SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

Highly Nonlinear Sensor Classification

COMPLEMENTARY TRAINING ACTIVITIES

Professor:
Enrico Tronci

Students:
Andrea Di Marco
Jemuel Espiritu Fuentes
{dimarco.1835169, fuentes.1803530}@studenti.uniroma1.it

Academic Year 2023/2024

Contents

1 Introduction

During the course of this project we attempted to predict the spikes in a multidimensional, highly nonlinear series of sensor readings belonging to a cyber-physical system.

We experimented with several AI-based techniques ranging from regression to classification. Given the highly nonlinear and noisy nature of the sensor array, regression quickly became unsuccessful so we eventually fell back to a classification problem. Instead of trying to predict the value of the timeseries the new task was to predict the sign and intensity of spikes representing values that deviated too much from the mean.

Clustering the sensors based on their correlation and predicting the sensors in the clusters at once proved to be successful in most cases but not all clusters had enough correlation and there wasn't any guarantee that the approach would work with other datasets. For this reason we also implemented a solution for predicting single sensors.

We implemented the project using Python 3 [1] and PyTorch [2], the source code can be found here.

2 Development

The implementation was done entirely on Python using the PyTorch and Scipy libraries.

2.1 Dataset

The starting dataset is comprised of 998 different sensors belonging to the same cyber-physical system. All sensors have been differentiated and normalized in $[-1, +1]$ with mean $\mu = 0$. The sensors containing null values in their respective series were excluded and resulting in 526 remaining sensors.

We handled the dataset using the libraries Pandas [3] and NumPy [4]. As previously stated the sensors are highly nonlinear and very noisy, making the task of predicting their behavior non trivial. A plot of the sensor 2 and its auto correlation function can be seen the in figure (1) and (2).

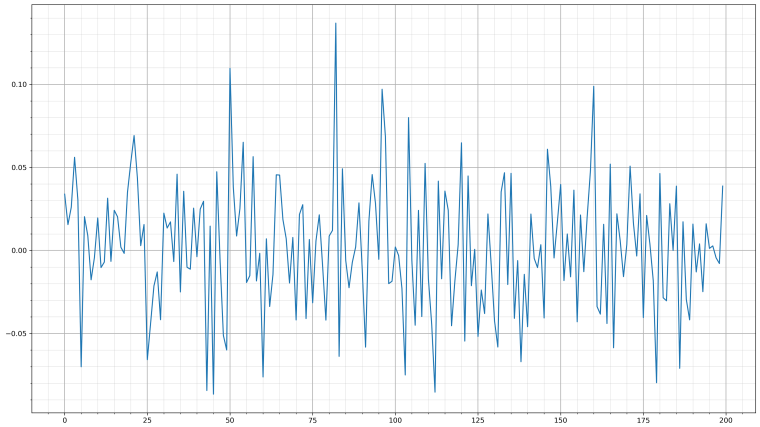

Figure 1: First 200 steps of sensor *2*

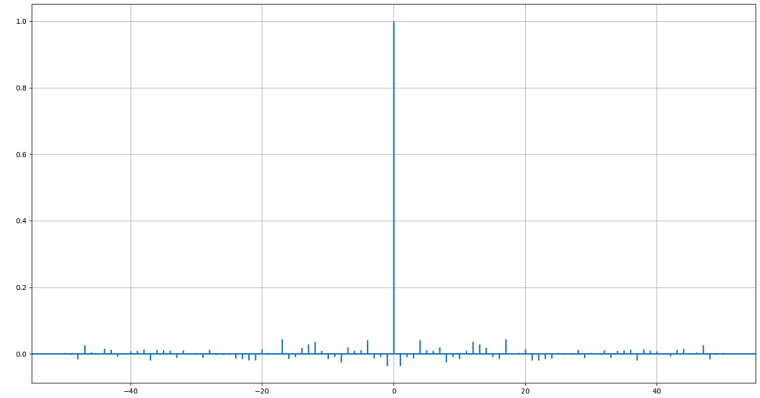

Figure 2: ACF of sensor *2*

2.2 FFSF

The first model tried was the Feed Forward Sensor Forecasting (FFSF), a n-layer feed forward network with $ReLU$ activation function for the intermediate layers. The model receives the multidimensional sample sequences unrolled into one dimension. The length of the sequence is fixed and can't be changed after initializing the model. Training used the *Mean Absolute Error* as the loss function. The FFSF model also makes use of privileged information [5] techniques by feeding its last layer with an array containing the sensor's mean, standard deviation, minimum value and maximum value observed withing a lookback window.

This model didn't perform well, as it can be seen in figure (3) where the blue line represents the ground truth, the red line represents the model's prediction on training data and the green line represents the model's prediction on testing data. Several attempts were made with FFSF by making predictions of single sensors or batches of sensors, all were unsuccessful. More information can be seen in Table 1.

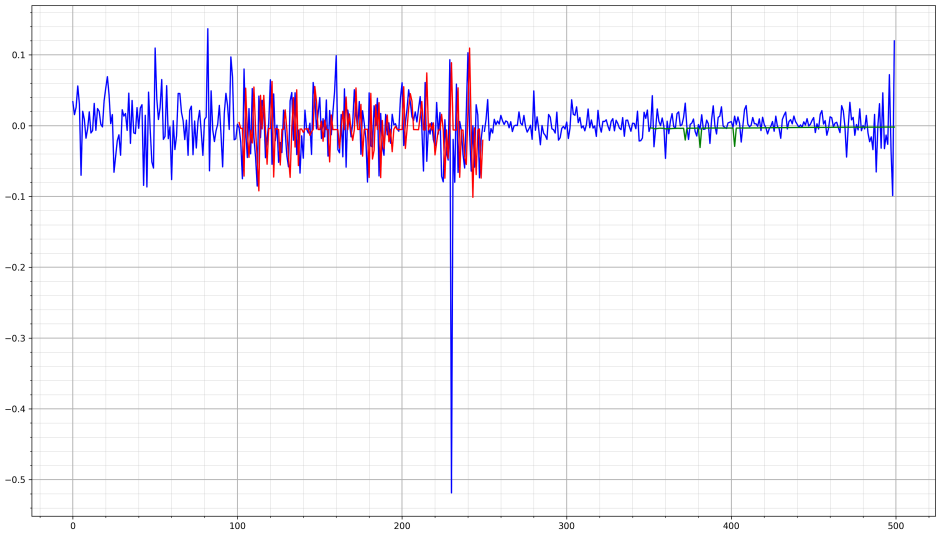

Figure 3: FFSF forecasting performance on sensor *2*.

2.3 SSF

The next model tried was the Simple Sensor Forecasting (SSF) model comprised of n-layers of LSTM [6] modules and a final fully connected layer. This model allowed to be trained with longer sequences and tested with smaller sequences. The loss function used during training was the *Mean Square Error*.

While unsuccessful in predicting the sensors separately, this model managed to almost perfectly predict the objective function when trained with all sensors at once, as it can be seen in figure (4) where the blue line represents the ground truth, the red line represents the model's prediction on training data and the green line represents the model's prediction on testing data. The model's setting for this to work requires more than 12 Million parameters, for this reason we shifted our attention towards smaller models. More information can be seen in Table 1.

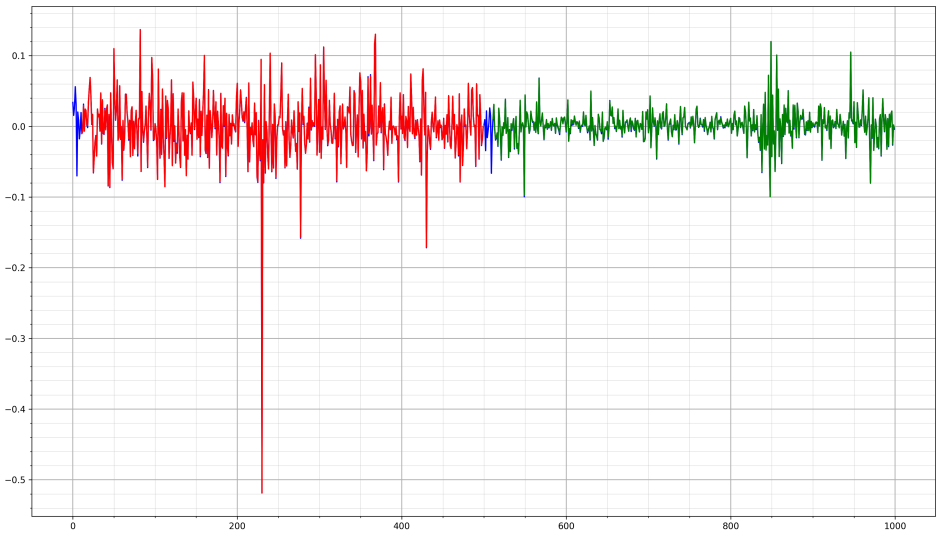

Figure 4: SSF forecasting performance on sensor *2*.

2.4 Clustering

The increase in SSF's performance when training with multiple sensors led us to believe that sensors are related with each other, for this reason we clustered them based on their correlation into 20 clusters.

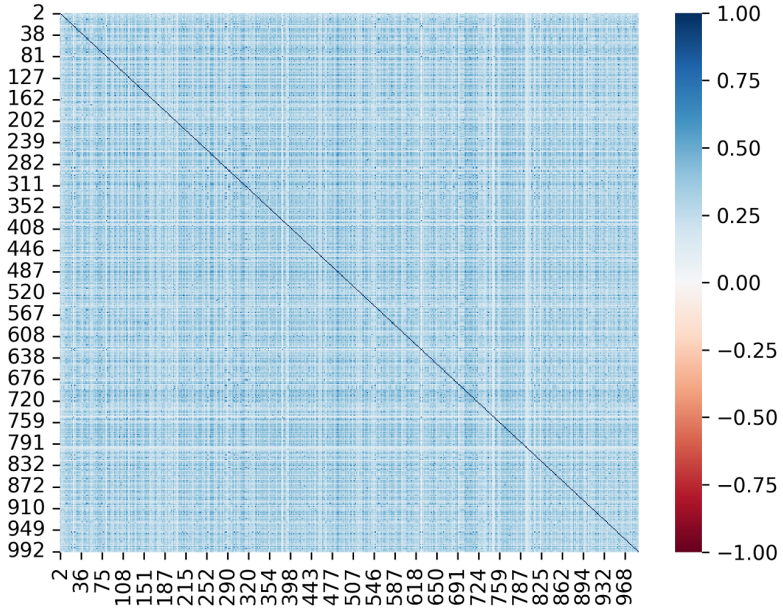

Figure 5: Correlation matrix of every sensor.

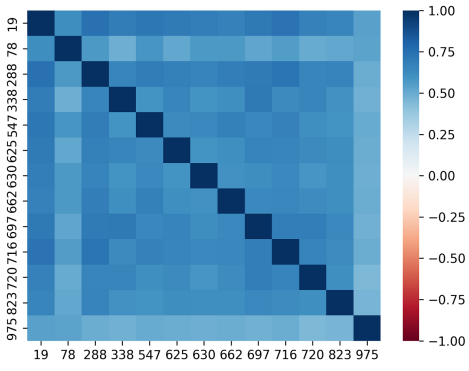

Figure 6: Most correlated cluster.

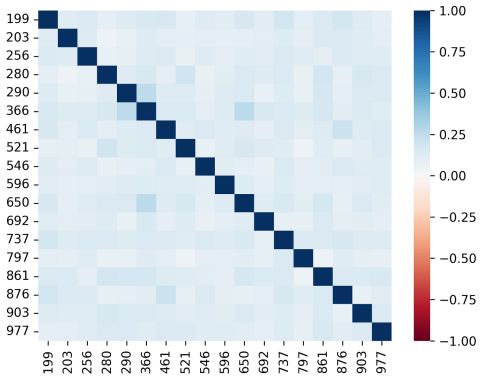

Figure 7: Least correlated cluster.

2.5 GSF (MSE)

The next model attempted is the GRU Sensor Forecasting (GSF) model. Similarly to SSF (2.3), this model has n-layers of GRU [7] modules and a final feed forward layer. GRU modules allow for dynamic sequence length and have significantly less parameters than LSTM modules. The loss function utilized during training was *Mean Square Error*.

Results were promising when asked to forecast clusters of sensors as figure (8) shows. More information can be seen in Table 1.

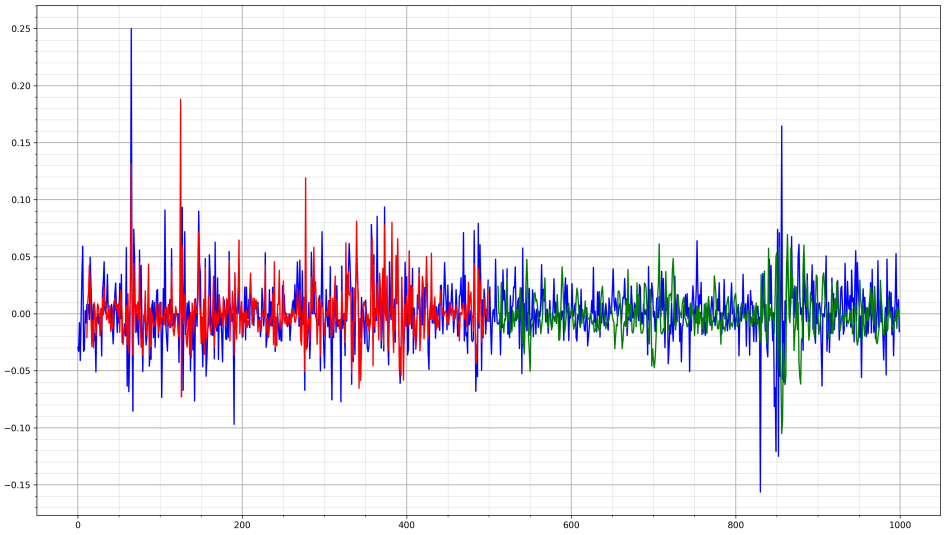

Figure 8: GSF forecasting for sensor *21*

2.6 GSF (Spike Loss)

A different loss was attempted for the GRU based model (2.5), focusing on amplifying the loss when in the presence of a spike and suppressing the loss when both the ground truth and the prediction absolute value are within the standard deviation.

$$L_s = \frac{1}{N}\sum_{i=0}^{N} |p_i - t_i| \left(\frac{|t_i - \mu_i|}{\sigma_i} + \frac{|p_i - t_i|}{\sqrt{\sigma_i}} \right) \tag{1}$$

Where:

- $B :=$ batch size
- $S :=$ sequence length
- $D :=$ sample dimension
- $N := B \times S \times D$
- $p :=$ predicted value
- $t :=$ target value
- $\mu :=$ sensor mean
- $\sigma :=$ sensor standard deviation

Experiments were positive although it is not trivial to see how much the model actually misses the target because of the nature of the new loss. The results of the new loss on the GSF model can be seen in figure (9). More information can be seen in Table 1.

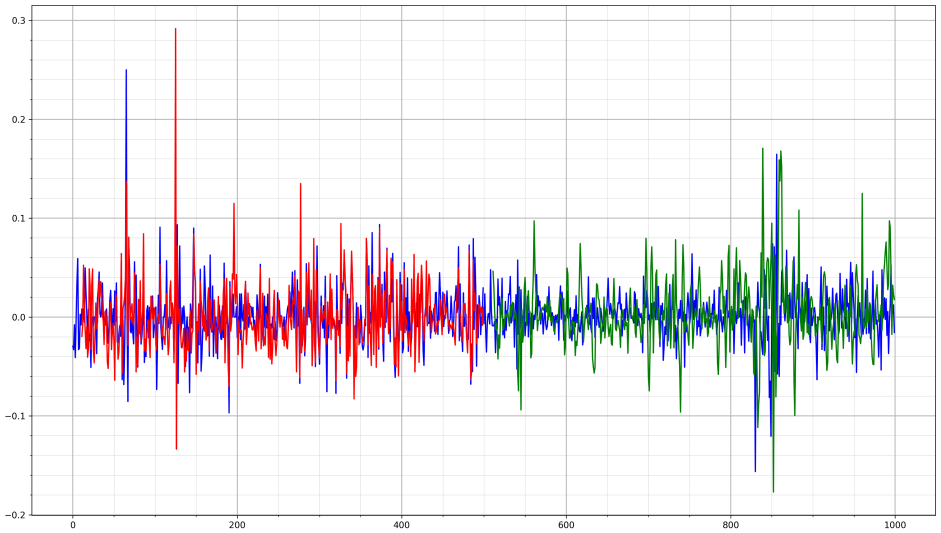

Figure 9: GSF with spike loss, forecasting sensor *21*

2.7 Classification Problem

Since our main goal was mainly to predict the sign and intensity of the spikes we shifted our attention from a regression problem to a classification problem, by discretizing the sensor's value to predict into 5 classes based on how far the value was from the mean. Given the individual sensor's mean μ and standard deviation σ, computed over a lookback window of 10 steps, for every sample s said classes have the meaning:

- ***-2***: $s - \mu < -2\sigma$
- ***-1***: $-2\sigma \leq s - \mu < -\sigma$
- ***0***: $-\sigma \leq s - \mu < \sigma$
- ***+1***: $\sigma \leq s - \mu < 2\sigma$
- ***+2***: $s - \mu \geq 2\sigma$

Due to the erratic nature of spikes, only a few sequences had their class set to ± 1 or ± 2, for this reason traditional techniques for training classification models were applied such as upsampling the minority classes. Upsampling proved useful for the Feed Forward based model (FFSD 2.10) but not for the GRU based classifier (SSD 2.11). We believe this happened because for FFSD each sequence had a single class describing the sample to be forecasted. GRU modules on the other hand return predictions at every step, meaning that a single sequence has multiple classes attached to it already, by performing upsampling we must discard the intermediate classes and only consider the last one, reducing the model's ability to learn.

Models were trained using a standard *Binary Cross-Entropy* loss.

2.8 Dataset Differentiation

One of the biggest improvement regarding the performance of the models was utilizing sensor data that was differentiated for a number of times. The maximum degree of differentiation was 3 as it brought the greatest increase in the performance of the models. In particular this technique is model agnostic as the general performance for any model is inferior to the one trained using a differentiated dataset. Figures (10), (11), (12) and (13) show the GRU-based model (SSD 2.11) on the differentiated datasets.

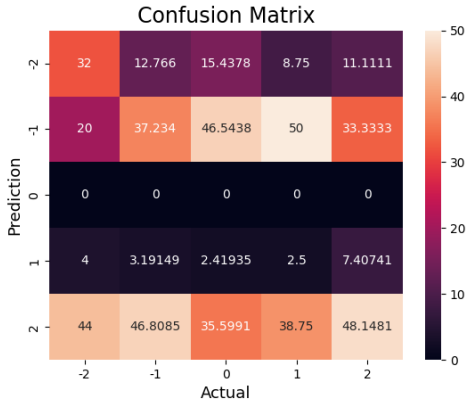

Figure 10: SSD model with 0 differentiation

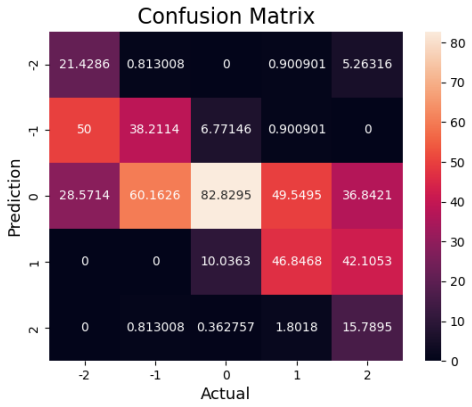

Figure 11: SSD model with 1 differentiation

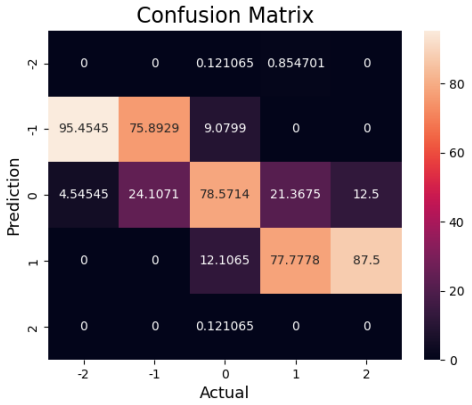

Figure 12: Dataset with 2 differentiation

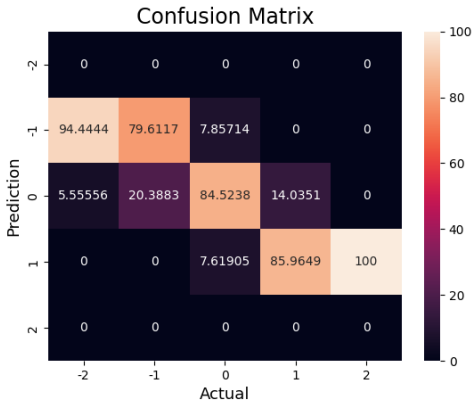

Figure 13: SSD model with 3 differentiation

2.9 Adversary

To have a comparison in performance for the models, an adversary was implemented using simpler machine learning techniques based on Gradient Boosting [8]. This adversary has the same task as the other models, classifying time series spikes after a sequence of sensor readings.

The implementation was done using the ensemble module of the *scikit-learn* [9] library. The adversary's main parameters are the number of estimators, the learning rate and the maximum depth which were fixed to 100, 1.0 and 2 respectively. The adversary gives high false positive rates and low true positive rates for the majority of the sensors. Figures (15), (14) and (16) show the worst, best and average results of adversaries trained on different sensors.

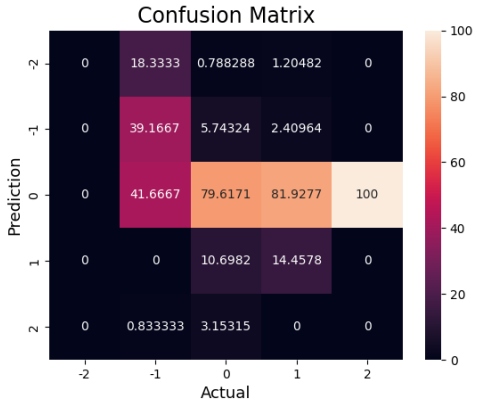

Figure 14: Adversary classification of sensor *548*

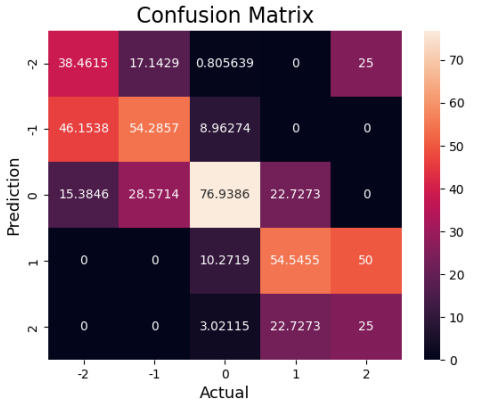

Figure 15: Adversary classification of sensor *801*

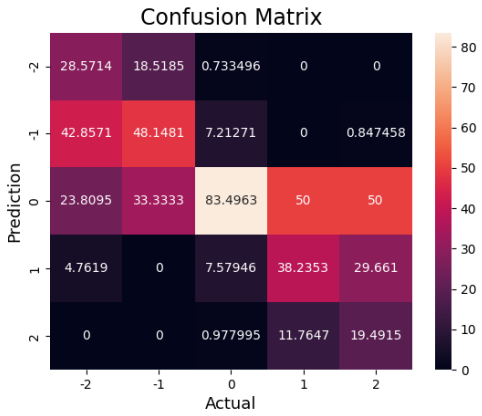

Figure 16: Adversary classification of sensor *630*

2.10 FFSD

The Feed Forward Sensor Discretization (FFSD) model is the evolution of the FFSF model (2.2) for the classification task. The main difference is the *Softmax* activation function, as it is standard for classification models. The model is tasked with predicting the spike intensity for the cluster seen during training.

The performance improved significantly with the help of upsampling and dataset differentiation as it can be seen in figure (17). More information can be seen in Table 1.

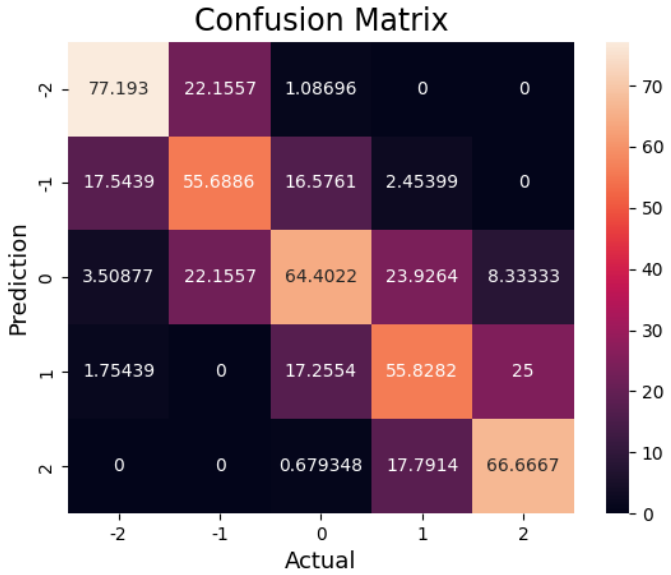

Figure 17: FFSD classification of sensor *8* with upsampling and differentiation

2.11 SSD (cluster)

The Simple Sensor Discretization (SSD) model is a GRU-based classifier with similar architecture to the GSF model (2.5) but using the *Softmax* activation function.

Results of this model when classifying sensors from clusters can be seen in figures (10), (11), (12) and (13).

This model turned out to be the overall best, although the solution proposed so far is not easily generalizable since not all cluster returned the same performance and most clusters required different hyperparameters. More information can be seen in Table 1.

2.12 SSD (single sensor)

To obtain a more general model we used the SSD (2.11) model for predicting sensors separately and without changing the hyperparameters. Results were satisfactory. We show the best, worst and a general results in figures (19), (18) and (20). We tested the method on all 526 sensors and the averaged results for both SSD and the adversary (2.9) are shown below in figures (22) and (21). More information can be seen in Table 1.

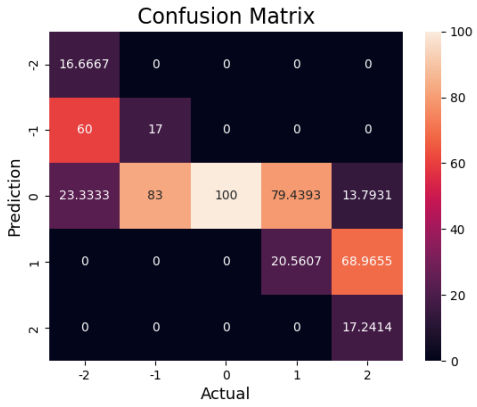

Figure 18: SSD classification of sensor *781*

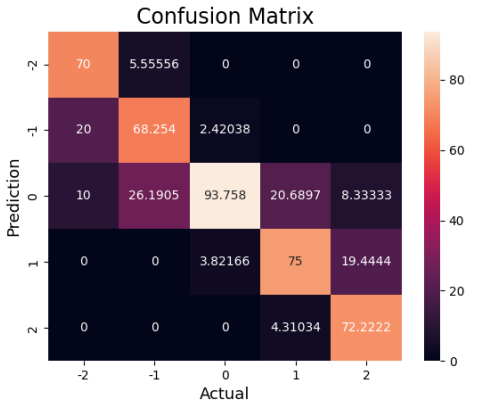

Figure 19: SSD classification of sensor *392*

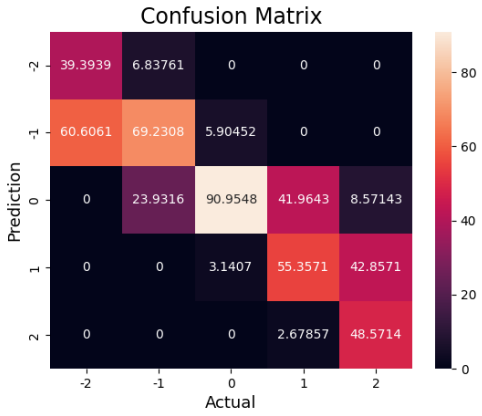

Figure 20: SSD classification of sensor *630*

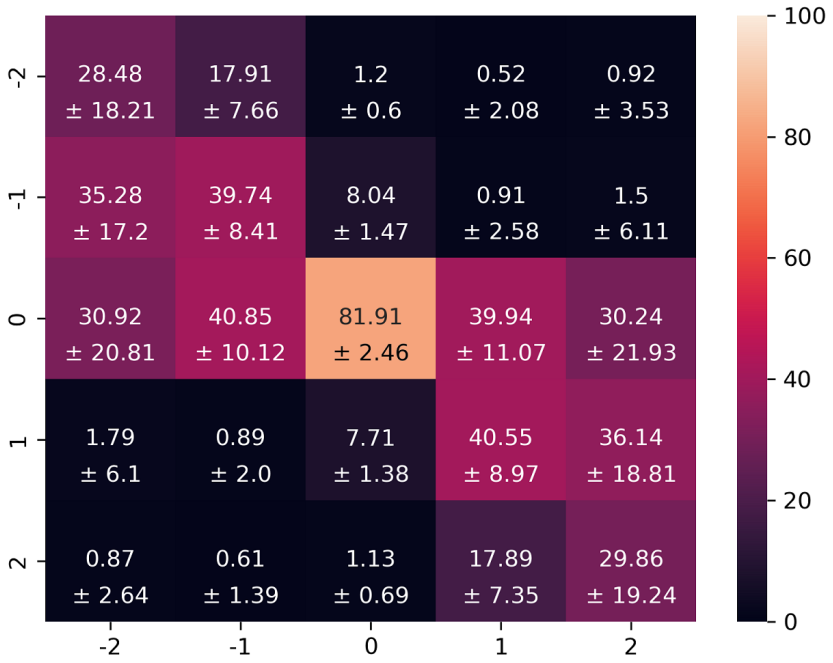

Figure 21: Adversary Confusion Matrix

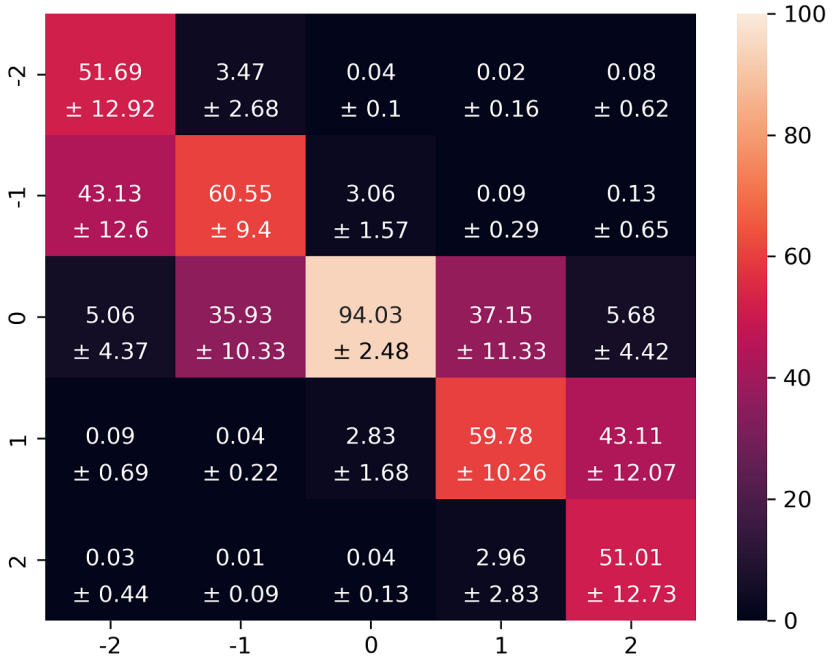

Figure 22: SSD Confusion Matrix

2.13 Hyperparameters

In table 1 we show the hyperparameters and a comparison of the different models used for predicting the time series based on various metrics.

Model	**Parameters**	**Loss**	**Hidden Dim.**	**Layers**	**Training Time**
FFSF 2.2	635	0.020	10	2	1867s
SSF 2.3**	12957526	0.002	1500	1	16114s
GSF (MSE) 2.5*	1685	0.021	12	2	268s
GSF (L_s) 2.6*	1685	0.016	12	2	266s
FFSD 2.10	269	0.673	4	3	681s
SSD (cluster) 2.11*	1105	0.100	10	2	336s
SSD 2.12	146	0.133	3	2	254s

Table 1: Hyperparameters for the models

* hyperparameters for cluster of 5 sensors.

** hyperparameters for all 526 sensors

3 Limitations and Future Work

The difficulty in working with noisy data proved to be the biggest factor in the changes regarding the task of this project. It would be interesting to develop models capable of predicting the next step of the time series with acceptable precision instead of being limited at classifiers.

The explainability of the classifier was not explored in this work and it would be helpful to understand the reasoning behind the choices made by the model.

An experimental activity that was not done in this project and would be a valid course of action is the testing of the methodology over a different dataset of a similar setting as the one used. Such activity could help in identifying potential weaknesses and possible evolution of our work.

The methodology could be used in conjunction with anomaly detection techniques, especially in safety critical scenarios such as the medical field, where being able to detect outliers representing potentially dangerous situations is of extreme importance.

4 Conclusions

The work presented shows the development of a classification tool capable of predicting spikes in a highly noisy data with great efficiency. The classifier used for spikes works with adequate precision on every sensor of the dataset. In particular the sign is correctly predicted in almost every case and the predicted class is within one step from the correct one with very high percentage.

When confronted with traditional machine learning techniques such as Gradient Boosting, the results are vastly superior. The adversary classifier presents erroneous outputs in many cases while the predictions of Deep Learning classifiers are much more reliable.

After trying different models it appears that GRU-based networks are a valid compromise between performance with the given task and size of the model, especially compared with the LSTM based classifier and the simpler but ineffective Feed Forward networks.

References

[1] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[3] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[5] Rickard K. A. Karlsson, Martin Willbo, Zeshan Hussain, Rahul G. Krishnan, David Sontag, and Fredrik D. Johansson. Using time-series privileged information for provably efficient learning of prediction models, 2022.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[8] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.