Machine learning for noisy multivariate time series classification: a comparison and practical evaluation

Aldomar Pietro Santana Silva[1], Lucas Riera Abbade[1], Rodrigo da Silva Cunha[1], Tomaz Maia Suller[1], Eric Gomes[1], Edson Satoshi Gomi[1], Anna Helena Reali Costa[1]

[1]Escola Politécnica – Universidade de São Paulo (USP)

{pietrosantana,lucasabbade,rodrigo.cunha}@usp.br

{tomaz.suller,ericog,gomi,anna.reali}@usp.br

Abstract. *Multivariate Time Series Classification (MTSC) is a complex problem that has seen great advances in recent years from the application of state-of-the-art machine learning techniques. However, there is still a need for a thorough evaluation of the effect of signal noise in the classification performance of MTSC techniques. To this end, in this paper, we evaluate three current and effective MTSC classifiers – DDTW, ROCKET and InceptionTime – and propose their use in a real-world classification problem: the detection of mooring line failure in offshore platforms. We show that all of them feature state-of-the-art accuracy, with ROCKET presenting very good results, and InceptionTime being marginally more accurate and resilient to noise.*

1. Introduction

Time series are sequences of values ordered in time. They are present in various natural and man-made phenomena such as weather, sound wave, and financial data [Längkvist et al. 2014]. The amount of time series continues to grow due to the increase in data generation and collection. A multivariate time series consists of multiple time-dependent variables, each of which correlates with its past values and with those of the other variables. Univariate time series methods lack the ability to capture the relationship between variables and therefore cannot be extrapolated directly to multivariate problems.

Classification of time series relies on assigning a class to a previously unseen time series, exploring the temporal component of the data, unlike traditional classification. Multivariate Time Series Classification (MTSC) in particular is relevant to several areas, with usually non-linear relationships between variables and their past values warranting the use of machine learning techniques in its solution.

Since the release of the UCR Time Series Classification Repository[1] there has been rapid growth in the number of publications proposing and evaluating univariate time series classification (UTSC) algorithms. [Bagnall et al. 2017] identified three UTSC methods as the most accurate among those present in the UCR repository: (i) BOSS [Schäfer 2015], a dictionary-based classifier; (ii) HIVE-COTE [Lines et al. 2016], a large ensemble of other classifiers, with BOSS included; and (iii) Shapelet Transform [Hills et al. 2013], a classifier based on finding discriminatory subseries. All these classifiers were considered state-of-the-art methods, with HIVE-COTE being the most accurate.

[1]https://timeseriesclassification.com/

Despite its accuracy, the high computational complexity of HIVE-COTE makes it slow even for small data sets, and intractable for large ones [Dempster et al. 2020]. Thus, a wide variety of methods were developed aiming to be as accurate as the state-of-the-art, but faster and more scalable, such as ROCKET [Dempster et al. 2020] and InceptionTime [Ismail Fawaz et al. 2020].

Some of these methods can be used for both UTSC and MTSC. With the 2018 release of the UEA repository with 30 MTSC datasets, more attention was devoted to MTSC. [Ruiz et al. 2021] conducted a detailed review in order to assess the performance of the MTSC methods present in the UCR & UEA repository in 26 datasets. Their results were obtained from the average measures over thirty resamples of each dataset to present a single statistic for each ⟨dataset - classifier⟩ pair, which were compared to the benchmark Dynamic Time Warping (DTW) classifier [Shokoohi-Yekta et al. 2017, Kruskal and Liberman 1983]. They concluded that only four models were statistically significantly better than the benchmark DTW classifier, highlighting that DTW is a classical approach that is still hard to beat and, for the next evaluations, suggested new MTSC algorithms be compared to Dependent DTW (DDTW) and to ROCKET, the most accurate analyzed classifier.

However, it is known that real-world data contain noise – irrelevant, spurious or missing values – that significantly affects the results of MTSC algorithms [Gupta and Gupta 2019], leading to various studies, such as [Kalapanidas et al. 2003] on noise sensitivity of machine learning algorithms, and, [Schäfer 2015] specifically on time series classification. The latter noted noise had received surprisingly little attention until then, as it was assumed to be filtered out as part of manual data pre-processing and proposed the BOSS model to address this concern. It is also important to note that experiments in [Schäfer 2015] were conducted before the UEA repository was made available, creating the need for a reassessment of how the best MTSC models are affected by noisy data.

Motivated by these previous works, we compare three MTSC algorithms: DDTW [Shokoohi-Yekta et al. 2017] and ROCKET [Dempster et al. 2020], suggested by [Ruiz et al. 2021], and InceptionTime [Ismail Fawaz et al. 2020], one of the four models that surpasses the DTW benchmark in [Ruiz et al. 2021]. In addition, we evaluated the noise sensitivity of these models. To this end, we employ data from both the UCR & UEA epilepsy dataset with varying levels of mixed-in noise, and from a real-world problem: the detection of mooring line failure in offshore oil extraction platforms. This problem severely impacts crew safety and the occurrence of environmental disasters such as oil spills in the oil and gas sector. [Ma et al. 2013].

In summary, we make the following main contributions in this paper: **(i)** we verify the reproducibility of the experiments conducted by [Ruiz et al. 2021] with the best MTSC algorithms in the UCR & UEA repository; **(ii)** we analyze the sensitivity of these algorithms to increasing noise in the benchmark data, and **(iii)** we address a real-world problem using these algorithms, evaluating and comparing their performances on noisy data.

This paper is organized as follows. We review the selected algorithms in Section 2, describe the epilepsy data and experimental design and present results in Section 3.

Section 4 describes mooring line failure detection and presents the performance of the MTSC algorithms. We close this paper in Section 5, summarizing our main findings and discussing future directions.

2. Multivariate Time Series Classification

A univariate time series is defined as a sequence of n observations over time of a given measured quantity, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$. A multivariate time series may be represented by an $n \times m$ matrix of m variables, each with n observations, i.e. $\mathbf{X} = (\mathbf{x}_1^\top, \ldots, \mathbf{x}_m^\top)$, $\mathbf{x}_j = (x_{1,j}, \ldots, x_{n,j})$, $1 \leq j \leq m$. The set of observations is given by X, with $\mathbf{X} \in X$. In a classification context, each time series is associated with a label y which is a value of a discrete class variable Y. An instance is defined as a pair $\langle \mathbf{X}, y \rangle$, and k pairs form a dataset $S = \{\langle \mathbf{X_1}, y_1 \rangle, \ldots, \langle \mathbf{X_k}, y_k \rangle\}$.

A classifier can be either a function f from sequences to labels ($f : X \rightarrow Y$) or a function g which maps the space of possible inputs to a probability distribution over the class variable values ($g : X \times Y \rightarrow [0, 1]$). Finding the function that best approximates f or g using the dataset S is referred to as training the classifier. Next, we briefly describe the three classifiers chosen to carry out the experiments and comparative analysis.

2.1. Dependent Dynamic Time Warping (DDTW)

Dynamic Time Warping (DTW) has emerged as the distance measure of choice for many time series data mining applications. In the multi-dimensional case, DTW was generalized in one of two ways: dependent or independent warping. Here, we use the dependent warping approach, as suggested in [Ruiz et al. 2021]. This approach employs DTW as a distance measure for a 1-Nearest Neighbour classifier, compensating for a possible confounding offset by allowing some series realignment [Ruiz et al. 2021]. The DTW distance subsumes the Euclidean distance as a special case and has been shown to be significantly more accurate.

For DTW alignment of a univariate time series instance, an $n \times n$ matrix is constructed, whose element (i, j) contains the squared Euclidean distance $d(q_i, c_j) = (q_i - c_j)^2$ between two points q_i and c_j of the univariate time series $\mathbf{q}$ and $\mathbf{c}$. Then, a warping path P, a contiguous set of matrix elements defining a map between Q and C, is calculated, usually subject to several constraints, such as: (i) the steps in the warping path are restricted to adjacent cells; (ii) the path must start and finish in diagonally opposite corner cells of the matrix and (iii) the points in the warping path must be monotonically spaced in time. Also, it is common to constrain the warping path by limiting how far it may stay from the diagonal [Ding et al. 2008]. The warping path that minimizes the warping cost is chosen. The warping cost D consists of the sum of the points of the warping path, and a recurrence function can be used to find it [Kruskal and Liberman 1983]:

$$D(i, j) = d(q_i, c_j) + min\{D(i-1, j-1), D(i-1, j), D(i, j-1)\}. \quad (1)$$

Here we use Dependent Dynamic Time Warping (DDTW). The dependent approach is an adaptation of the DTW for multivariate time series where $d(q_i, c_j)$ is redefined and assumes the value of the cumulative squared Euclidean distances of M data points:

$$d(q_i, c_j) = \sum_{l=1}^{M} (q_{i,l} - c_{j,l})^2, \quad (2)$$

where i and j are data points of the lth dimension of the multivariate time series. Before computing the DTW distance, a z-normalization of each dimension of the time series is necessary to make the distance measure scale and offset. Finally, the computed distances are used in a 1-Nearest-Neighbour classifier.

2.2. Random Convolutional Kernel Transform

The Random Convolutional Kernel Transform (ROCKET) [Dempster et al. 2020] was chosen for its remarkable accuracy and speed on a wide variety of datasets from the UCR & UEA repository [Ruiz et al. 2021].

ROCKET uses a large number of random convolutional kernels of different lengths, weights, biases, dilation and padding. Every generated kernel is applied to each instance. The maximum value from the resulting feature maps (`max`) and the proportion of positive values (`ppv`) are returned. The resulting features are used to train a linear classifier, such as Ridge regression [McDonald 2009]. A convolution operation is applied between the kernels and the series. The resultant values of this operation are `max` and `ppv`, which is a summary of the proportion of the series correlated to the kernel and a measure that significantly improves classification accuracy [Ruiz et al. 2021]. When 10,000 kernels are randomly generated (the default value), 20,000 attribute instances are generated for each series. The resulting attribute dataset is used to train the Ridge regression classifier (or another linear classifier).

The only ROCKET hyperparameter is the number of kernels K, which establishes a trade-off between classification accuracy and computer processing time: the greater K, the greater the classification accuracy, but the longer the processing time. As for multivariate time series, kernels are generated with randomly assigned dimensions, with weight being generated for each channel. The `max` value and `ppv` are now calculated within more dimensions, but it still produces a 20,000 attributes instance.

2.3. InceptionTime

The InceptionTime model [Ismail Fawaz et al. 2020] was chosen because it is a good representative of Deep Learning models for MTSC since it is based on Convolutional Neural Networks (CNNs), which have been shown to have good performance and low training computational cost in the UCR & UEA repository [Fawaz et al. 2019]. CNNs possess several characteristics which make them attractive for time series classification. In particular, the same filter (or kernel) parameters are used across the entire time series, allowing the model to abstract long-term patterns, and have fewer trainable parameters, this way facilitating training and curbing overfitting [Goodfellow et al. 2016].

InceptionTime consists of several identical modules, denoted inception modules, concatenated sequentially. Each inception module has a number of convolutional filters, each with different parameters (size, stride, padding, etc.) whose outputs are aggregated in order to serve the next module. Skip connections, also called residual connections, are used between some modules, directly connecting the output of the $(i-1)$th module to the input of the $(i+1)$th, reducing issues associated with gradient vanishing present in very deep neural networks.

The input layers are structured so that the model can process either univariate or multivariate time series; its output is the probability of the input data belonging to each of

the Y predefined output classes, calculated using a Global Average Pooling (GAP) layer followed by a final fully-connected layer with a softmax activation function.

Each inception module first applies a single one-dimensional convolutional filter with a width of 1, which aims to aggregate multiple different time series – either from a multivariate input or from one of the previous inception modules – into a single time series. The result from this layer serves as input to the different convolutional filters applied. Each inception module contains a total of 97 convolutional filters – 32 for each of the widths 40, 20 and 10, and one for the direct output of the one-dimensional layer. The concatenated result is processed through a batch normalization layer and subsequently through a ReLU activation function, resulting in a multivariate time series which is used as input either to the next inception module or to the output calculation. The presence of a learning rate scheduler is also noteworthy, as it can make the model more resilient to sub-optimal learning rate configurations, particularly to exceedingly high learning rates.

3. Experimental Setup and Results

Experiments were performed on a single server with the following hardware: AMD Ryzen 7 3800X 8-Core Processor @ 4.3 GHz, Corsair MP600 M.2 NVMe SSD, 32 GB @ 3 GHz RAM, and an NVIDIA RTX 3070 24 GB GPU. The server software configuration comprises Windows 10 and Python 3.8.13. Python libraries `sktime`[2] – which implements ROCKET and DDTW – and `sktime-dl`[3] – which implements InceptionTime – were employed to facilitate the reproduction of results from [Ruiz et al. 2021]. NumPy and pandas were also used to handle data and calculate metrics.

We selected the epilepsy dataset to reproduce the results obtained in [Ruiz et al. 2021] for the three chosen classifiers. This dataset has four classes: walking, running, sawing, and seizure imitation, representing the activities performed by volunteers when using a 3D accelerometer, which provided the data. This dataset was selected for two reasons. First, all three models presented high precision in this dataset, which helps us in assessing performance degradation at different noise levels. Second, the epilepsy data is similar to the real-world problem data that was evaluated in Section 4. The data were obtained from the repository provided by [Ruiz et al. 2021]. Thus, the data is already divided into training and test sets, with 137 training and 137 test three-dimensional instances, with each dimension representing movement in one of the x, y or z axes.

The methodology used in [Kalapanidas et al. 2003] is followed for sensitivity analysis. Thus, white noise is added to each epilepsy dataset instance. White noise follows a normal distribution with zero mean and finite standard deviation σ, denoted noise level. Therefore, for each time stamp t of each instance of the dataset, noise s_t is sampled from $\mathcal{N}(0, \sigma)$ and new values $x'_{t,j} = x_{t,j} + s_t,\ j = 1, 2, 3$ are calculated. Our experiments employed $\sigma \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 1\}$. The random number generators employed to create noise had a fixed seed for reproducibility purposes. Consequently, 7 different datasets were created for the noise sensitivity experiment, one without added noise, and others with increasing noise levels. Figure 1 shows an example of how the noise level impacts a single time series.

[2]https://github.com/alan-turing-institute/sktime/.

[3]https://github.com/sktime/sktime-dl.

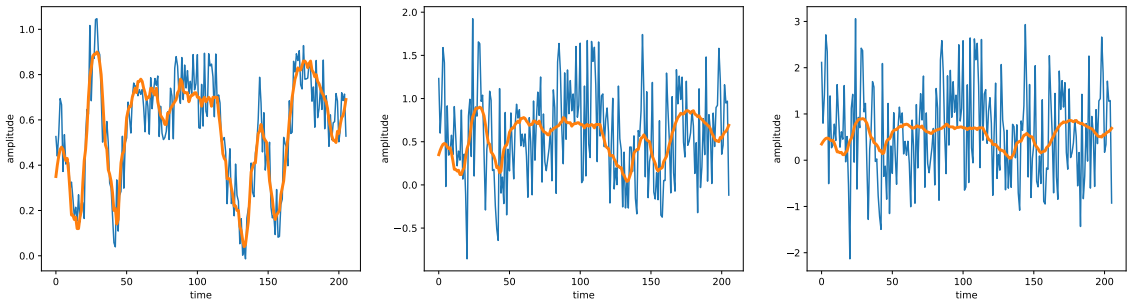

Figure 1. Example of one dimension from an instance of the epilepsy dataset (in orange) with different noise levels (from left to right, 0.1, 0.5 and 1).

The main metric evaluated was accuracy, $Accuracy = (TP + TN)/(TP + TN + FP + FN)$, where TP, TN, FP and FN are True Positives, True Negatives, False Positives and False Negatives, respectively. However, in some experiments, $Precision = TP/(TP + FP)$; $Recall = TP/(TP + FN)$; and F1-score were also used for comparative purposes, $\textit{F1-score} = (2 \times Precision \times Recall)/(Precision + Recall)$.

Each model was trained on the pre-defined training set with the default hyperparameters provided by the library implementation. An exception is the experiment with XGBoost on ROCKET's transformed dataset, in which training was done in the linear classifier XGBoost available on library `sklearn`. XGBoost's parameters optimizations were done using the library Optuna, associated with 5-fold cross-validation.

Each classifier was trained on 30 random resamples of the training set, after each of which it was evaluated using the test set. Accuracy, precision, recall, and F1-score were calculated for each iteration, the mean and standard deviation of which are presented in the results. In the noise evaluation experiment, each noisy dataset generated was trained and tested only once, in contrast to the 30 random resamples executed in the comparative experiment. This explains why the results in Table 1 are different from the results with $\sigma = 0$ shown in tables that contain the noise experiment. The observations with noise were then used for testing the models again and verifying the decay in accuracy as noise levels increased.

As for results, we show the values obtained with the epilepsy dataset, with and without noise, for each of the three selected classifiers: ROCKET, InceptionTime and DDTW. We also present the values obtained from the experiments of the XGBoost combined with ROCKET, replacing the default Ridge Classifier.

Initially, we show in Table 1 the results of experiments using the original dataset, without adding noise. The results achieved in these experiments demonstrate that DDTW, InceptionTime and ROCKET + Ridge show similar performances, as expected from previous results presented in [Ruiz et al. 2021]. While ROCKET + XGBoost shows the worst results on all metrics, ROCKET + Optimized XGBoost shows the highest values on the metrics in the Table 1.

In this context, ROCKET is often used combined with the Ridge classifier and our results may suggest that combining ROCKET with other linear classifiers can be a good alternative to solve problems. But we cannot forget the fact that the tiny improvement that

Table 1. Epilepsy metrics for each model over 30 resamples in percentage points. The best mean values are in bold.

Model	Accuracy	Precision	Recall	F1-Score
DDTW	90.36 ± 2.28	91.57 ± 1.89	90.36 ± 2.28	90.31 ± 2.27
ROCKET + Ridge	95.70 ± 1.45	95.81 ± 1.42	95.70 ± 1.45	95.69 ± 1.46
ROCKET + XGBoost	88.02 ± 3.72	88.56 ± 3.54	88.02 ± 3.72	87.94 ± 3.76
ROCKET + Opt. XGBoost	$\mathbf{97.15 \pm 1.46}$	$\mathbf{97.23 \pm 1.41}$	$\mathbf{97.15 \pm 1.46}$	$\mathbf{97.13 \pm 1.48}$
InceptionTime	96.52 ± 1.67	96.65 ± 1.59	96.52 ± 1.67	96.51 ± 1.67

ROCKET+Opt. XGBoost delivers over other models like ROCKET+ Ridge, for example, may not be worth the extra computational resources employed to optimize the models. In the end, despite the varied accuracy, all classifiers fit the problem well, which will help us in the next step, which is the noise sensitivity analysis.

Figures 2 and 3 show how sensitive each model is to different noise levels introduced in both training and test data. Elements in blue represent lower values of accuracy, and elements in red have higher values of accuracy. Figure 4 summarizes the sensitivity of models when noise is only inserted in the test data.

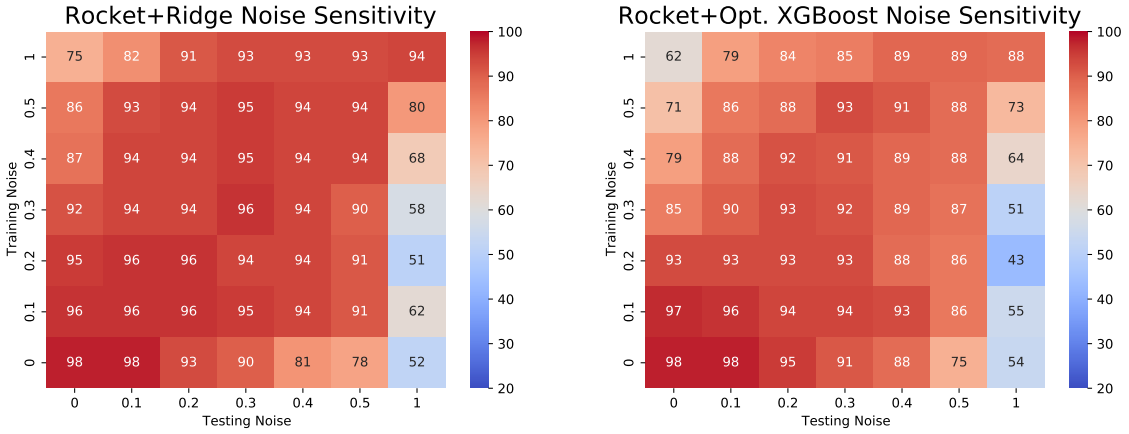

Figure 2. Accuracy of ROCKET with Ridge and with Optimized XGBoost in relation to noise sensitivity. Without noise in both training and testing, the classifiers have the best accuracy (represented by the darkest red). The numbers within each cell indicate accuracy.

Performing a qualitative analysis, we can see from Figure 4 that InceptionTime has the higher values at dealing with white noise in the test data, with significant accuracy degradation only for $\sigma > 0.5$, while the performance of other models continuously degrades with increasing noise levels. This can also be seen in Figures 2 and 3, showing that the highest amount of red color is in the InceptionTime classifier map.

To compare the models in this experiment, we generate a Critical Difference Diagram using Wilcoxon-Holm post-hoc analysis to detect pairwise significance. To generate the diagram, we considered each combination of training noise and test noise as a single dataset. For example, the combination of training without noise and testing with $\sigma = 0.1$ of noise represents one data set, while training without noise and testing with $\sigma = 0.2$ of noise represents another. So we ended up with 49 datasets to compare the models.

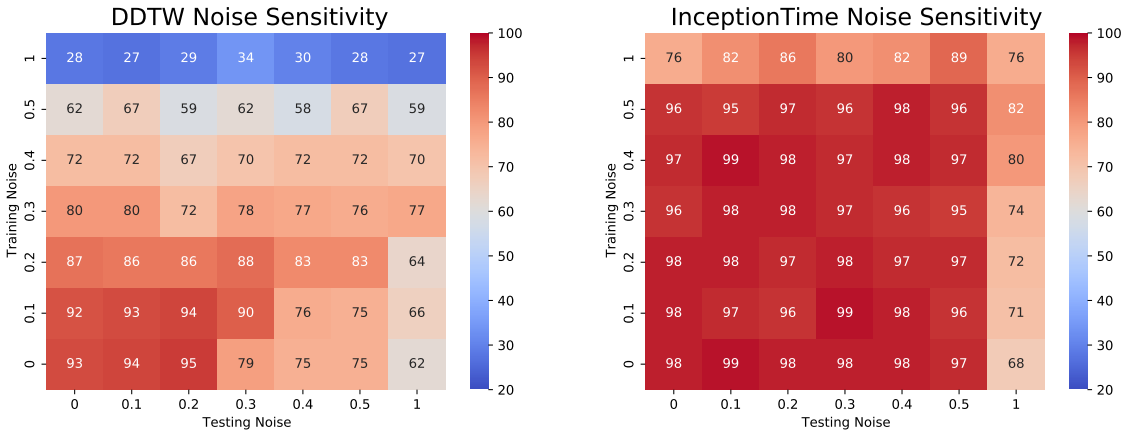

Figure 3. Accuracy of Dependent Dynamic Time Warping and InceptionTime in relation to noise sensitivity. Without noise in both training and testing, the classifiers have the best accuracy (represented by the darkest red). The numbers within each cell indicate accuracy.

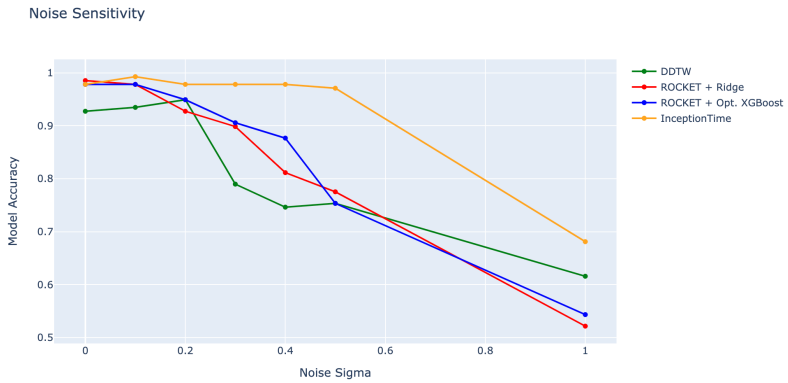

Figure 4. Classifiers' accuracy by noise standard deviation. This graph shows the sensitivity of models when noise is added to the test data (training without noise).

In fact, the results presented by the models are all statistically different from each other, with InceptionTime being the best model in this approach. InceptionTime is a CNN and its better performance can be explained by its ability to extract features due to the convolutions made in its input layers. After that, all features are filtered through the hidden layers of the network, further reducing the harmful effects of noise on classification. The diagram also shows that ROCKET+Ridge is the second-best classifier. This reinforces the possibility that classifiers like XGBoost associated with ROCKET may have less synergy with the transformer than the Ridge classifier, given its pretty solid performance. Finally, the DDTW had the worst performance, with accuracy values below 30, when all other models presented values of at least 43.

4. Mooring Line Failure Detection

In this section we propose a comparison between the models in the mooring line failure detection problem. Here we describe the problem, how the data is generated and organized, the conducted experiments and their results.

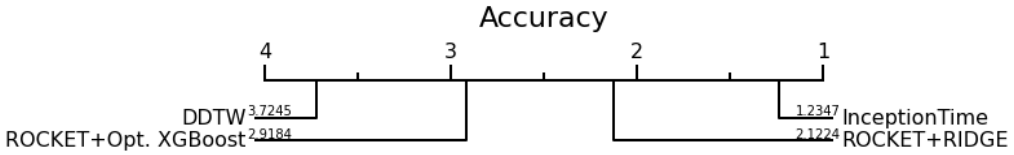

Figure 5. Critical Difference Diagram over the noise sensitivity experiment. Models closer to "1" present better results, while models closer to "4" present the worst ones. A horizontal line connecting two models means that their results have no statistical difference. Here we see that InceptionTime presents the best results.

4.1. Problem Definition

Offshore oil platforms present motions in 6 degrees of freedom, and the horizontal motions, given by the variables surge, sway and yaw, are the most affected when there is a rupture of some mooring line that holds the platform in the desired position in the ocean. Thus, the proposal here consists of classifying the time series of horizontal motions into 5 classes that indicate either that there is no break or in which of the 4 groups of lines there was a break. In previous approaches, motion data were used to train future motion prediction models whose outputs were compared with actual motions, generating a prediction error that feeds a classifier that identifies the failure [Saad et al. 2021]. Our approach, however, explores motion data as multivariate time series, using MTSC classifiers to detect whether or not there is a failure.

4.2. Dataset

The data used in this experiment is generated by the Dynasim simulator [Nishimoto et al. 2002]. Dynasim receives as input the hydrodynamic model of the platform, the environmental conditions to which the platform is subjected, as well as whether or not there is a line break – if so, the user must define which line broke and when the break occurred. As output, Dynasim provides the motion time series of the platform. We executed the experiments in a single draft – i.e., the depth at which the ship is submerged in water – of 16 meters, which represents the most frequent value of platform draft in this application. The hydrodynamic model was granted by the Petróleo Brasileiro SA Petrobras company. The environmental conditions were retrieved from a weather station located in Campos Basin of Rio de Janeiro, Brazil, from 2003 to 2006 in intervals of 3 hours, totaling 18000 different environmental conditions. For each environmental condition, Dynasim generates 3-hour motion series of the platform, as shown in Figure 6.

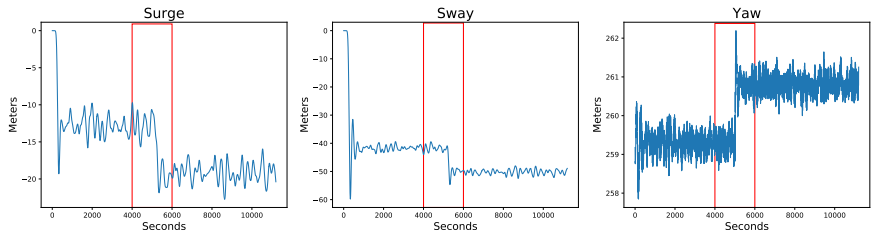

Figure 6. Example of the three horizontal motion series of the platform. The red rectangle highlights the motion series changes with line breakage.

We discarded the first 3600 seconds of motion, since they are subject to transient effects; and also remove 1400 seconds to ensure that all data would have the same size, given that line breakage was set to occur at 5000 seconds in selected simulations. This process was made to avoid an unbalanced dataset. Then, the center of the coordinate axis was shifted to be in the stern of the platform. Finally, the data was normalized and each time series was divided into 600 second non-overlapping windows, each of which was annotated by one of the 5 classes defined for the problem: no breakage or breakage in one of the four mooring line groups.

The 18000 real environmental conditions were divided into 7 clusters, from which 7002 environmental conditions were sampled to be simulated in Dynasim, giving a diversity of situations to which the platform is subject, with sea conditions ranging from calm to stormy. From the motion series resulting, 9 juxtaposed windows of 600 seconds were extracted (without overlapping), as in [Saad et al. 2021], resulting in a dataset with 63,018 windows equally distributed between the five classes. This dataset was divided into training (60%), validation (20%) and test (20%) sets.

4.3. Results

The experiments executed in the real problem consist of training of the 3 models (DDTW, ROCKET and InceptionTime) with the training data, as explained above. It is important to note that although this dataset is simulated, it represents a natural phenomenon and, also, still contains noise in its data.

Some tuning was done with the models to run the experiments. For InceptionTime, most of the parameters were kept the same as presented in the original paper [Ismail Fawaz et al. 2020]. The number of training epochs was set to 500 and the batch size was kept at 64; furthermore, the Adam optimizer was used with a learning rate of 10^{-3}. Another important configuration added to the training was a learning rate scheduler, that reduced the learning rate when the model stopped improving for a certain number of epochs; this component was kept the same as in the original InceptionTime. The model was then evaluated on the test set, resulting in a test accuracy of 99.77%. For ROCKET, we keep the number of random convolutional kernels at 10,000, as the default of the original paper. For the classifiers associated with ROCKET, the Ridge classifier was used as default, and XGBoost was tested with its default values and with optimized ones.

In Table 2 we have gathered the results obtained in the three models. The results achieved for ROCKET (associated with Ridge, XGBoost or Optimized XGBoost classifier) and InceptionTime are very promising. Almost all of the examples in the single draft dataset were correctly classified with any of these models. By a slight margin, ROCKET with the Ridge classifier had a higher accuracy.Dynamic Time Warping, however, didn't present positive results: DDTW training did not finish within an acceptable timeframe (12 hours), which was far more than what it took ROCKET and InceptionTime, highlighting the unsuitability of traditional state-of-the-art methods for massive datasets.

5. Conclusion and Future Work

Our experiments showed that the InceptionTime and ROCKET classifiers are highly effective in MTSC tasks. InceptionTime appears to be more robust when it comes to noise, while ROCKET has advantages in simplicity and runtime. It's also worth noting that XGBoost can be used with other classifiers, but its benefits have to be better analyzed.

Table 2. Accuracy of the three models in a single draft experiment

Model	Test Accuracy	Model for ROCKET	Test Accuracy
		+ XGBoost	99.77
DDTW	*timeout*	+ Opt. XGBoost	99.91
InceptionTime	99.77	+ Ridge Classifier	**99.97**

For the next steps, other benchmark datasets should be used to compare the performance of the models. In the context of the real problem, motivated by the excellent results presented by the modeling of the problem as MTSC, we will expand the classification to a dataset that includes all drafts and varied environmental conditions. In addition, we also intend to analyze this problem with data that comes directly from sensors located on oil platforms. This data is much noisier than the simulated data used in current experiments, making it a very challenging problem.

Acknowledgments

We thank the support given by the Brazilian National Council for Scientific and Technological Development (CNPq grant number 310085/2020-9), the Coordination for the Improvement of Higher Education Personnel (CAPES Finance Code 001), Brazil, and Itaú Unibanco S.A. through the PBI program of the *Centro de Ciência de Dados* (C^2D) of Escola Politécnica at Universidade de São Paulo. This work was partially carried out at the Center for Artificial Intelligence (C4AI-USP), with support from the Sao Paulo Research Foundation (FAPESP) under grant number 2019/07665-4 and by the IBM Corporation. Finally, the authors also thank Petróleo Brasileiro S.A. (Petrobras) for providing crucial data for this research.

References

Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660.

Dempster, A., Petitjean, F., and Webb, G. I. (2020). ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495. arXiv: 1910.13051.

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB*, 1:1542–1552.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Convolutional Networks*, pages 329–335. MIT Press. `http://www.deeplearningbook.org`.

Gupta, S. and Gupta, A. (2019). Dealing with Noise Problem in Machine Learning Datasets: A Systematic Review. *Procedia Computer Science*, 161:466–474.

Hills, J., Lines, J., Baranauskas, E., Mapp, J., and Bagnall, A. (2013). Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28.

Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., and Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Min. Knowl. Discov.*, 34(6):1936—-1962.

Kalapanidas, E., Avouris, N., Craciun, M., and Neagu, D. (2003). Machine learning algorithms: a study on noise sensitivity. In *Proc. 1st Balcan Conference in Informatics*, pages 356–365.

Kruskal, J. and Liberman, M. (1983). The symmetric time-warping problem: From continuous to discrete. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*.

Lines, J., Taylor, S., and Bagnall, A. (2016). Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1041–1046. IEEE.

Längkvist, M., Karlsson, L., and Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24.

Ma, K.-t., Shu, H., Smedley, P., L'Hostis, D., and Duggal, A. (2013). A historical review on integrity issues of permanent mooring systems. In *Offshore technology conference*. OnePetro.

McDonald, G. C. (2009). Ridge regression. *WIREs Computational Statistics*, 1(1):93–100.

Nishimoto, K., Fucatu, C., and Masetti, I. (2002). Dynasim - a time domain simulator of anchored fpso. *Journal of Offshore Mechanics and Arctic Engineering-transactions of The Asme - J OFFSHORE MECH ARCTIC ENG*, 124.

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., and Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449.

Saad, A. M., Schopp, F., Barreira, R. A., Santos, I. H. F., Tannuri, E. A., Gomi, E. S., and Costa, A. H. R. (2021). Using neural network approaches to detect mooring line failure. *IEEE Access*, 9:27678–27695.

Schäfer, P. (2015). The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1–23.

Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J., and Keogh, E. (2017). Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery*, 31(1):1–31.