Supplementary Material

1 Experimental Settings

The results were generated using sktime package [1] version 0.23.0, and PyTorch [2] version 2.0.1 with CUDA version 11.8. The experiments were conducted on a system with an Intel(R) Xeon(R) Gold 5220R CPU, an NVIDIA RTX A5000 GPU, and 756 GB of RAM.

2 Termination Condition

Training on our model is stopped after 1000 iterations, with each iteration representing a complete pass through the training set. Additionally, early stopping with a threshold of 40 iterations is employed.

3 Hyper-Parameter Experiments

Due to the large size of the search space, we adopted random search for hyper-parameter tuning. For the binary and multi-class classification in the baseline models, 200 and 300 parameter sets from search space were randomly chosen, respectively. In the case of our method, 500 randomly selected parameter sets were used for both binary and multi-class classification and then we fine-tuned the best found parameters manually to reach the optimal performance.

In order to span more of the search space to find best hyper-parameters, we opted to run each set of parameters for each algorithm only once. However, the performance of non-deterministic algorithms (in this study, only the baseline SVM algorithm is deterministic) vary from run to run, leading to uncertainty in the results of each individual run. To address this variability, we executed each algorithm with its best-found hyper-parameters a total of 10 times, and only then compared them to each other.

In this study, there are two categories of hyper-parameters: preprocessing hyper-parameters and model hyper-parameters. In the following two sections, we describe each of them in detail.

3.1 Preprocessing Hyper-Parameters

A preprocessing hyper-parameter, determines the distribution of instances of each class in the training set and data operations conducted prior to inputting them into the algorithms. These operations are the normalization algorithm and the approach for handling *null* values within the dataset. Subsequently, data is prepared according to tuning result on these hyper-parameters for each algorithm and then is provided as input to the respective algorithm. Table 1 illustrates the possible values for each hyper-parameter.

Table 1. Preprocessing hyper-parameters. In our experiments the best set of hyper-parameters are chosen through random search.

Hyper-parameter		**Possible Values**	
Sampling strategy		Random sampling, active region sampling	
Method for Handling *null* values		Instance removal, average imputation, zero imputation	
Normalizations		Min-Max normalization, z-score normalization	
		Multi-class	**Binary-class**
Distribution of instances	Class Q	$400, 800, 1200, 1600$	$400, 800, 1200, , 4000$
	Class BC	$300, 600, 900, 1200$	
	Class M	$200, 400, 600, 800$	$200, 400, 600, , 1400$
	Class X	$40, 80, 120, 160$	

Handling *null* Values There are gaps in the satellite observations which are filled with *null* values in the SWAN-SF dataset in order to maintain the 12-minute cadence. To address these values, we employ one of three strategies, each chosen through hyper-parameter tuning based on what best suits each algorithm:

- **Instance Removal**: removing all of the MVTS instances that contain *null* values.
- **Average Imputation**: putting average of the values in train split in *null* positions.
- **Zero Imputation**: putting 0 in the place of *null* values.

For example, average imputation is best suited for our model's binary-class classification, instance removal is the best strategy for LSTM-FCN model in multi-class classification setting, and zero imputation is the best strategy for SVM multi-class classification.

Sampling strategies This hyper-parameter decides which of the two sampling strategies discussed in Section 4.4 should be used: random sampling or active region sampling.

Distribution of instances in Training Set This hyper-parameter determines the specific number of instances for each class in the training set. For multi-class classification, it specifies the number of instances for each of the Q, BC, M, and X classes. In the case of binary classification, it specifies the number of instances for the negative (Q and BC classes) and positive classes (M and X classes). Different combinations of the number of instances of each class is tested for each algorithm in hyper-parameter tuning and the best performing one is selected for its respective algorithm. The possible number of instances for each class is shown in Table 1. For example, in the case of the CIF method for multi-class

classification, the combination of 400 instances of Q, 300 of BC, 200 of M, and 160 of X classes produced the best results. This hyper-parameter essentially is optimizing the amount of undersampling of the data for each algorithm.

Normalizations Feature vectors of the dataset are normalized "locally", meaning that normalization considers only the statistics of the training set. Non-normalized runs yielded poor results, so they were excluded from our hyper-parameter tuning. For each algorithm, we applied two types of normalization on the data and chose the one that produced best results. The following are the normalization methods that we considered:

- **Min-Max Normalization**: each feature vector was scaled to the range of [0, 1] using the formula

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}.$$

- **Z-score Normalization**: each feature vector was scaled according to

$$x' = \frac{x - \mu}{\sigma},$$

  where μ represents the mean and σ represents the standard deviation withing the training set.

3.2 Model Hyper-Parameters

These hyper-parameters encompass the various aspects of our model. In this section, we present the results of two experiments related to two of these hyper-parameters: the learning rate and the train-validation ratio.

3.3 Learning Rate

We employed the Adam optimizer for model optimization, and the influence of different learning rates on the performance of the binary classifier is depicted in Figure 1.

3.4 Train-Validation Ratio

As explained in Section 4.4, a portion of the training set is extracted and assigned as the validation set. An experiment was conducted to examine the impact of different training-validation ratios on the model's performance in the binary setting. The results of this experiment are illustrated in Figure 3.4.

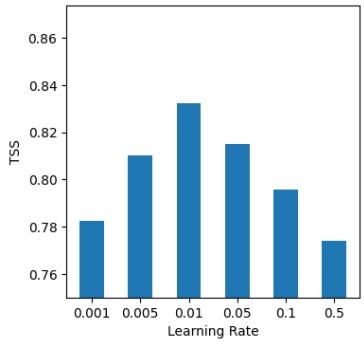

Fig. 1. Impact of various learning rates on the performance of our model in a binary classification setting.

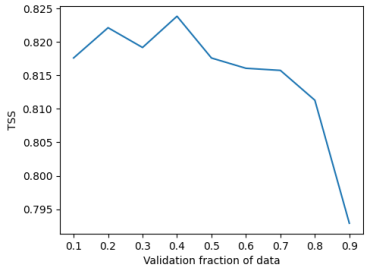

Fig. 2. The influence of different training-validation ratios on the performance of our model in the binary classification setting.

4 Ablation study

In our ablation study, we evaluated the impact of removing the FCN component from our neural network architecture in binary classification, in order to understand the specific contribution of the FCN, which operates after the convolutional blocks. We trained both the model without FCN and the original model 10 times, and the results of their performance are depicted in Figure 3, indicating that the presence of the FCN enhances the model's predictive performance.

References

1. Löning, Markus, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J. Király. "sktime: A unified interface for machine learning with time series." arXiv preprint arXiv:1909.07872 (2019).
2. Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al. "Pytorch: An imperative style, high-performance deep learning library." Advances in neural information processing systems 32 (2019).

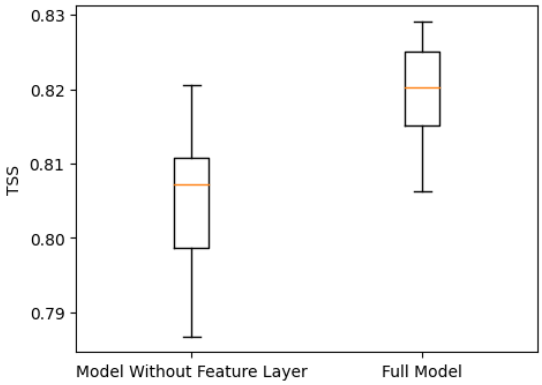

Fig. 3. Comparison of the full model with the model without FCN component.