TUM Data Innovation Lab

Munich Data Science Institute (MDSI)
Technical University of Munich

&

TUM Chair of Remote Sensing Technology cooperating with the German Space Operations Center (GSOC) at the German Aerospace Center (DLR)

Final report of project:

Power Signature of Satellite Components from Telemetry Data

Author	Junbo Leng, Vincent von Appen, Anish Pathak, Tabea Lüdde
Scientific Lead	Dr. Clemens Schefels, M. Sc. Leonard Schlag (DLR)
Co-Mentor	Prof. Dr. Michael Eineder (TUM)
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

Jul 2023

Contents

Abstract

To conduct a satellite mission successfully, satellite operators rely on telemetry data to control the satellite in outer space. In the event of a corrupted or defective sensor, operating without knowledge of its status becomes challenging. Therefore, this project aims to predict which status components are ON and which are OFF given signature values at certain times. This is based on real telemetry data obtained from satellite missions conducted by the German Aerospace Center (German: Deutsches Zentrum für Luft- und Raumfahrt).

This project involves analyzing and preprocessing a given large-scale dataset, as well as the implementation of appropriate Machine Learning models. The main findings of an extensive literature research on novel approaches are outlined in section 2 which covers all the steps undertaken.

The given input from the German Aerospace Center (German: Deutsches Zentrum für Luft- und Raumfahrt) consists out of 25 signature files, whereas each signature corresponds to one sensor on the satellite that captures a parameter. Further, 19 status files are provided, whereas each status file corresponds to one component on the satellite and when it was ON or OFF. All files have different sampling rates and hence different sizes between mostly ~ 450.000 and $\sim 4 \times 10^6$ rows.

In subsection 3.2 we noticed a uneven distribution of values of status and signature component on time due to multi-rate sampling, which we tackled by resampling time-wisely. Further, the individual signatures are highly biased as the spikes are not synchronous.

As the signature and status files with corresponding indexes are not correlated to each other, we implemented Transformer Architecture for Time Series Data, InceptionTime, Multi-Channel Attention-Based Long Short-Term Memory with Fully Convolutional Network and Rocket as machine learning models with the objective to predict values of (power) signatures. Therefore, we summed up most novel literature in section 2 and the coding steps including the creation of the dataloader, model architecture and hyperparameter tuning in subsection 5.2. We conducted evaluations of all models using various data preparation methods and recorded their corresponding accurancy metrics in section 6.

The capability to identify anomalies in power consumption of satellites holds significant value, as it not only aids in detecting sensor failures but also ensures potential sensor or equipment issues can be addressed promptly. Consequently, this project contributes to preventing further system downtime.

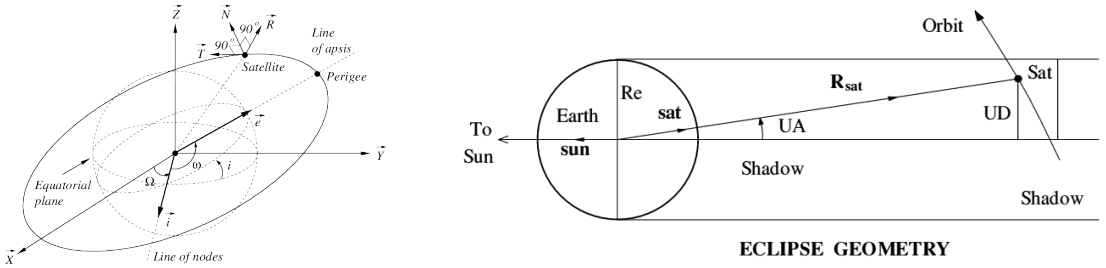
Abbreviations List

AI	artificial intelligence
CNN	Convolutional Neural Network
Comp.	Compatibility
Comp. exp.	Computational Expense
COTE	Collective of Transformation-based Ensembles
DLR	German Aerospace Center (German: Deutsches Zentrum für Luft- und Raumfahrt)
DOI	Difficulty of implementation
FCN	Fully Convolutional Network
Interp.	Interpretability
IT	InceptionTime
LSTM	Long Short-Term Memory
MALSTM-FCN	Multi-Channel Attention-Based Long Short-Term Memory with Fully Convolutional Network
MCNN	Multi-scale Convolutional Neural Network
ML	Machine Learning
MTS	Multivariate time series
MTSC	Multivariate time series classification
PA	Perceived Accuracy
ResNet	Residual Network
RNN	recurrent neural network
SOTA	state-of-the-art
ST	Shapelet Transform
SVM	support vector machine
t-LeNet	Time Le-Net
TS	time series
TST	Transformer Architecture for Time Series Data
tsai	time series artificial intelligence (AI)
TSC	time series classification

1 Introduction

Satellites have to be remotely operated from earth once launched into space. Therefore, they are equipped with many different sensors to keep track of the satellite's system status. Some satellites operated by the German Space Operations Center (GSOC) at German Aerospace Center (German: Deutsches Zentrum für Luft- und Raumfahrt) (DLR), transmit about 80′000 telemetry parameters. Telemetry data is vital for the safe operation and control of a satellite in outer space. In the event of a sensor malfunction, the inability to repair it due to its location in space poses a significant challenge. Operating the satellite without knowledge of its component status becomes risky, potentially leading to incorrect decisions and mission failure. To address this issue, we propose implementing a backup system for component status sensing.

The aim of this project is to leverage Machine Learning (ML) approaches to predict the status values of power signatures of components based on the provided satellite telemetry data. The concept of power signatures, inspired by the electrical power characteristics observed in smart meters, refers to the unique power consumption patterns exhibited by different components over time. By analyzing and classifying these characteristic signature values, valuable insights into the status and behavior of satellite components can be gained. The telemetry data encompasses various parameters, including the eclipse value, which represents the occurrence of an eclipse when the satellite is in Earth's shadow, see Figure 1b.



(a) Satellites orbit parameter [1] (b) Eclipse of a satellite in the earths shadow [2]

Figure 1: Schematic satellite overview

Satellite orbits are characterized by several important parameters that define their motion and position in space, see Fig. 1a. Some of the key parameters of a satellite orbit are: *altitude* refers to the vertical distance between the satellite and the Earth's surface while the *attitude* refers to the orientation to a reference frame. The *period* of a satellite orbit is the time taken by the satellite to complete one full revolution around the Earth.

This project holds several advantages for the DLR, including status prediction, error detection in case of sensor failure, redundancy in communication failures, optimization of satellites, and informing future missions and construction of satellites. This academic report outlines the objectives and tasks involved, including data exploration and preparation, surveying pattern recognition research, creating models, ensuring component-

agnosticism, and documents the process.

Combining multiple signature parameters, such as battery voltage and the eclipse flag, can provide further insights. In the literature this is refereed to as a Multivariate time series (MTS). MTS is a type of data that captures the evolution of multiple interrelated variables or features over time. Each observation represents a collection of synchronous measurements taken simultaneously across different dimensions. These dimensions can be diverse and can include various physical quantities, here: sensor readings, such as the battery voltage or the eclipse flag. Unlike univariate time series, where only one variable is observed over time, MTS provides a more comprehensive and interconnected view of the underlying phenomena. Due to the high dimensionality and interactions between the variables, the analysis is complex. However, it can provide valuable insights into complex systems and facilitate forecasting, anomaly detection, classification, and other time-dependent analyses. This project is mostly focusing on time series (TS) classification.

1.1 Time Series Classification

Definition 1.1. A univariate time series $X = [x_1, x_2, ..., x_T]$ is an ordered set of real values. The length of X is equal to the number of real values T.

Definition 1.2. An M-dimensional MTS, $X = [X^1, X^2, ..., X^M]$ consists of M different univariate time series with $X^i \in \mathbb{R}^T$.

Definition 1.1 and 1.2 provide a definition of MTS adapted from [3]. Figure 2 illustrates the problem graphically. The MTS consist of M different univariate time series. Section 3 and 4 discuss the shape of the data in more detail. The non-linear transformations are discussed theoretical in Section 2 and the implementation in section 5.2. The output should indicate one of the K classes by a probability distribution.

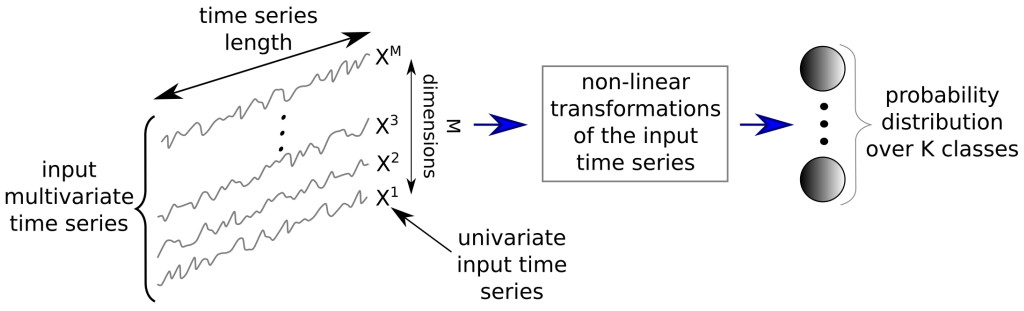


Figure 2: Time Series Classification [3]

1.2 Problem Definition and Goals of the Project

The goal of this project is to investigate ML approaches to predict binary status values of (power) signatures of components in given satellite status parameters. The status values are time series consisting of values that can be mapped to a logical 1 or 0. Thus, the objective is to utilize this principle and classify different status values based on their

characteristic signature values.

Consequently, the term power signature refers to the power consumption over time. While this project focuses on telemetry, such as the eclipse value, there are other parameters present. For instance, combining multiple signature parameters like battery voltage and the eclipse flag can provide additional insights, as it would be when the solar panels charge the battery as soon as the satellite is lit by the sun. Expected advantages of this new development for the DLR include:

- An approach can investigate given time series telemetry data, and accurately predict the parameter status values
- If there are anomalies in the power consumption due to some other reason, this method can be used to catch errors in case of sensor failure.
- If other measures fail, one can apply it to check the status of components. When a system is not reachable through the satellite communication bus, this method offers redundancy as it allows for predicting the status based on other sensors.
- Potentially this may be used to optimize satellites, specifically tiny CubeSats due to their size & weight constraints because fewer hardware sensors might be used.
- The DLR could use this initial investigation to optimize the future missions and construction of satellites.

1.3 Objectives and Tasks

- Investigate the telemetry data and explore approaches for dealing with the issues (i.e. incongruencies in sampling rate, data types and missing values) [Section 3]
- Survey the current research in pattern recognition and identifying power signatures [Section 2]
- Prepare the data using analytical techniques in order to be able to perform ML and classical methods [Section 4]
- Create time series forecasting models that can identify the power signature for a component at a given time, decide on best-fitting one and optimize for a chosen metric, such as performance and accuracy [Section 5.2]
- Explore ways to make it as component-agnostic as possible [Section 5.2]

1.4 Acceptance Criteria

This section covers the acceptance criteria for the project. This was divided into required criteria, optional criteria and delimitation. Delimitations define criteria that are outside the scope of this project and intentionally omitted to draw a clear line.

Required Criteria

- Reproducible method of pre-processing the data
- Use an analytical approach to subset the data (e.g., deal with the bias in data)
- Evaluate performance metrics of the model
- Accuracy above 90% (validation, test)

- Precision, recall, F1 score at least 75%
- Avoid overfitting (e.g., monitor good generalization of the approach by losses, etc.)
- Model should work accurately for different components (component-agnostic)
- Train, test and validate data set, e.g. a feasible split would be 70/20/10
- Monitor and document training approaches, e.g. Tensorboard
- Compare approaches with the same pre-processed data
- Document work in a visual manner
- Use tables for comparison
- Comments in Code

Optional Criteria

- Define performance metrics, such as accuracy above 95%
- Level of confidence for every label which corresponds to the actual probability of the label
- Test scenarios, have pre-written test cases as part of the deployment process to ensure code quality
- Effectiveness on different datasets (test on another publicly available dataset or sourced from DLR if possible)
- Explainability (a highly accurate deep neural network is not as good as a slightly less accurate model that's easy to understand)
- A model with outputs power signatures of different components like shapelets, meaning shapes that cluster explain the decision

Project Delimitation

The project assumes that power signatures remain consistent throughout the data, without degradation. Bias in the data needs to be addressed, possibly by excluding certain data. The project does not require any additional artifacts beyond the specified ones (code base, final report, and final presentation), and publishing a potential paper is optional. There are no limitations on the learning approaches to be used, and frameworks can be utilized for implementation. The number of framework dependencies should not be reduced. Real-time predictions are not necessary; batch processing is sufficient. There are no specific constraints on compute requirements, except for accessibility. The chosen approach must be applicable to the provided data without the need for generalization to other satellites.

2 Related Work

In order to choose adequate methods for the given project, beforehand, novel approaches in MTS are researched.

2.1 Statistical Methods and Machine Learning

To make predictions in classification tasks, the key idea of proximity forest is that instances with similar features are presumed to belong to the same class or have similar labels. This

approach leverages that instances within close proximity in feature space are more likely to share common characteristics. In comparison, support vector machine (SVM) underlies the principle of finding an optimal hyperplane that separates instances belonging to different classes. Therefore, the key idea of SVM is to identify a decision boundary, which maximizes the margin between classes. Both proximity forest and SVM are capable of handling high-dimensional data and capturing intricate patterns, albeit using statistical foundations.

In the context of a project involving satellite data analysis, several methods are relevant for MTS processing. Bagnall et al. [4] conduct an experimental evaluation of different algorithms for time series classification, including statistical methods, machine learning models, and deep learning approaches. They provide a comprehensive analysis of the strengths and weaknesses of these algorithms on a diverse set of time series datasets, helping researchers and practitioners understand their performance characteristics and choose appropriate methods for time series classification tasks. As result, Collective of Transformation-based Ensembles (COTE) methods were found superior to other published techniques with average accuracy of 8% higher than Dynamic Time Warping. Hive-COTE 2.0 is a recent and powerful technique that combines diverse classifiers, such as shapelet transform-based classifiers and time series forest, to achieve state-of-the-art (SOTA) performance [5]. Rocket is another notable method that utilizes random convolutional kernels to transform time series into fixed-length feature vectors, enabling efficient classification with linear models [6]. Shapelets, on the other hand, are discriminative sub-series that capture local patterns within time series, making them valuable for tasks such as anomaly detection or classification [7]. Additionally, the shapes can be extracted. Thus this approach is really good in terms of interpretability.
Toth et al. [8] propose a Bayesian learning method that combines Gaussian processes with signature covariances and deep learning models to handle sequential data. This approach allows for the analysis of sequences of different lengths and achieves competitive results in multivariate time series classification tasks. It could be applied in a multivariate time series setting to classify and make predictions based on sequential data with varying lengths.

2.2 Deep Learning Approaches

Fawaz et. al. provides a comparison of various neural network architectures for time series classification in [3]. Among the examined architectures, three stand out: Fully Convolutional Network (FCN), Residual Network (ResNet), and encoder.

The FCN architecture harnesses the power of convolutions to extract features from time series data. It employs three convolutional blocks with batch normalization, ReLU activation, and average pooling over the time dimension. Notably, the filters used in the convolutions have specific sizes, and the stride and padding are adjusted accordingly. This design allows FCN to capture relevant patterns. However, FCN may face challenges when dealing with very long time series due to fixed filter lengths [9].

Similar to FCN, the ResNet architecture introduces residual connections, enabling the

training of deeper networks without encountering vanishing gradients. Additionally, it incorporates a global average pooling (GAP) layer after the convolutions, facilitating efficient information aggregation across the entire time series. ResNet has demonstrated superior performance in time series classification tasks. However, its deeper architecture may require more computational resources [9].

Inspired by FCN, the encoder architecture incorporates modified filters and an attention mechanism that learns the importance of different elements in the time series. This attention mechanism proves useful for accurate classification [10]. While other architectures, such as Multi-Layer Perceptron (MLP), Multi-scale Convolutional Neural Network (MCNN) [11], Time Le-Net (t-LeNet) [12], Multi Channel Deep Convolutional Neural Network (MCDCNN) [13; 14], and Time Warping Invariant Echo State Network (TWIESN) [15], have also been explored, they are less relevant for this project. Since they usually provide less accuracy on the test data and infeasible data augmentation methods, such as window warping in t-LeNet, which manipulate the time series by squeezing or dilating them. Further some Window Slicing (WS) is applied in MCNN.

Another methods leveraged Convolutional Neural Networks (CNNs) methods for fault detection in satellite power systems. It employs the Stockwell transform, a time-frequency analysis technique, to map the time domain signal into the time-frequency domain. This transformed data is then processed using a CNN for fault detection. The Stockwell transform provides a more detailed representation of the signal in both time and frequency domains compared to traditional methods like the Fourier transform. This approach has shown promising results in achieving high accuracy for fault detection in satellite power systems [16]. Further, the autoregressive CNN model is designed to capture temporal dependencies and local patterns in asynchronous time series data. It leverages dilated convolutions to account for irregular time intervals between observations. The model has shown excelent performance compared to traditional methods and standard CNNs in prediction accuracy for asynchronous time series [17].

In a multivariate time series (MTS) setting, the AR-CNN model can be applied to tasks such as forecasting, anomaly detection, and classification. It can effectively handle time series data with varying sampling rates and irregular time intervals, making it suitable for domains where the observations arrive asynchronously across multiple variables.

In summary, the FCN, ResNet, and encoder architectures have shown promise in time series classification tasks. FCN excels in leveraging convolutions for feature extraction, ResNet addresses the challenge of training deeper networks, and Encoder incorporates attention mechanisms for improved classification accuracy.

2.3 Image Methods

Some Deep Learning (DL) methods use convolutional layers with time series as input vectors, while others convert the time series data into 2D images. This image-based approach provides the advantage of being invariant to different sampling rates. Chen et al.

introduce the Relative Position Matrix (RPM) technique, which encodes MTS data into a matrix representing the relative positions among its components. This matrix is then fed into a CNN for classification, capturing spatial dependencies and improving classification performance. Yang et al. extend this approach by converting MTS data into 2D colored images using encoding techniques such as Gramian Angular Summation Field (GASF), Gramian Angular Difference Field (GADF), and Markov Transition Field (MTF). These methods offer visual representations that capture patterns and relationships in the time series. However, further investigation is needed to determine the computational resources required for practical implementations. Despite this, these approaches make valuable contributions to time series analysis and classification, opening avenues for further research in analyzing MTS data.

2.4 LSTM

LSTM is a recurrent neural network (RNN) architecture that has gained significant attention and proven to be highly effective in modeling and forecasting MTS data. It addresses the limitations of traditional RNNs, such as the vanishing and exploding gradient problems, by incorporating memory cells that allow the network to selectively retain and forget information [18]. This unique memory cell structure enables LSTM to capture long-range dependencies and learn complex temporal patterns within the data. By effectively preserving and updating information over time, LSTM can capture both short-term fluctuations and long-term trends, making it a powerful tool for analyzing and predicting MTS data in various domains. The versatility and robustness of LSTM make it a valuable asset in the field of MTS analysis [18; 19].

In their work, [19] address the task of Multivariate time series classification (MTSC) and propose two deep learning models. The authors introduce a transformation of the Long Short-Term Memory Fully Convolutional Network (LSTM-FCN) and Attention LSTM-FCN (ALSTM-FCN) models into MTSC models. They achieve this by augmenting the fully convolutional block with a squeeze-and-excitation block to further improve accuracy. Notably, these proposed models demonstrate superior performance while requiring minimal preprocessing. They prove to be effective on various complex MTSC tasks such as activity recognition and action recognition. Additionally, the models are efficient at test time and possess a small memory footprint, making them suitable for deployment on memory-constrained systems. The experimental results on 35 datasets validate the strength of the proposed models, showing impressive classification accuracies across a wide range of datasets.

2.5 InceptionTime

InceptionTime initially proposed for end-to-end image classification [20] and has since ben adapted for time series classification (TSC) [3]. According to Fawaz et al. [3], each Inception network in InceptionTime composed of two different Inception blocks, which further contain three Inception modules. These modules surpass traditional fully convolutional layers found in other network architectures by enabling multi-scale feature extraction and adaptability to different domains. This is made possible through the

use of parallel convolutional layers with varying filter sizes. By incorporating filters with different sizes, each filter captures features at a different scales, allowing for the extraction of multi-scale information. This attribute enhances interpretability by making it easier to identify which filters are responsible for detecting specific patterns or characteristic within time series. Additionally, Inception blocks utilize parameter sharing across the parallel convolutional layers, which reduces computational complexity while minimizing the amount of parameters.

2.6 Transformer

Transformers, originally developed for natural language processing tasks, can also be applied for TSC. Transformers excel at capturing long-range dependencies and contextual information, making them suitable for analyzing sequential data. In the context of time series classification, transformers can model temporal patterns, identify relevant features, and make accurate predictions. By leveraging self-attention mechanisms, transformers can focus on different time steps and capture complex relationships within the time series. With their ability to handle variable-length inputs, transformers provide a flexible and powerful framework.

Drawing inspiration from the success of transformers in NLP and sequence generation tasks, a framework for training a transformer encoder to extract dense vector representations of MTS through an autoregressive denoising objective is proposed by [21]. They demonstrate that the transformer model outperform existing approaches, even with limited training data, and show that unsupervised learning provides an advantage over supervised learning for MTSC and regression tasks. Despite preconceptions about the computational requirements of transformers, the authors show that their models, with a reduced number of parameters, can be efficiently trained using standard GPUs. These representations can then be utilized for various downstream tasks, such as regression, classification, imputation, and forecasting.

3 Exploratory Data Analysis

3.1 Data Acquisition

The telemetry data was obtained during a mission of the TET-1-satellite. The satellite is part of the "On-orbit Verification of new techniques and technologies"(OOV) program and was built in Germany to facilitate the testing and qualification of new systems in space conditions [22; 23]. It provided research institutions and industrial companies with a versatile platform to validate their latest products. The satellite offered a payload volume of 460 x 460 x 428 mm^3, weighing up to 50 kg. Launched in July 2012, TET-1 operated in a target orbit at approximately 520 kilometers altitude, with an orbital period of 90 minutes and 60 minutes of sunlight exposure [22; 23]. For more intuition on the parameters, see Figure 1a. For more detailed parameters about the satellite and its orbit, see Table 1.

Satellite	Parameter	Orbit	Parameter (3σ)
Satellite weight	120 kg	Semi-major axis	6879.0 km (± 10 km)
Dimensions	$670 \times 580 \times 880 mm^3$	Eccentricity	0.00148 (±0.0015)
Start date	22 July 2012	Inclination	(97.43° ± 8 arcmin)

Table 1: TET-1 Satellite and Orbit Parameters

There are two types of data we obtained from DLR: signature_components and status_components. The signature_components contain time series of the values of physical quantities, such as voltage, current, byte flow, etc. The correspondence between components (such as sensors, appliances) on the satellite and signature_components is unknown to us and not relevant to the task. The status_components, on the other hand, document the ON/OFF status of certain components on the satellite, which exact component sent out which status_components is also unknown. Additionally, there exists no direct mapping between signature_components and status_components.

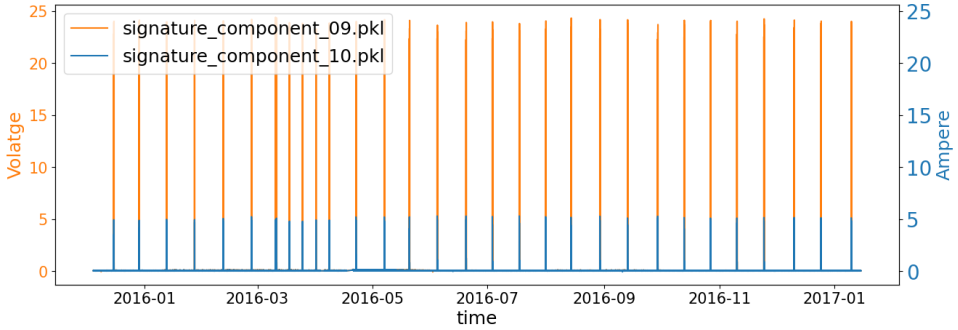
There are in total 19 signature_component_X.pkl files (.pkl, short for pickle, is a popular format used to serialize and deserialize data types), where X corresponds to the index respectively as well as 25 status_component_X.pkl files with indexes ranging from 1-18 and 20-26. The total size of the data is around 320Mb in compressed form.

The dataset contains two main data fields. The timestamp_ms captures the time when an event occurred and is represented as an integer (int64), a floating-point number (float64), or a string. The value field represents the measurement or value associated with the event and can be expressed as a float64, int64, or string data type. The sampling rate for both fields is varying. Important to note here is that the signature and status files with corresponding indexes are not correlated to each other. Arbitrary 19 signature files with different units are given. The 25 status files contain the labels of which the power signatures should be predicted at a later time.

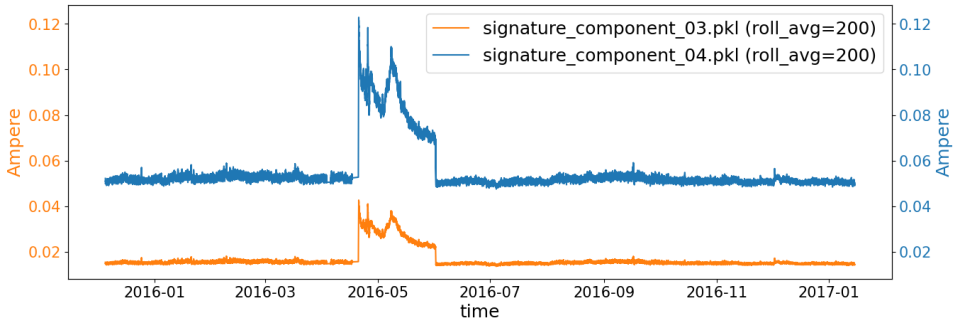
3.2 Analysis of Raw Data

3.2.1 Signature Component Analysis

Signature 1 and 2 (see Figure 11 and Figure 12) have the precisely same values, their correlation is 1, and their units are both Volt(see Figure 4). According to a discussion with our DLR advisers, One of them could be data collected by a backup sensor, and it would make sense to drop it to avoid abundance. Signature 3 and 4 (see Figure 3b) share similar trends in values, and they are highly correlated with a correlation 0.98, and they are both in Ampere. Signature 5 and 7 (see Figure 13 and Figure 14) are loosely correlated. Their units are both Ampere. Signature 9 (Volt) and 10 (Ampere) exhibit 'spikes' at the exact times, in total 30 of them, as shown in Figure 3a. Their correlation is around 0.73, meaning they are highly correlated.



(a) Signature component 9 and 10



(b) Rolled value of signature component 3 and 4

Figure 3: Comparisons of signature value pairs

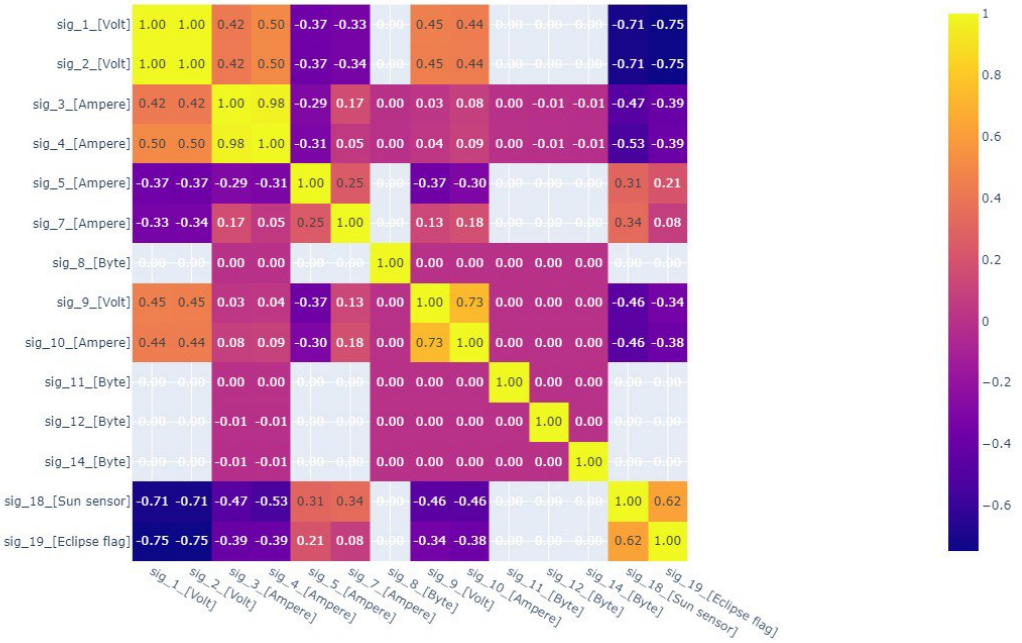


Figure 4: Correlation matrix of signature components

Signature 8, 11, 12, 14 (see Figure 15 and Figure 16) are highly biased. Each has one visible spike, which is not synchronous. Further, they have a small mean with a comparably large standard deviation Table 6. Their units are all in Bytes, which could be linked to data transmission. Signature 18 is the estimated z-component of the sun vector. It

is highly correlated with signature 19 (Boolean), the eclipse flag, which you can see in Figure 4. We skip the signature components that have a constant value throughout the entire time span, which are signature 6, 13, 15, 16, 17. An overview is provided in Figure 17.

The signature analysis table (Table 6) in Appendix X.I provides a comprehensive overview of various statistics in a dataset. The voltage measurements in signature 1 exhibit relatively small variations with a mean of 22.053 volts. Signatures 5 and 7 show diverse current values, including negative readings, with considerable standard deviations. This table provides valuable insights into the dataset, allowing further analysis and interpretation of the measured signatures.

3.2.2 Status Component Analysis

One characteristic of the status component is the high bias. Among the 15 status components, it is noticeable that status components 11 and 12 have different patterns from others as you can see in Figure 18 in the appendix, in that their statuses flip between 0 and 1 very frequently.

In contrast, the rest have fewer changes and are more biased. Table 7 provided in Appendix X.II provides a comprehensive overview of various statistics.

3.2.3 Timestamps Analysis

In the field of time series analysis, the timestamp itself is an important feature and contains valuable information for data analysts to discover. Since we are working with telemetry data collected from sensors on a satellite, we must understand the timestamps' temporal characteristics.

Among the 44 component datasets (19 signature components as features and the 25 status components as labels), we have 24 of them with exactly 451,754 matching timestamps, and the rest 20 have lengths around 4,094,070. A detailed head count can be found in Table 2. Although the length of the longer dataset varies, the timestamps at the very beginning and the very end are identical throughout all the longer datasets, meaning that they cover the same time range. If we compare the timestamps from the shorter and longer datasets, one can spot that the sensor for shorter datasets started collecting data around 20s later and finished collecting data around 40s earlier than the sensors for bigger datasets. This mismatch is relatively negligible considering the timespan of the dataset, which is 13 months. These specific features from satellite data require us to do the preprocessing in the next section.

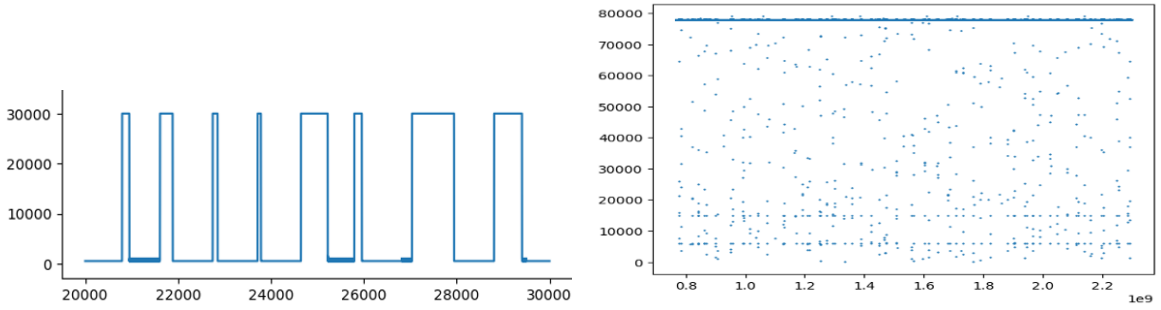
	Signature files				
No. of entries	451754	4094068	4094070	4094073	4094077
Files of this size	13	2	1	2	1

	Status files			
No. of entries	451754	4094066	4094069	4094073
Files of this size	11	12	1	1

Table 2: Overview on the file size

The timestamps are encoded in Unix Epoch Time(UET), the total number of milliseconds after 1 January, 1970 at 00:00:00 UTC. The timestamps do not correspond to the actual time when the satellite was functioning. A temporal offset unknown to us was manually added to the timestamp by DLR, for the sake of data privacy.

After converting the timestamps from UET format to real-time, it shows that the time range of the dataset starts from 5 December 2015, and ends on 14 January 2017, ranging around 13 months. One thing worth mentioning is that the data are sampled under varying sampling rates, which means that the time series is not equally distributed time-wise. The sampling rate is calculated by subtracting the previous timestamp from the current timestamp. The most common time gap between two timestamps in the longer time series (around 4 million rows) is 500ms, followed by 30,000ms and finally 1,000ms, see Figure 5a. (Some are a few milliseconds shorter, others a few milliseconds longer.) The sampling rate of the shorter dataset with exactly 451,754 rows is also changing all the time, while the most dominant value lies somewhere around 78,000ms, as shown in Figure 5b.



(a) Longer time series (b) Shorter time series

Figure 5: Temporal distribution of sampling rate

The uneven distribution on time due to multi-rate sampling is something we have to work on. We decided to tackle this in two ways, one would require time-wise resampling, and the other one does not require resampling but only works on special models.

3.2.4 Eclipse Cycle

One feature among the 19 signature components (signature component 19) contains the flag information on whether an eclipse takes place. One complete cycle of flag switching from 0 (no eclipse) to 1 (eclipse) is around 90 minutes, corresponding to the satellite orbit as seen in Figure 6. This allows us to look into the dataset from a cyclic point of view and also helps with further dataset reshaping, because we can use the eclipse cycle as the third dimension in the 3D array.

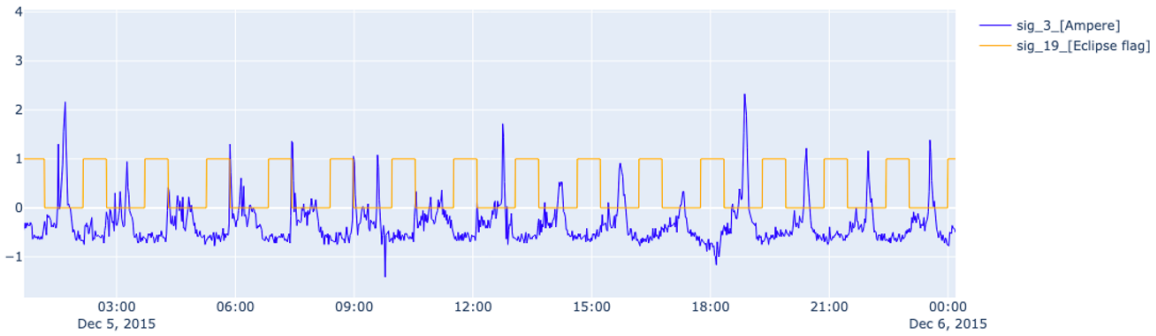


Figure 6: Local peaks of signature component 3 compared with eclipse signal

4 Data Preprocessing

4.1 Data Cleansing

Our objective in the data cleaning phase was to take the individual raw files and transform them into one cohesive dataset with matching input and output shapes. Cleaning is essential in all real-world data solutions, yet it is surprisingly sparsely documented because most of the data used in research is already clean. Hence, this is an essential contribution to our project to precisely document the steps needed to pre-process our data, which can also help outline the method for other data. Starting from the raw individual files, in chronological order, these are the steps:

1. Merge the individual files on the timestamp column into inputs (signatures) and outputs (statuses)
2. Drop all columns that are "non-useful", i.e. they have a constant value of 0 throughout (usually this would relate to a backup sensor or component that is not used throughout the duration of this data collection)
3. Drop duplicate timestamps and keep only the latest values; these duplicates arise due to post-processing the data at the ground station which causes rounding off the timestamps to a certain degree
4. For the outputs (statuses), booleanize the different string representations into 0 or 1 - in our case [OFF, PL_DEACTIVATED, NONE] maps to 0, while [ON, PL_MEASURE, PL_OPERABLE, PWR1] maps to 1
5. Interpolate the missing values - since the individual files have different sampling rates, merging them will cause a lot of missing values to be present due to mis-

matching timestamps; for the continuous variables we use linear interpolation, and for the categorical variables we use "nearest" interpolation
6. Drop any remaining missing values that could not be interpolated since they are either at the beginning or at the end
7. Standardize the input values (subtract the mean, then divide by the standard deviation)

As a result, we have unified inputs and outputs with no missing values and matching shapes. After cleaning, we are left with 14 inputs (signatures) and 15 outputs (statuses), both with 4482157 samples with matching timestamps. This dataset forms the first cohesive basis for our model preparation. This entire process was done automatically using a Python script, which could be reused for other similarly structured satellite data.

4.2 Feature Engineering

When talking about the input to the models we use, time series data is always considered as a 3D tensor with dimensions:

- number of samples
- number of features (aka variables, dimensions, channels)
- number of steps (or length, time steps, sequence steps)

We have explored different ways of reshaping the data that address slightly different problem statements and offers varied perspectives and usability.

4.2.1 Batch Samples Individually

In this method, we consider each timestamp or recorded data point to be one sample (or one time sequence). Then the dimensions of the input 3D tensor are (4482157 samples, 14 features, 1 step), where the features are the sensor values that contribute to a component's power signature.
Since each timestamp is treated individually by the models, this also allows us to predict which components are ON or OFF at any given timestamp. If we are able to do this with a high enough accuracy, this should prove to be the most useful information for system engineers who might rely on our model to analyze a satellite's components.

4.2.2 Batch on Eclipse Cycle

Another way of looking at the task would be to consider a certain duration rather than a specific timestamp, and measure if a component switches its status in this duration (i.e., flips from ON to OFF, or conversely OFF to ON). Rather than take any arbitrary duration, we look at eclipse cycles - an essential part of a satellite's routine. An eclipse cycle consists of two consecutive periods of sunlight and shadow for a satellite, similar to a day for us on Earth. Fortunately we have information from a Sun sensor in our data, which tells us whether the satellite is in eclipse, see Figure 1b.

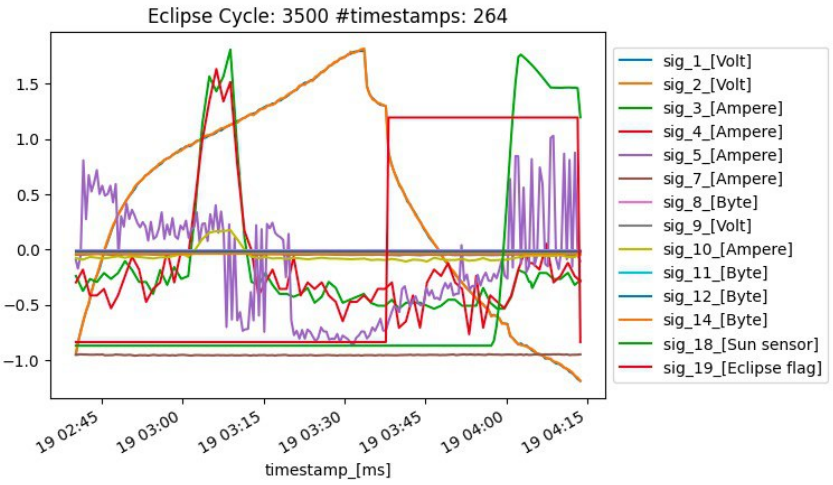


Figure 7: A resampled batch with all the signatures (features) spanning one eclipse cycle

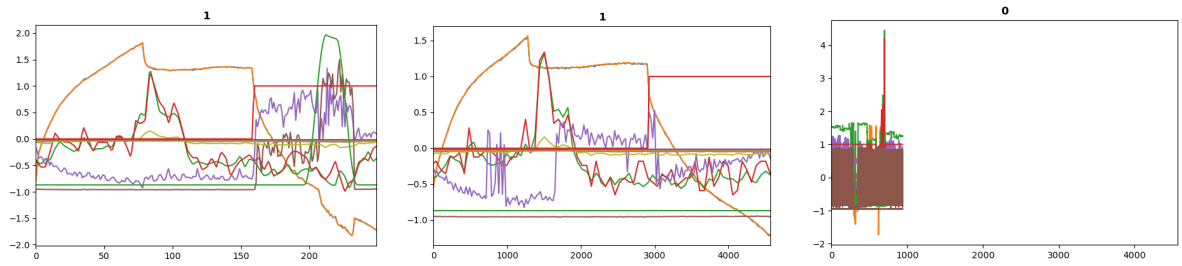
Using this eclipse cycle as the third dimension of our input 3D tensor also gives rise to the problem of varying time sequence lengths. Each eclipse cycle is roughly 90 minutes in duration. Since our tensor needs to be of a uniform shape, there are a few ways in which we can tackle this problem, discussed in the next section. The resultant dimensions of the data are (6268 samples, 14 features, 250 steps).
The number of steps as 250 was chosen due to a good trade-off between being computationally inexpensive when running models and not losing too much information.

4.2.3 Resampling Signature Data

As mentioned above, eclipse cycles are taken as a frame of reference instead of just individual timestamps. In order to do so, the data must be grouped by eclipse cycle. Due to the multiple sampling rates in the data, each eclipse cycle consists of a varying number of data points, between 50 to 4500, with 715 being the mean. Since the input 3D tensor requires a fixed length, there are a few possible ways to reshape the inputs:

1. Resample and downsize data to a fixed, uniform frequency
2. Resample and upsize data to the maximum frequency
3. Pad all eclipse cycles with empty values to reach the maximum frequency

Method 1 is very efficient in making the data easier to deal with by downsizing, although it might aggregate some critical information that is then lost. Method 2 would represent the data well, but it is very computationally expensive and needs too much memory for us to be able to implement it. A similar problem arises with method 3, in which we try to retain the original, inherent mixed sampling rates in the data and assume that the model can deal with the empty values - it simply requires too much memory. As such, only method 1 was implemented, although with much success, as we will show later.Thus, the signatures (or features) are grouped by eclipse cycle and then resampled to a fixed frequency of 250, using linear interpolation for the continuous variables and “nearest” interpolation for the discrete variables.



(a) (b) (c)

Figure 8: (a) Resampled batch with 250 data points per eclipse cycle; (b) Resampled batch with 4571 (maximum) data points per eclipse cycle; (c) Padded batch showing empty values added at the end of data

4.2.4 Reshaping Status Data

Now that the features have been reshaped, we also need to reshape the labels accordingly. For each eclipse cycle, we should have one label. However, how do we aggregate the labels? This, by definition, changes the problem statement from "Can we know which components are ON or OFF at any given time?" to "What can we know about the components in a given eclipse cycle?".

Our answer was to observe whether a component's status changes in an eclipse cycle. If it does change (from ON to OFF, or OFF to ON), we assign True/1. If it stays the same, we assign False/0. While this fundamentally represents different information, it might still be helpful for system engineers to know. For example, in the error detection task, it can help narrow down the zone of inspection to one eclipse cycle (approx. 90 mins in duration).
Hence, we have 6268 labels, each representing whether or not a component's status has changed in that eclipse cycle. This means it remains a classification problem, and we reframe our data to run on the same models.

4.3 Multi-class Labels

Initially, we considered performing binary classification for each component. This means having 15 models for 15 different components, all of which would have to be newly trained every time these methods are applied to a new satellite project. This is an inconvenient process for system engineers who would have to run a different model whenever they want to know the status of a single component. On top of monitoring satellites simultaneously, it can become cumbersome. A much more manageable approach would be to have a few multi-class classification models that deal with multiple components.

Looking at 18, we can recognize two distinct types of components. Components 11 & 12 have frequently changing status throughout the data, while the other components seldom show spikes or trenches. From this eye test, we can discern that it might be worthwhile to build 2 multi-class models: one for components 11 & 12 which are frequently changing, and one for the rest of the components which seldom change (referred to hereafter as the

"other components").
The process for building these multi-class models is very standard - we take all possible combinations of the binary labels of components and assign them to a class. For example, in the case of components 11 & 12, we would have 4 classes: 00, 01, 10 and 11, where the first digit represents the status of component 11 and the second digit represents the status of component 12. Then we perform classification based on these 4 classes.

4.4 Train-Test-Validation Split

We split the dataset into training, validation and testing sets as standard in a 70%, 20%, and 10% fashion. The split was performed without shuffling to maintain the chronological order. However, stratification was used to have a balanced representation of the data across the different sets. Due to the stratification, any classes with a frequency of occurrence less than 10 would not have a single sample in the testing set (10% of anything less than 10 is less than 1). Hence, such classes were also removed from the data.

4.5 Removing Bias

When considering the 2 multi-class models as mentioned above, it is interesting to note their respective class distributions.

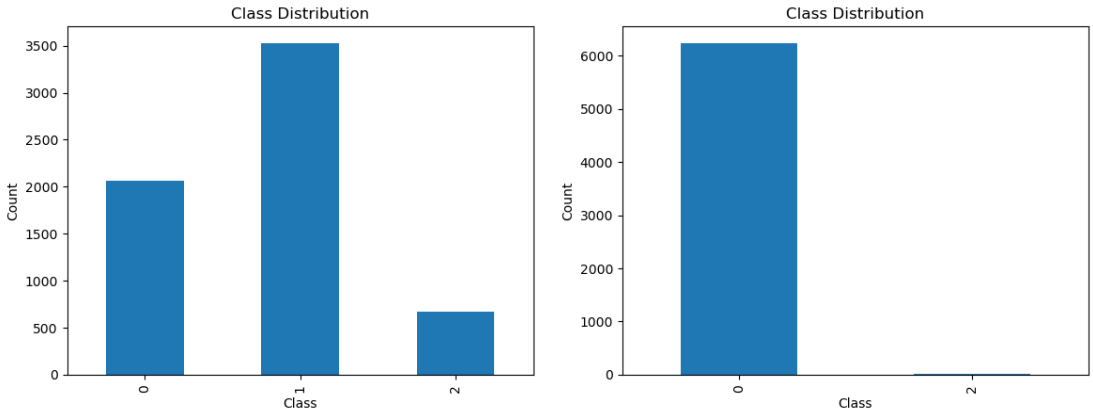


Figure 9: Class distribution for: (left) classes 11 & 12, (right) the other classes

One can see in the above figure how the left image looks much more balanced than the middle one. Since components other than 11 & 12 are OFF most of the time and rarely change their values, the class distribution is skewed. Nevertheless, these components, which seldom change their status, are likely of higher interest to system engineers - they may represent more specialized structures like cameras or other payload. Whereas the frequently changing ones (status 11 & 12 in our case) might represent reaction wheels or solar panels, which are supposed to be used very often.

Hence, we tried to remove some of the bias in the class distribution for the components other than 11 & 12. We reframe the problem slightly - instead of detecting a change in the status, we see whether a component has been ON at least once in the duration of an eclipse cycle.

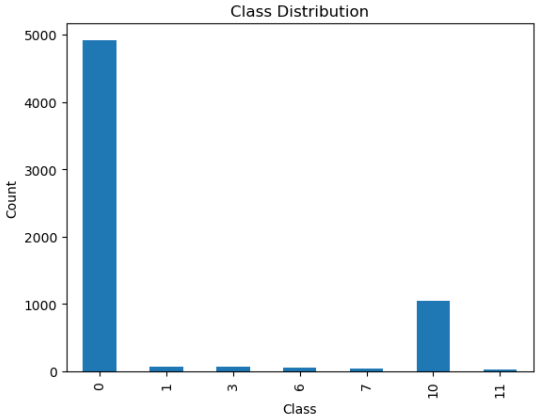


Figure 10: Class distribution for the other classes (batched on eclipse cycles), but instead of detecting a status flip, checking whether the value is at least ON once

The distribution looks slightly more balanced in the rightmost figure. Many more classes appear in the graph due to reframing the problem statement and changing the labeling function. Yet as we will show later, this data might still be too biased for us to make accurate and reliable predictions.

Throughout this section, we have discussed the various points of interest in preparing our data - the bulk of many real-world data science tasks. Now we will look at the modeling side, what methods we used, and how we implemented them.

5 Modeling

The section will introduce the metrics utilized for algorithm comparison and selection, followed by a detailed explanation of the theoretical background of chosen methods and implementation process.

5.1 Selection of Algorithms and Methodology

To address the given problem domain, the most relevant evaluation criteria are:

- Computational Expense (Comp. exp.): The time and resources required to run and train the model, particularly important in this task due to the large-scale dataset. Additionally, the availability of only student laptops for training, and as the DLR uses no cloud infrastructure at their control center for security reasons, limited run-time resources are available.
- Perceived Accuracy (PA): Assessment of the model's predictive accuracy. Accurate predictions are essential for making informed decisions regarding the components of the satellite based on the model's output.
- Difficulty of implementation (DOI): Considering the limited time available for the project, the level of complexity and challenges associated with executing the separate models must be carefully evaluated.

- Compatibility (Comp.): An examination of the extent to which different models align with the given project requirements and constraints is particularly crucial. The suitability of the models to the project's specific objectives and limitations needs to be thoroughly assessed.

- Interpretability (Interp.): Model's capacity to be understood and its predictions to be explained in a comprehensible manner. A clear understanding of the model's inner workings helps us gain insights into its decision-making process and facilitates its application in real-world scenarios.

In order to conduct a selection of suitable models for implementation, approximate estimations based on the corresponding paper named below are made for the listed criteria. Subsequently, after implementation, the generated values for these evaluation criteria are compared to determine the best-fitting model for the given dataset. Based on novel work presented in Section 2, Hive-COTE 2.0 [3; 5], ResNet [9] and Rocket [6] are compared in Table 9 as baseline models.

In the assessment of the distinct evaluation criteria in Table 9, we would like to emphasize the significant divergence of values for decision criteria Comp. exp. and PA. On the other hand, the values of DOI, Comp., and Interp. demonstrate a high degree of similarity. Our assessment of these individual values is primarily based on extensive literature research [3; 5; 9; 6]. For instance, determining the computational expense draws upon findings presented in Figure 19. Due to computational limitations within this project, the criterion of computational expense is given added weight in the event of a tie. Consequently, the Rocket model emerges as the most suitable baseline model in Table 9.

By comparing Multi-Channel Attention-Based Long Short-Term Memory with Fully-Convolutional Network (MALSTM-FCN) [19], Transformer Architecture for Time Series Data (TST) [21], CNN [24], Shapelet Transform (ST) [7] and InceptionTime (IT) [3] in Table 10, a choice for advanced models can be made.

Method	Comp. exp.	PA	DOI	Comp.	Interp.	Sum
TST	4	4	4	3	2	15
MALSTM-FCN	3	5	5	2	2	15
CNN	2	4	1	5	2	12
ST	3	5	2	4	5	14
IT	5	5	4	4	3	21

Table 3: Selection of advanced approaches based on assessment of evaluation criteria on a scale from 0 to 5; with 0 being worst and 5 being best

Regarding Table 10, we highlight the values of near accuracy, while the values of all other sections are widely spread. Moreover, TST, MALSTM-FCN, IT are the three models with the highest score in Table 10, whose implementation will be presented below.

To implement these four models, the time series AI (tsai) package [25] offers a comprehensive set of tools and functionalities specifically designed for time series analysis and

modeling, which is a suitable choice for effectively implementing these four models for the given TS dataset.

5.2 Implementation

With the help of the tsai package, we were able to develop a common framework of execution for our models. The following steps and their detailed descriptions illustrate the order of execution in this framework:

- **Define batch size & create dataloader**: The batch size refers to the number of samples that will be propagated through the model at once. The smaller the batch size, the lower the training speed, while larger batch sizes require more memory. We define the batch size as **64**, which worked well across our different student laptops with limited memory. Then, a dataloader is created to efficiently load and process the data in batches during training.

- **Define the model architecture**: The model architecture refers to the specific structure of the machine learning model being implemented. It involves defining the layers, connections, and operations that make up the model. This step includes choosing the appropriate model (from **TST, MALSTM-FCN, ROCKET and IT**) and configuring its parameters (e.g., number of layers, hidden units). We used the default architectures provided by the authors of the different papers, without adding any tweaks of our own.

  The loss function is also defined, quantifying the error or mismatch between the predicted output and the ground truth labels. The choice of loss function depends on the specific task and the type of data being used. We used **label smoothing cross-entropy** loss.

  Hyperparameters are parameters that are set before the training process begins and are not learned from the data. They control aspects such as learning rate, regularization strength, and dropout rate. Due to limited time and resources, we did not do extensive tuning but stuck with some values that worked well for us.

- **Find good options for learning rate**: The learning rate determines the step size at each iteration during the optimization process. It is a critical hyperparameter that affects the speed and quality of convergence during training. We use the learning rate finder function from the tsai package to find points where the loss is decreasing. Learning rates in the range of 1×10^{-4} to 1×10^{-3} served us well.

- **Train the model on a defined number of epochs**: Training the model involves feeding the training data through the model for a specific number of epochs or iterations. Each epoch consists of a forward pass (where inputs are fed through the model to generate predictions) and a backward pass (where the model's parameters are updated based on the computed loss). For the individual timestamps, we set the number of epochs to **50**. For eclipse cycle batches, we found that **15** epochs were enough to achieve stable and sufficient results.

- **Evaluate the model on the testing set**: After training the model, evaluating its performance on unseen data is essential to assess its generalization ability. The testing set, separate from the training set, is used for this purpose. As stated before, our training, validation, and testing set split ratio is 70/20/10, respectively. The model is applied to the testing set, and the predicted outputs are compared to the ground truth labels. A classification report is generated including metrics such as accuracy, precision, recall, and F1-score, which offer a comprehensive evaluation of the model's performance across different classes.

We ran all our models with different data preparation methods and recorded their evaluation metrics. As we show, while they have different degrees of usability, they all provide valuable insights into modeling such satellite data.

6 Results

We first applied the Transformer model (TST) to classify the status label using all 14 signature components to generate some baseline results with a brute-force method.
Status component 11 and 12 were tested since they are less biased. The results are shown in **??**, we measure the performance by accuracy, precision, recall and f1-score. Notably, the naive accuracy for status component 11 and 12 are 0.90 and 0.83, which means that the Transformer model managed to outperform the naive accuracy. Naive accuracy is the proportion of the label that takes up the most cases among all classes. The performance for status 12 is 0.99 in all four metrics, and there is room for improvement in precision, recall, and f1-score for status component 11. Still, we were unsatisfied with this and hoped to have one model that works well for all input signals.

Single component as label	Naïve Accuracy	Metrics	TST
Status component 11	0.90	Accuracy	0.92
		Precision	0.61
		Recall	0.56
		F1-score	0.58
Status component 12	0.83	Accuracy	**0.99**
		Precision	**0.99**
		Recall	**0.99**
		F1-score	**0.99**

We now combine the two status components and have three possible combinations in every timestamp: status 11 OFF and status 12 OFF (Class 0), status 11 ON and status 12 OFF (Class 1), and status 11 OFF and status 12 ON (Class 2). For some reason, the two statuses are never switched on simultaneously. The results are shown in the first row of Table 4. For multiclass classification, we take the weighted average of those metrics. LSTM outperforms the other two models, achieving the highest score in all four metrics.

As mentioned in Section 4.3, we introduced the notion of the eclipse cycle and investigated if one specific status component would change during eclipse cycles. We applied TST,

InceptionTime, and LSTM for this multiclass classification task. The results are shown in the second row of Table 4. InceptionTime outperforms the other two with less training time.

Similarly, we did the same preprocessing to status components other than 11 and 12 and tried to reshape the labels using eclipse cycles and individual timestamps. The results are shown in Table 5. All three selected models' performance is around the same level or worse than naive accuracy when the inputs are individual timestamps. For eclipse cycle implementation, the results are all higher than naive accuracy, while LSTM model scores the highest in all the metrics.

Components 11 & 12 (multi-class)	Naïve Accuracy	Metrics	TST	Rocket	IT	LSTM
Individual timestamps	0.78	Accuracy	0.75	0.88	0.77	**0.94**
		Precision	0.85	0.85	0.84	**0.93**
		Recall	0.75	0.88	0.77	**0.94**
		F1-score	0.79	0.84	0.79	**0.93**
Eclipse cycles (status flips)	0.56	Accuracy	0.83	0.87	**0.95**	0.92
		Precision	0.76	0.85	**0.95**	0.92
		Recall	0.83	0.87	**0.95**	0.92
		F1-score	0.79	0.85	**0.94**	0.91

Table 4: Performance metrics for status 11 and 12

As shown above, we tried to first solve the task from an individual timestamp perspective, meaning that at every timestamp, given the 14 input signature component signals, the model should be able to predict the 15 binary status component signals output. This would be the ideal way to interpret the problem statement and therefore come up with the most useful information for system engineers. However, it might be inconvenient to have individual models for each component and to retrain all of them repeatedly.

This leads us to the reframing of the problem statement. Instead of predicting the value of certain status components at one specific timestamp, we grouped timestamps into an eclipse cycle. We studied the overall behavior of the status component during the cycle. The prediction on eclipse cycle flipping for status 11 and 12 are very accurate and may provide information on important time windows to system engineers or a level of monitoring that is not too detailed nor too detached.

For the other status components, the data we obtained are excessively biased, prohibiting us from coming up with meaningful and usable inferences. We tried to balance the distribution of ON and OFF in the hope that the balanced dataset might allow better performance without success. Moreover, for the eclipse cycles, they are accurate. However, they only tell if a component was at all ON in that eclipse cycle rather than whether it flipped, which provides limited information.

Other components (multi-class)	Naïve Accuracy	Metrics	TST	Rocket	IT	LSTM
Individual timestamps	0.77	Accuracy	**0.78**	0.78	0.72	0.75
		Precision	**0.77**	0.76	0.71	0.75
		Recall	**0.78**	0.78	0.72	0.75
		F1-score	0.69	0.69	0.71	**0.74**
Eclipse cycles (atleast once ON)	0.78	Accuracy	0.84	0.83	0.88	**0.92**
		Precision	0.80	0.79	0.88	**0.93**
		Recall	0.84	0.83	0.88	**0.92**
		F1-score	0.81	0.81	0.88	**0.91**

Table 5: Performance metrics for other status components

7 Conclusions

In this project, a method for processing the present satellite data is developed. The initial mission statement included acceptance criteria that are evaluated in this section. Overall 12/13 required criteria (subsection 1.4) and 1/6 optional criteria (section 1.4) have been fulfilled .

The performance metrics of our model were evaluated extensively, with a primary focus on achieving high accuracy. We accomplished an accuracy rate above 90% on both the validation and test datasets, demonstrating the effectiveness of our approach. Additionally, we achieved precision, recall, and F1 scores of at least 75%, indicating a balanced performance. We diligently monitored and documented our training approaches with extensive plots. Our dataset was split reasonably into a 90/10 ratio, providing sufficient data for training and testing. We compared different approaches using the same preprocessed data, allowing for a fair comparison of their performance. The documentation of our work was presented in a visual manner, including high-level descriptions, schematic overviews of the network architecture, and tables for detailed comparisons. Throughout the code, we included comments to enhance future usage.

Although the project successfully accomplished most required acceptance criteria, there are some optional criteria that have not been fully met, leaving room for further improvement. The project lacked a mechanism to provide a level of confidence or probability for each label prediction. Incorporating such a feature would offer valuable insights into the certainty of the model's predictions. Explainability was not the scope of this project. By focusing on developing a model that is not only highly accurate but also interpretable, stakeholders would gain better insights into the decision-making process. Some option in the this area would be other methods like shapelets [7]. Another area that was not fully explored was evaluating the model's effectiveness on different datasets by ourselves. All the models were carefully chosen and in most of the cases compared against 85 UCR/UAE Datasets and/or multivariate datasets, e.g. [3].

8 Future Work

We feel that our work represents a useful but only an initial incursion into exploring the potential of what can be done with such satellite telemetry data. For instance, during data preparation, we use linear interpolation to fill in the missing values of our data. For some variables the number of missing values far outweigh the number of existing values, and linear interpolation might force the variable's distribution to become linear in nature even when that might not be its true representation. In such cases, there are advanced methods of time series interpolation which may perform better, like Deep Markov Models[26] or LSTM[27]. The resulting data distributions would be closer to their original characteristics and hence offer better quality of data during modeling.

Secondly, we did not perform much hyperparameter tuning of our models. This was due to both time and resource constraints. With access to better compute resources, we may have been able to perform advanced search techniques like Bayesian Optimization [28] for ideal hyperparameters for our models. With additional time, we would have been able to look into each specific model and fine tune it to our benefit, for example, by changing the positional encoding scheme in the Transformer model to one that suits multi-rate timeseries data better, or configuring the architecture of InceptionTime to create our own custom CNN model.

The major stumbling block was the bias in the data. We tried a few different ways of dealing with it to mixed degrees of success. On top of that, the data we worked with was recorded during the peak operational period of the satellite, which means that standard real-world data would be more biased. We feel that domain expertise could be the key to solving this problem. Discussing with system engineers, listening to their experience and insights and taking into account their feedback could be crucial to finding the most worthwhile and useful problem to solve given this data. It would be all too cliché for ML engineers to get caught up in the pursuit of metrics like accuracy and precision, but our lessons in this project have taught us to look at the bigger picture - which we think should also be the mindset for further incremental work.

9 Discussion and Project Reflections

Working with real-world data presented unique challenges and uncertainties that differed from the controlled environments typically encountered in academic studies. At the project's start, DLR supervisors expressed initial doubts about the usability of the data, making it necessary to navigate the problem statement and devise appropriate methods. This process of exploring and adapting approaches in the absence of a predefined solution provided invaluable learning opportunities. It taught us the importance of problem-solving, adaptability, and the ability to navigate uncharted territories in order to make tangible impacts in real-world applications.

One of the major challenges encountered during this project was the significant bias present in the data. The limited availability of positive cases for training made it difficult

to accurately learn and predict the status values. Despite our efforts, this bias remained a substantial hurdle throughout the project. Preprocessing emerged as the most challenging and time-consuming aspect. The telemetry data required extensive cleaning and normalization. Preserving important timestamps and their corresponding values after resampling proved to be a significant challenge. These steps also aimed to overcome the computational limitations we faced due to restricted access to resources from the Leibniz Supercomputing Centre (German: Leibniz-Rechenzentrum) (LRZ) and DLR, which could have allowed us to explore a few different avenues and enhance model performance. Nonetheless, we made the most of the available resources and diligently worked within the given constraints to achieve meaningful results.

Addressing these preprocessing and computational challenges necessitated careful consideration and experimentation to ensure data integrity and quality. Our team collaborated closely, exchanging ideas, discussing best practices, and supporting each other to overcome these obstacles. This shared commitment and problem-solving mindset were instrumental in devising effective preprocessing strategies, enabling successful modeling and prediction of component status values.

Despite the encountered struggles, our findings yielded several positive outcomes and insights. Surprisingly, the baseline model we developed outperformed the SOTA Transformer model, leaving limited room for improvement within the given scope. This unexpected result prompted a critical examination of our assumptions and exploration of alternative avenues for progress. Reframing the data by considering both single class and multiclass approaches offered valuable insights and potential for the development of specialized models for each component.

In conclusion, despite the numerous challenges faced throughout the project, our investigation into ML approaches for predicting status values based on power signatures in satellite telemetry data provided valuable insights and significant advantages. The project highlighted the importance of addressing biased labels, adapting to computational limitations, and tackling the complexities of preprocessing. Our work establishes a solid foundation for further research and optimization in this domain, offering potential benefits to the DLR and the broader field of satellite technology.

The teamwork within our group played a crucial role in overcoming the encountered challenges. The complexities of the project and the aforementioned obstacles tested the limits of our skills and knowledge. However, our collaborative spirit and shared commitment to success propelled us forward. We fostered an environment of mutual support, engaging in open discussions, brainstorming sessions, and knowledge sharing. This collaborative approach allowed us to leverage our diverse perspectives and expertise to find innovative solutions. Together, we overcame hurdles, celebrated small victories, and collectively pushed the project towards its positive outcome. The resilience and camaraderie exhibited by our team exemplify the strength of collaborative efforts in tackling complex projects.

Bibliography

[1] D. Losa, M. Lovera, R. Drai, T. Dargent, and J. Amalric, "Electric station keeping of geostationary satellites: a differential inclusion approach." IEEE, pp. 7484–7489.

[2] J. Miller, "Sun's up!" 1994. [Online]. Available: https://www.amsat.org/articles/g3ruh/112.html [Accessed: 2023-07-18]

[3] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, pp. 917–963, 7 2019.

[4] A. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version," 2 2016. [Online]. Available: http://arxiv.org/abs/1602.01711

[5] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall, "Hive-cote 2.0: a new meta ensemble for time series classification," 4 2021. [Online]. Available: http://arxiv.org/abs/2104.07551

[6] A. Dempster, F. Petitjean, and G. I. Webb, "Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels," 10 2019. [Online]. Available: http://arxiv.org/abs/1910.13051http://dx.doi.org/10.1007/s10618-020-00701-z

[7] A. Guillaume, C. Vrain, and E. Wael, "Random dilated shapelet transform: A new approach for time series shapelets," 9 2021. [Online]. Available: http://arxiv.org/abs/2109.13514http://dx.doi.org/10.1007/978-3-031-09037-0_53

[8] C. Toth and H. Oberhauser, "Bayesian learning from sequential data using gaussian processes with signature covariances," 6 2019. [Online]. Available: http://arxiv.org/abs/1906.08215

[9] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," 11 2016. [Online]. Available: http://arxiv.org/abs/1611.06455

[10] J. Serrà, S. Pascual, and A. Karatzoglou, "Towards a universal neural network encoder for time series," 5 2018. [Online]. Available: http://arxiv.org/abs/1805.03908

[11] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 3 2016.

[12] A. L. Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," 2016.

[13] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," pp. 298–310, 2014.

[14] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification," *Frontiers of Computer Science*, vol. 10, pp. 96–112, 2 2016.

[15] P. Tanisaro and G. Heidemann, "Time series classification using time warping invariant echo state networks." IEEE, 12 2016, pp. 831–836.

[16] M. Ganesan, R. Lavanya, and M. Nirmala Devi, "Fault detection in satellite power system using convolutional neural network," *Telecommunication Systems*, vol. 76, pp. 505–511, 4 2021.

[17] G. T. Wilson, "Arma models for cyclical data," *IFAC Proceedings Volumes*, vol. 18, pp. 1949–1952, 7 1985. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474667017608541

[18] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "Lstm fully convolutional networks for time series classification," 9 2017.

[19] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate lstm-fcns for time series classification," *Neural Networks*, vol. 116, pp. 237–245, 8 2019.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, "Rethinking the inception architecture for computer vision," 2015.

[21] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning." Association for Computing Machinery, 8 2021, pp. 2114–2124.

[22] Deutsches Luft- und Raumfahrtzentrum (DLR), "Tet." [Online]. Available: https://www.dlr.de/firebird/en/desktopdefault.aspx/tabid-9091/15682_read-38833/ [Accessed: 2023-07-18]

[23] Deutsches Luft- und Raumfahrtzentrum (DLR), "Technologieerprobungsträger tet." [Online]. Available: https://www.dlr.de/rd/en/desktopdefault.aspx/tabid-2274/3396_read-5085/ [Accessed: 2023-07-18]

[24] C. L. Yang, Z. X. Chen, and C. Y. Yang, "Sensor classification using convolutional neural network by encoding multivariate time series as two-dimensional colored images," *Sensors (Switzerland)*, vol. 20, 1 2020.

[25] I. Oguiza, "tsai - a state-of-the-art deep learning library for time series and sequential data," Github, 2022. [Online]. Available: https://github.com/timeseriesAI/tsai [Accessed: 2023-07-18]

[26] Z. Che, S. Purushotham, G. Li, B. Jiang, and Y. Liu, "Hierarchical deep generative models for multi-rate multivariate time series," 2018.

[27] K. Ma and H. Leung, *A Novel LSTM Approach for Asynchronous Multivariate Time Series Prediction*. [Online]. Available: http://www.ieee.org/publications

[28] B. Solnik, D. Golovin, G. Kochanski, J. E. Karro, S. Moitra, and D. Sculley, "Bayesian optimization for a better dessert," in *Proceedings of the 2017 NIPS Workshop on Bayesian Optimization*, December 9, 2017, Long Beach, USA, 2017, the workshop is BayesOpt 2017 NIPS Workshop on Bayesian Optimization December 9, 2017, Long Beach, USA.

Appendix

X.I Signature Values

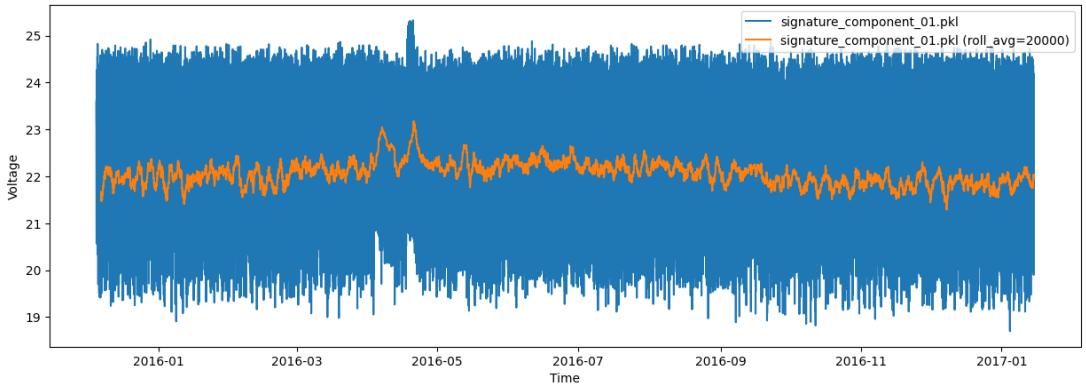


Figure 11: Signature component 1 and its rolling average at 20000 points

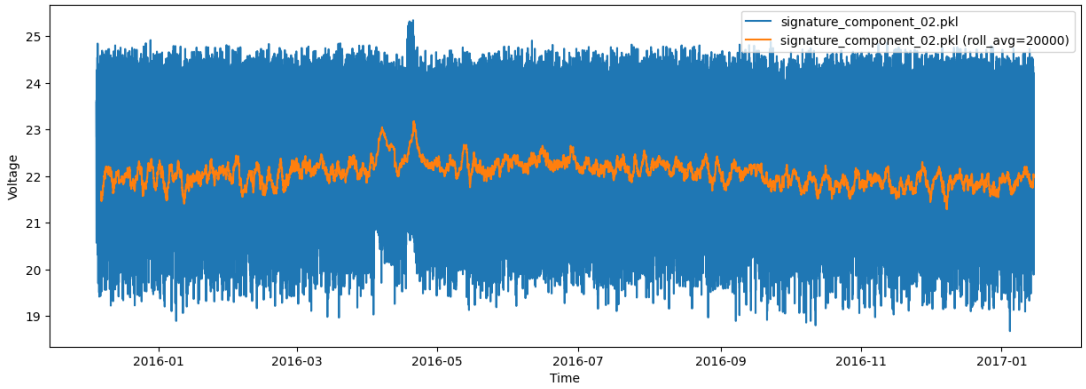


Figure 12: Signature component 2 and its rolling average at 20000 points

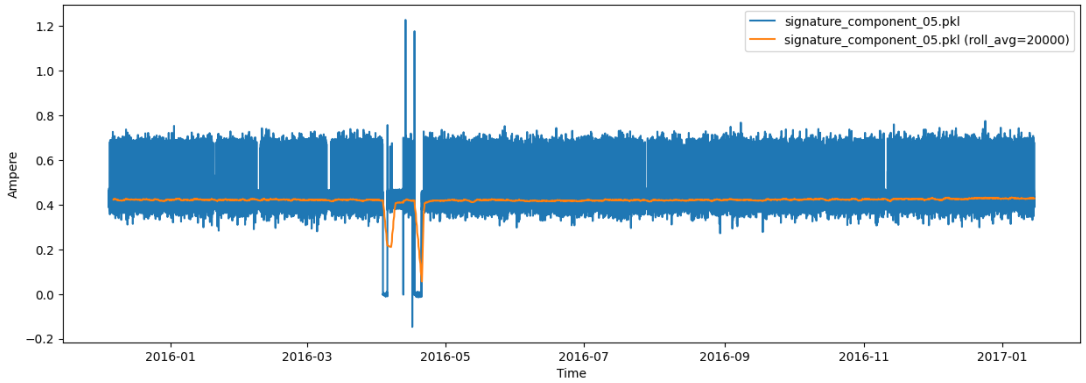


Figure 13: Signature component 5 and its rolling average at 20000 points

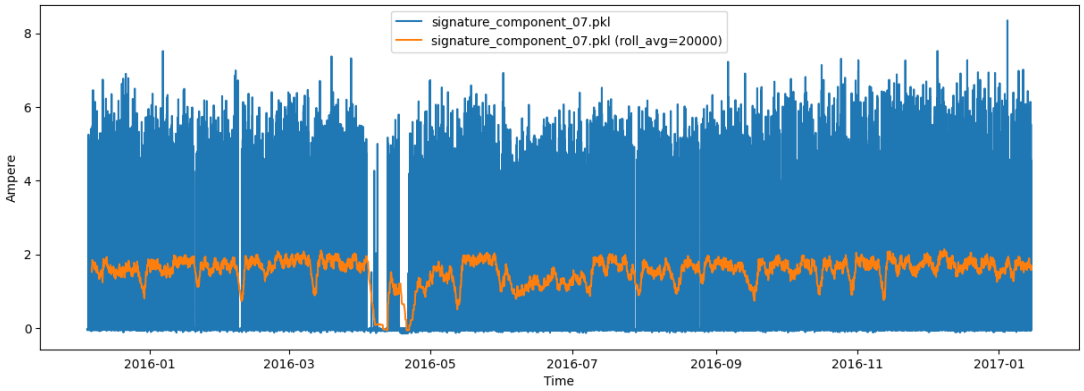


Figure 14: Signature component 7 and its rolling average at 20000 points

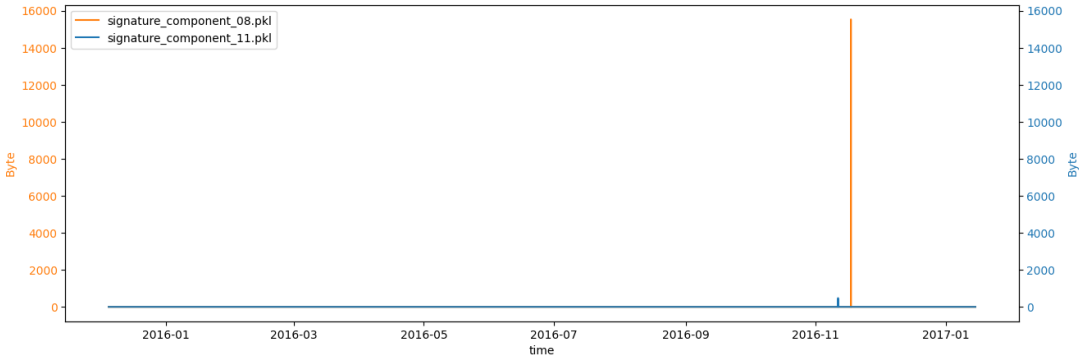


Figure 15: Signature component 8 and 11

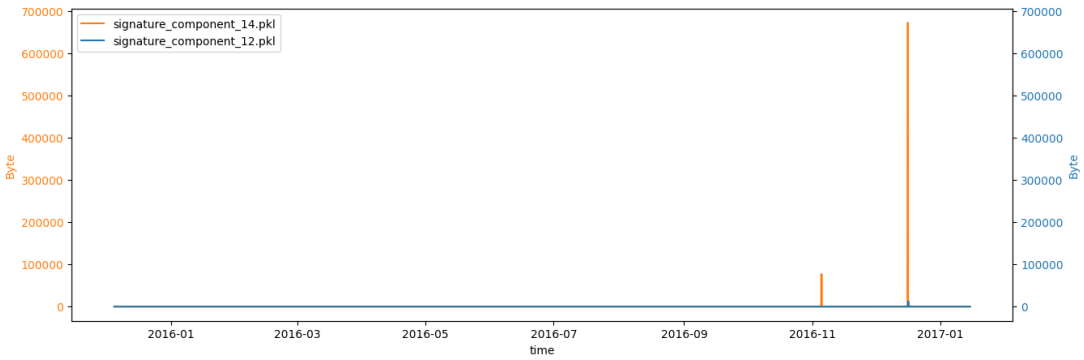


Figure 16: Signature component 12 and 14

	count	mean	std	min	25%	50%	75%	max
timestamp_[ms]	4545878	1.467E+12	1.031E+12	1.449E+12	1.457E+12	1.467E+12	1.476E+12	1.484E+12
sig_1_[Volt]	4094177	22.052783	1.184935	18.69996	21.00601	22.12103	23.08101	25.32744
sig_3_[Volt]	451754	0.016616	0.0055	-0.007013	0.013566	0.014331	0.016769	0.06649
sig_5_[Ampere]	4094177	0.418838	0.041622	-0.147076	0.403127	0.41977	0.436867	1.226653
sig_7_[Ampere]	4094177	1.576964	1.553677	-0.153638	-0.041467	1.656987	3.076604	8.351348
sig_8_[Byte]	451754	0.43954	80.754626	0	0	0	0	15532
sig_9_[Volt]	451754	0.159271	1.530246	-0.054883	0.046651	0.049401	0.055796	24.41017
sig_10_[Ampere]	451754	0.054779	0.202435	-0.049196	0.036183	0.038223	0.04473	5.308893
sig_11_[Byte]	451754	0.053963	5.078529	0	0	0	0	478
sig_12_[Byte]	451754	6.236067	277.51418	0	0	0	0	12356
sig_14_[Byte]	451754	515.409586	17077.1773	0	0	0	0	672640
sig_18_[Sun sensor]	4094177	-0.335416	0.705415	-1	-0.999939	-0.668274	0.40802	1.298767
sig_19_[Eclipse flag]	4094177	0.417596	0.493163	0	0	0	1	1

Table 6: Signature Analysis

X.II Status Values

	count	mean	std	min	25%	50%	75%	max
status_comp_1	4094085	0	0	0	0	0	0	0
status_comp_2	4094085	0	0	0	0	0	0	0
status_comp_3	4094085	0.987947	0.109123	0	1	1	1	1
status_comp_4	4094085	0.008092	0.089592	0	0	0	0	1
status_comp_5	4094085	0.966391	0.18022	0	1	1	1	1
status_comp_6	4094085	0.011181	0.105147	0	0	0	0	1
status_comp_7	4094085	0	0	0	0	0	0	0
status_comp_8	4094085	0.999975	0.004991	0	1	1	1	1
status_comp_9	4094085	0.999998	0.001308	0	1	1	1	1
status_comp_10	4094085	0.999998	0.001398	0	1	1	1	1
status_comp_11	4094085	0.105826	0.307615	0	0	0	0	1
status_comp_12	4094085	0.180594	0.384681	0	0	0	0	1
status_comp_13	4094085	0	0	0	0	0	0	0
status_comp_14	4094085	0.000017	0.004105	0	0	0	0	1
status_comp_15	451754	0	0	0	0	0	0	0
status_comp_16	451754	0.000009	0.002976	0	0	0	0	1
status_comp_17	451754	0.000004	0.002104	0	0	0	0	1
status_comp_18	451754	0.000018	0.004208	0	0	0	0	1
status_comp_20	451754	0	0	0	0	0	0	0
status_comp_21	451754	0.00017	0.013054	0	0	0	0	1
status_comp_22	451754	0.177765	0.382315	0	0	0	0	1
status_comp_23	451754	0	0	0	0	0	0	0
status_comp_24	451754	0	0	0	0	0	0	0
status_comp_25	451754	0	0	0	0	0	0	0
status_comp_26	451754	0	0	0	0	0	0	0

Table 7: Status Analysis

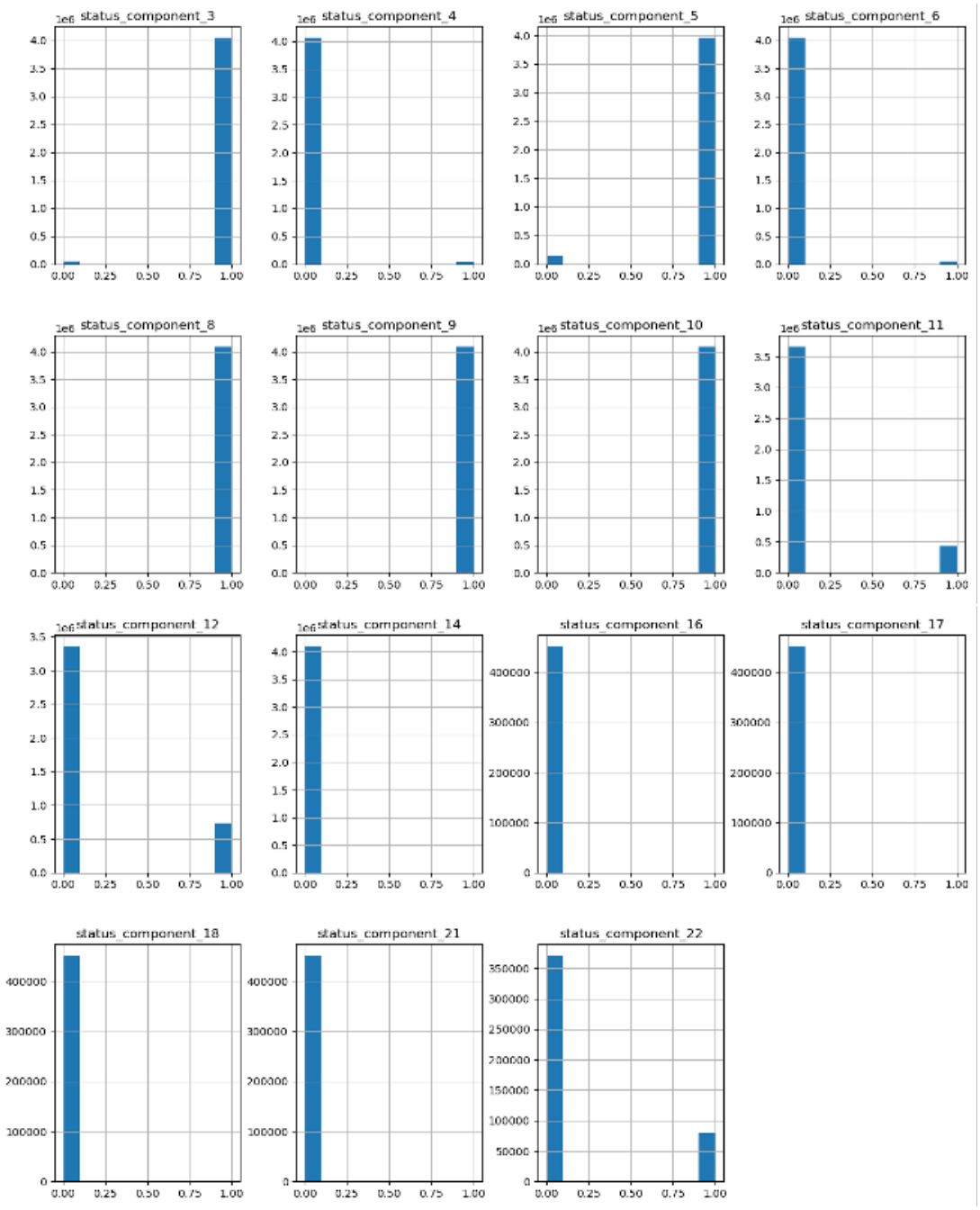


Figure 17: Distribution of 0/1 classes for status components

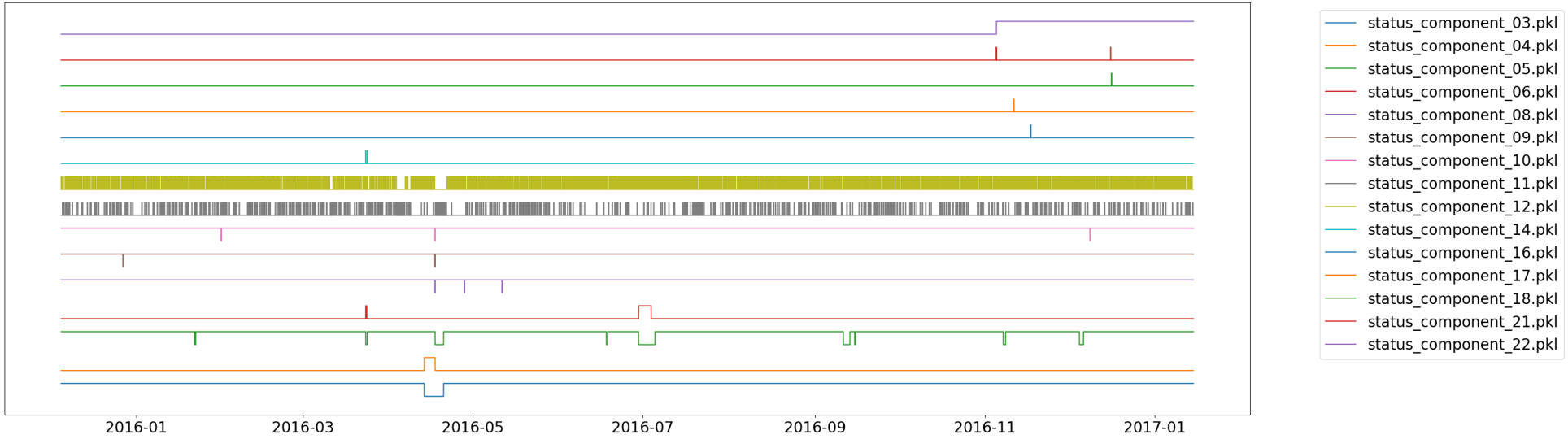


Figure 18: Status components

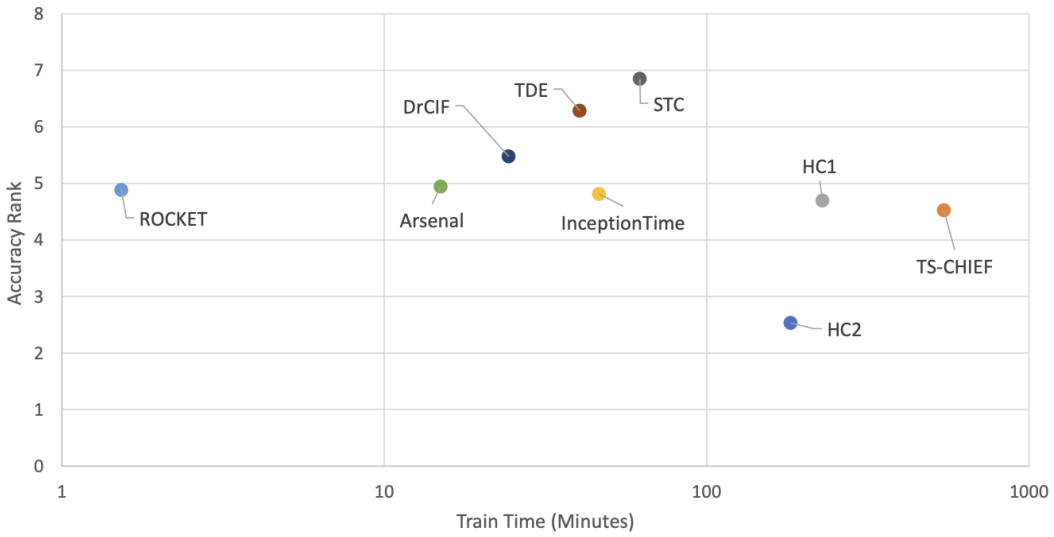


Figure 19: Comparison of computational expense of selected models [3]

Method	Comp. exp.	PA	DOI	Comp.	Interp.	Sum
Hive-COTE 2.0	1	5	4	5	0	15
ResNet	2	2	4	4	1	13
Rocket	5	3	3	4	1	15

Table 8: Selection of baseline model based on assessment of evaluation criteria on a scale from 0 to 5; with 0 being worst and 5 being best

Method	Computational Expense	Perceived Accuracy	Difficulty of implementation	Compatibility	Interpretability	Sum
Baseline Models						
Hive-COTE 2.0	1	5	4	5	0	15
ResNet	2	2	4	4	1	13
Rocket	5	3	3	4	1	15
Advanced Models						
TST	4	4	4	3	2	15
MALSTM-FCN	3	5	5	2	2	15
CNN	2	4	1	5	2	12
ST	3	5	2	4	5	14
IT	5	5	4	4	3	21

Table 9: Selection of baseline model based on assessment of evaluation criteria on a scale from 0 to 5; with 0 being worst and 5 being best

Method	Comp. exp.	PA	DOI	Comp.	Interp.	Sum
TST	4	4	4	3	2	15
MALSTM-FCN	3	5	5	2	2	15
CNN	2	4	1	5	2	12
ST	3	5	2	4	5	14
IT	5	5	4	4	3	21

Table 10: Selection of advanced approaches based on assessment of evaluation criteria on a scale from 0 to 5; with 0 being worst and 5 being best