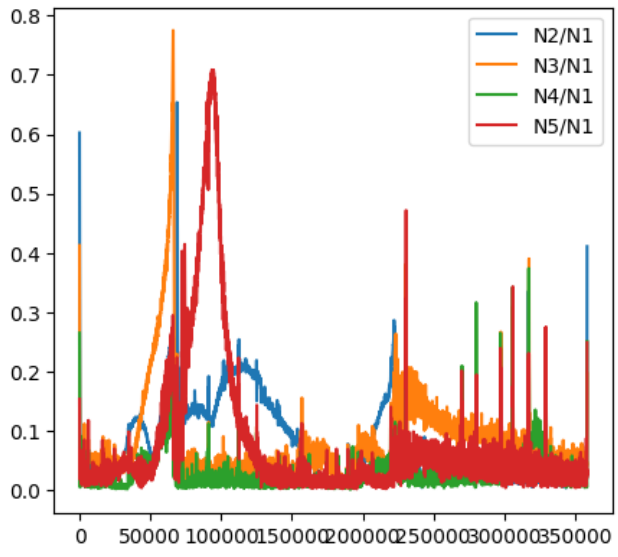
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$



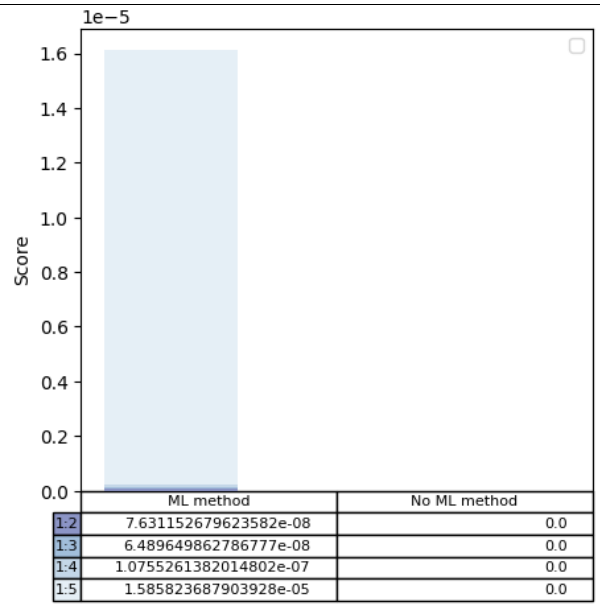
	ML method	No ML method
1:2	0.0010238212240994274	0.0035683596147622395
1:3	0.0001863158441480423 4	0.00301036749361099
1:4	0.0003484065509534025	0.0
1:5	0.10865406411807597	0.11559922773890433

Comparison between ML and No_ML methods

Sensor 1



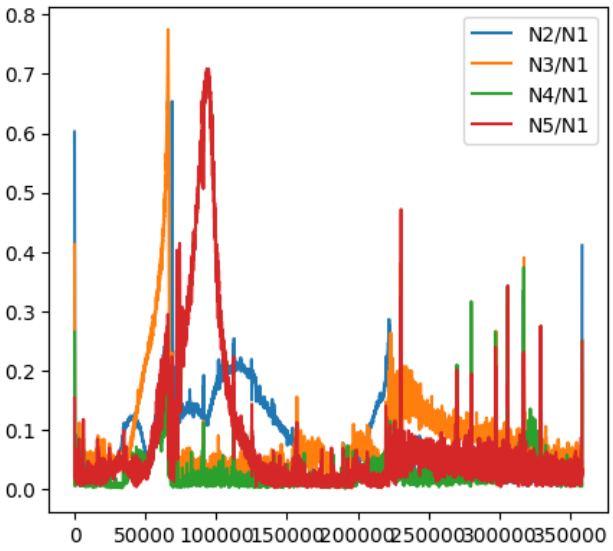
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$



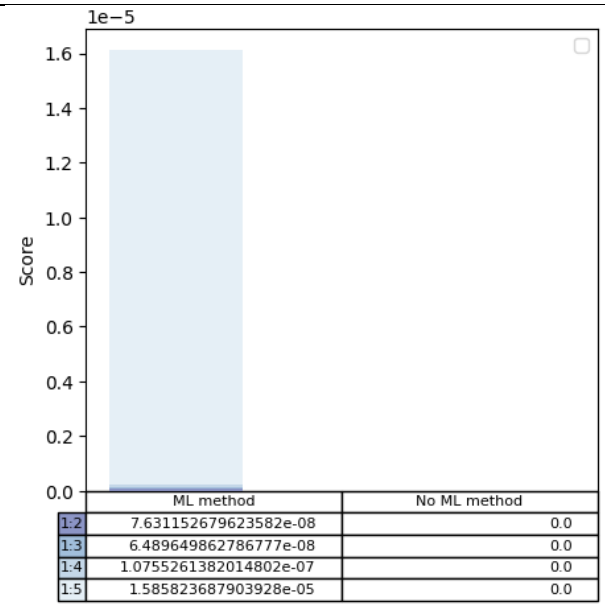
	ML method	No ML method
1:2	7.631152679623582e-08	0.0
1:3	6.489649862786777e-08	0.0
1:4	1.0755261382014802e-07	0.0
1:5	1.585823687903928e-05	0.0

Comparison between ML and No_ML methods

Sensor 2



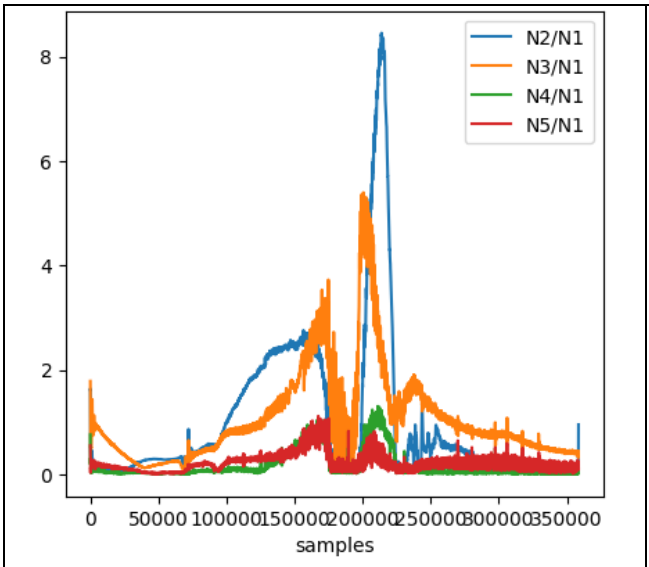
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$



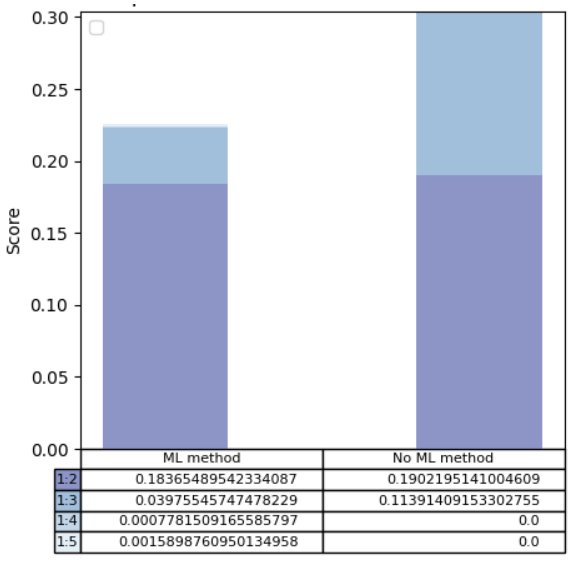
	ML method	No ML method
1-2	7.631152679623582e-08	0.0
1-3	6.489649862786777e-08	0.0
1-4	1.0755261382014802e-07	0.0
1-5	1.585823687903928e-05	0.0

Comparison between ML and No_ML methods

Sensor 3



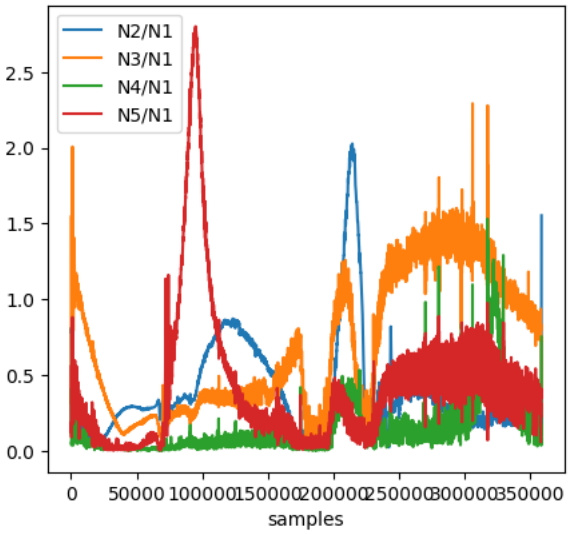
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$



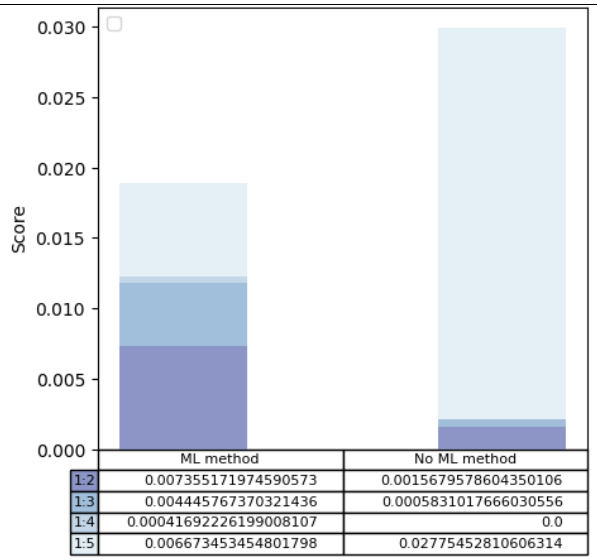
	ML method	No ML method
1-2	0.18365489542334087	0.1902195141004609
1-3	0.03975545747478229	0.11391409153302755
1-4	0.0007781509165585797	0.0
1-5	0.0015898760950134958	0.0

Comparison between ML and No_ML methods

Sensor 4

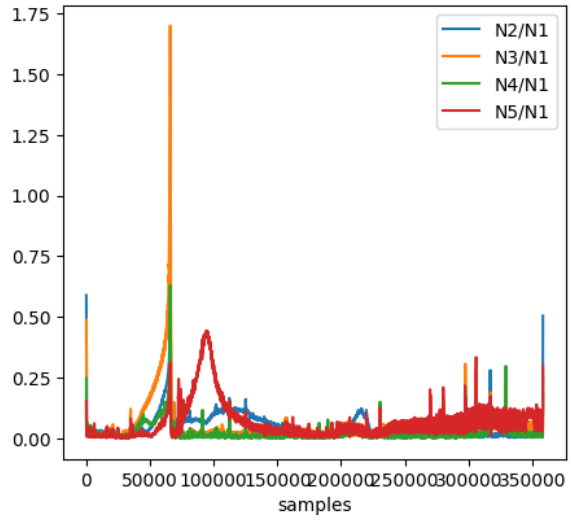


$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$

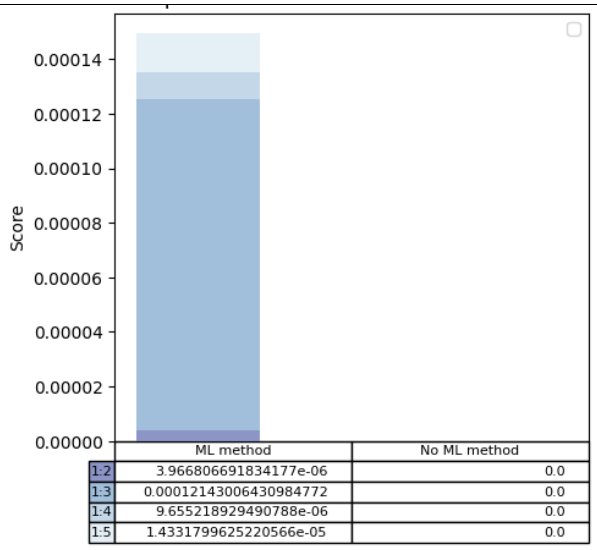


Comparison between ML and No_ML methods

Sensor 5

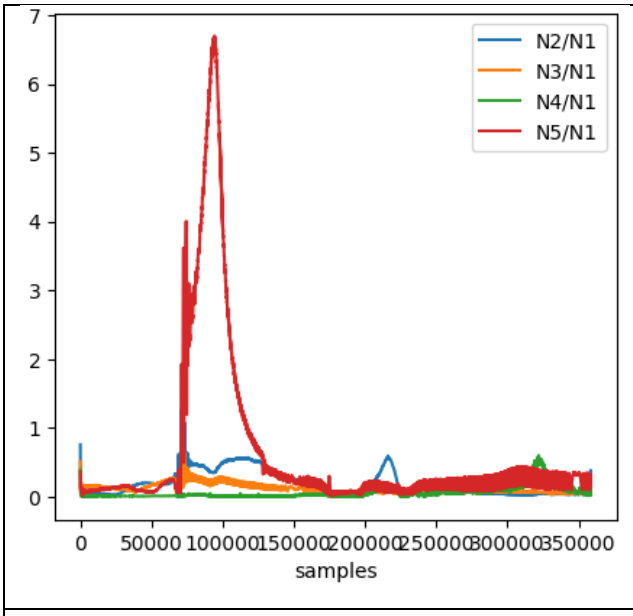
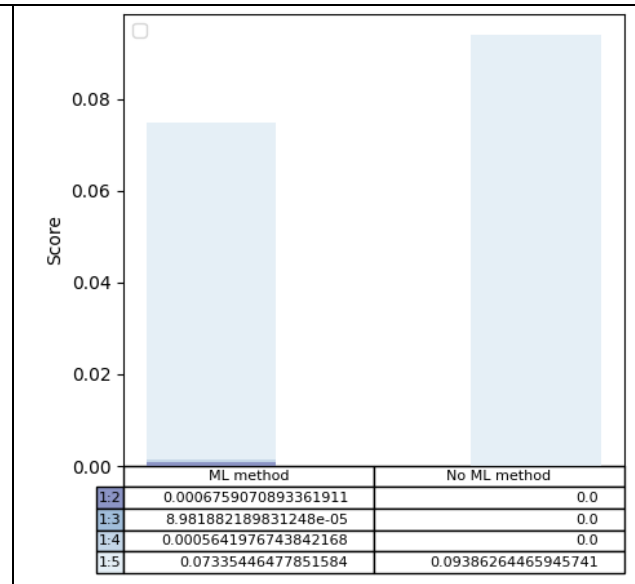


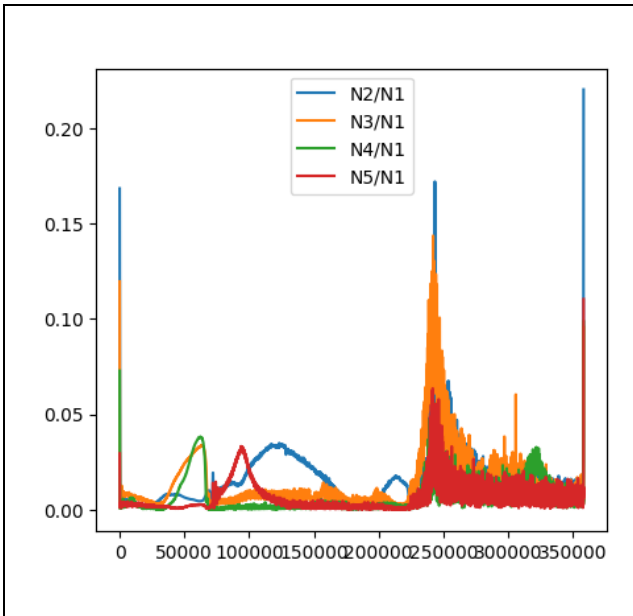
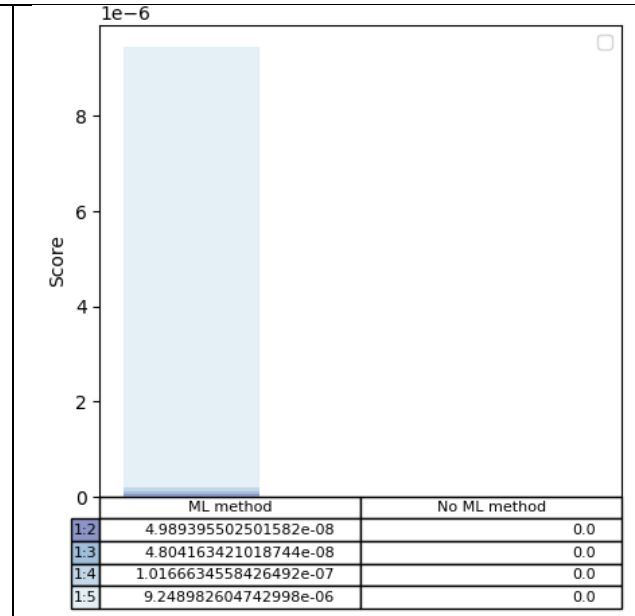
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$

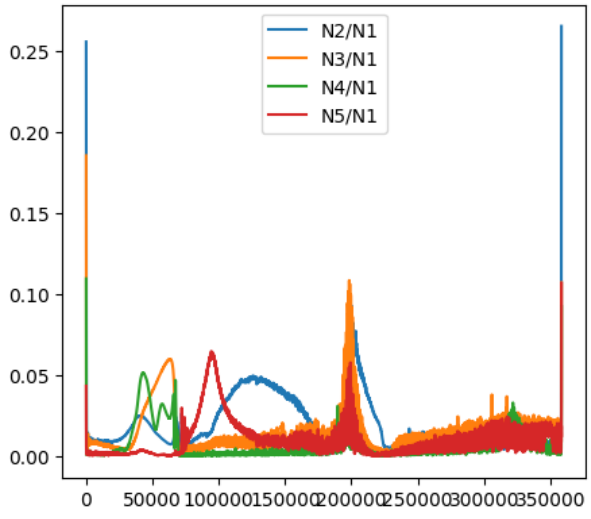
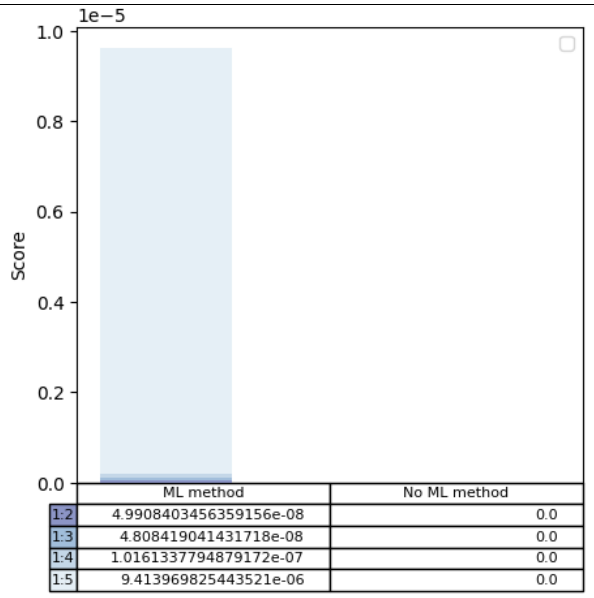


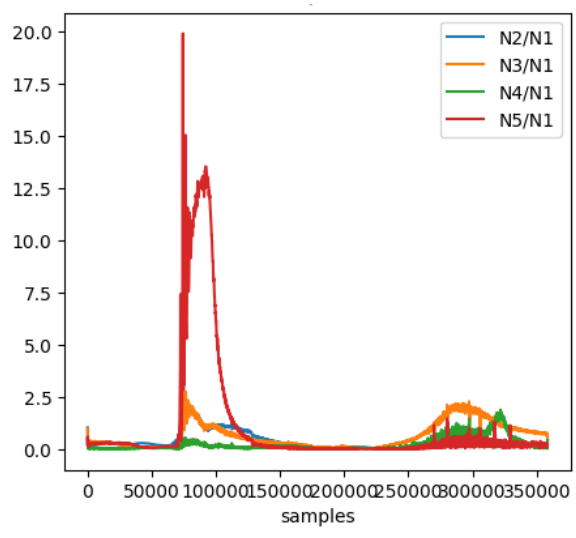
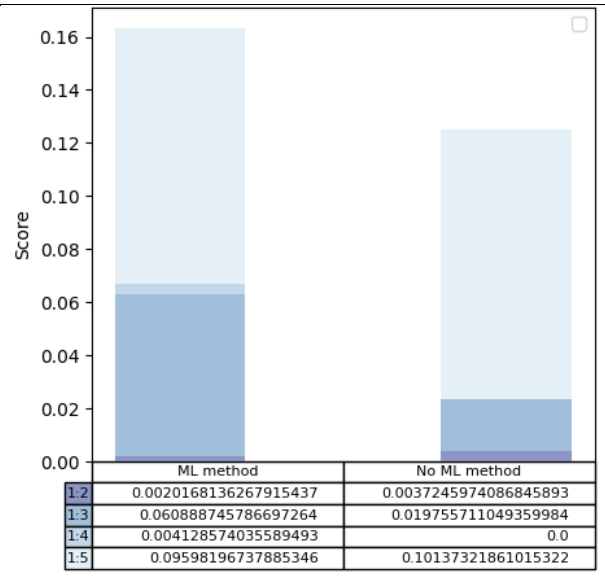
Comparison between ML and No_ML methods

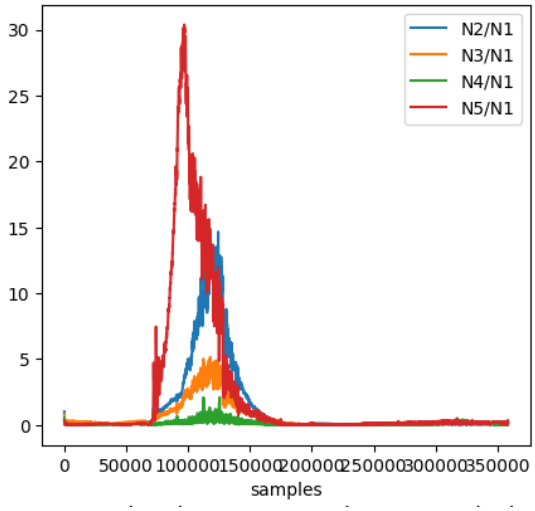
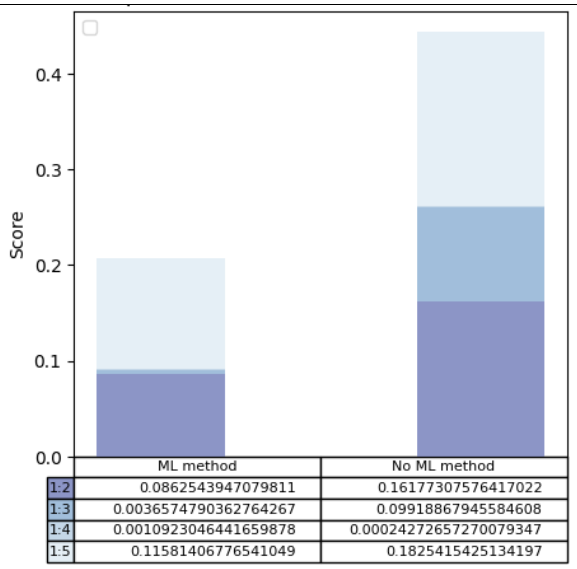
Sensor 6

	
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$	Comparison between ML and No_ML methods
Sensor 7	

	
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$	Comparison between ML and No_ML methods
Sensor 8	

	
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$	Comparison between ML and No_ML methods
Sensor 9	

	
$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$	Comparison between ML and No_ML methods
Sensor 10	

<table>
<tr><td> 
</td><td> 

	ML method	No ML method
1:2	0.0862543947079811	0.16177307576417022
1:3	0.0036574790362764267	0.09918867945584608
1:4	0.0010923046441659878	0.00024272657270079347
1:5	0.11581406776541049	0.1825415425134197

</td></tr>
<tr><td>$\frac{n_i}{n_1}$ $(i \in \{2,3,4,5\})$</td><td>Comparison between ML and No_ML methods</td></tr>
<tr><td colspan="2">Sensor 11</td></tr>
</table>