

# Automated Feature Engineering for Time Series Data

Bachelor's Thesis  
by

**Keyi Li**

Department of Informatics

Responsible Supervisor: Prof. Dr. Michael Beigl

Supervising Staff: Haibin Zhao, M.Sc.  
Yiran Huang, M.Sc.

Project Period: 01/04/2023 – 01/08/2023



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Time series data . . . . .	7
2.2	Automated feature engineering . . . . .	8
2.3	Machine learning model . . . . .	8
2.4	Bayesian optimisation . . . . .	9
<b>3</b>	<b>Pre-processing</b>	<b>11</b>
3.1	Data cleaning . . . . .	11
3.2	Data transformation . . . . .	11
3.3	Sliding window . . . . .	12
<b>4</b>	<b>Technology</b>	<b>15</b>
4.1	Automated feature engineering technology . . . . .	15
4.1.1	TSFRESH . . . . .	15
4.1.2	TSFEL . . . . .	16
4.1.3	KATS . . . . .	17
4.1.4	SEGLEARN . . . . .	17
4.1.5	CESIUM . . . . .	18
4.2	Discussions . . . . .	20
4.2.1	Overfitting . . . . .	21
4.2.2	Curse of dimensionality . . . . .	21
<b>5</b>	<b>Model</b>	<b>23</b>
5.1	$K$ -nearest neighbors . . . . .	23
5.2	Logistic regression . . . . .	24
5.3	Random forest . . . . .	24
5.4	Multilayer perceptron . . . . .	26
5.4.1	Activation function . . . . .	28
5.5	Discussions . . . . .	29
5.5.1	F1-score . . . . .	29
5.5.2	Macro average F1-score . . . . .	29
<b>6</b>	<b>Related work</b>	<b>31</b>
6.1	Novel feature selection methods . . . . .	31
6.2	Model hyperparameter tuning . . . . .	32
<b>7</b>	<b>Programming environments</b>	<b>33</b>

<b>8</b>	<b>Design of user interface</b>	<b>35</b>
8.1	Welcome page . . . . .	35
8.2	Main page . . . . .	36
8.2.1	Step 1: import data . . . . .	37
8.2.2	Step 2: pre-processing . . . . .	38
8.2.3	Step 3 & 4: feature extraction and feature selection . . . . .	39
8.2.4	Step 5: model . . . . .	40
8.2.5	Step 6: optimizer . . . . .	41
8.3	Navigation bar . . . . .	41
8.4	Datasets page . . . . .	42
8.5	Time series features document page . . . . .	43
8.6	User guide page . . . . .	44
8.6.1	User guide panel . . . . .	44
8.6.2	Video panel . . . . .	45
8.7	Results page . . . . .	46
8.7.1	Optimization history panel . . . . .	46
8.7.2	Model hyperparameter importances panel . . . . .	47
8.7.3	Feature importances panel . . . . .	48
8.7.4	Trial stury history panel . . . . .	49
<b>9</b>	<b>Evaluation</b>	<b>51</b>
9.1	Hypothesis . . . . .	51
9.2	Metric . . . . .	51
9.3	Datasets . . . . .	51
9.4	Design of experiments . . . . .	52
9.4.1	Benchmark results . . . . .	53
<b>10</b>	<b>Conclusion and Future Work</b>	<b>57</b>
10.1	Conclusion . . . . .	57
10.2	Future work . . . . .	57
	<b>Bibliography</b>	<b>59</b>

# List of Figures

1.1	Framework building flow chart. . . . .	6
3.1	A sketch of sliding window on time series data with a window size of 20 and a window stride of 10. . . . .	12
5.1	A simple $k$ NN-model with $k = 2$ . . . . .	23
5.2	Logistic regression with iris dataset. . . . .	25
5.3	Conceptual framework of random forest classifier [17, 18] . . . . .	25
5.4	The structure of a single perceptron. . . . .	26
5.5	A multilayer feed-forward neural network. . . . .	27
8.1	The animation on the welcome page. . . . .	35
8.2	The main page of feature engineering pipeline. . . . .	36
8.3	The default and customized mode of importing data. . . . .	37
8.4	The default and customized mode of pre-processing. . . . .	38
8.5	The details of the scaler setting and the dialog for the "cross-validation for id" information icon. . . . .	38
8.6	The step of feature extraction and feature selection. . . . .	39
8.7	The overview of the model. . . . .	40
8.8	The original and changed random forest model parameter settings. . . . .	40
8.9	The optimizer of the pipeline. . . . .	41
8.10	The navigation bar of the website. . . . .	41
8.11	The dataset page with basic details about built-in datasets. . . . .	42
8.12	The time series features document page, listing all feature extraction packages for time series. . . . .	43
8.13	The initial panel of the user guide page, offering a comprehensive textual description for each pipeline step. Clicking on a step leads users to its corresponding documentation or functionality. . . . .	44
8.14	An example of user guide panel. . . . .	45

8.15	Video walkthrough from 'import data' to final result viewing on the user guide page. . . . .	45
8.16	Optimization history panel highlighting the peak performance at trial number 1. . . . .	46
8.17	Hyperparameter importances in multi vs. singular model optimisation.	47
8.18	An example of a feature importances panel focusing on the top 20 features. . . . .	48
8.19	Detailed trial study history panel showing specific models, parameters, and feature selection methods for each trial. . . . .	49

# List of Tables

9.1	A summary of the used datasets in the UCR Multivariate Time Series Classification archive. . . . .	52
9.2	Benchmark classification results (in terms of macro average F1-score) for the baseline and combined tuple feature extraction packages (without tsfresh) approaches of each dataset. The (potentially tied) best score achieved for a dataset is in bold. . . . .	54
9.3	Benchmark classification results (in terms of macro average F1-score) for the baseline and combined tuple feature extraction packages (with tsfresh) approaches of each dataset. The (potentially tied) best score achieved for a dataset is in bold. . . . .	56









# Abstract

Feature engineering for time series data, a critical task in data science, involves the transformation or encoding of raw data to create more predictive input features. This paper introduces a novel web framework designed to automate the labor-intensive and expertise-demanding process of time series feature engineering. The framework comprises advanced methods for automated feature extraction and selection, providing a wide range of application possibilities. A Bayesian Optimization strategy is also integrated to identify optimal features and model parameters for specific datasets, thereby enhancing prediction performance. The paper thoroughly explores the framework's design principles and operational procedures, along with validation of its effectiveness across different domains using real-world datasets.



# Zusammenfassung

Die Merkmalskonstruktion von Zeitreihen, eine wichtige Aufgabe in der Datenwissenschaft, beinhaltet die Transformation oder Kodierung von Rohdaten zur Erstellung von vorhersagefähigeren Eingangsmerkmalen. Dieses Papier stellt ein neuartiges Web-Framework vor, das darauf abzielt, den arbeitsintensiven und expertenabhängigen Prozess der Merkmalskonstruktion von Zeitreihen zu automatisieren. Der Rahmen enthält fortschrittliche Methoden zur automatisierten Merkmalsextraktion und -auswahl und bietet eine breite Palette von Anwendungsmöglichkeiten. Eine bayesianische Hyperparameter-Optimierungsstrategie ist ebenfalls integriert, um optimale Merkmale und Modellparameter für spezifische Datensätze zu identifizieren, wodurch die Vorhersageleistung verbessert wird. Das Papier untersucht ausführlich die Designprinzipien und Betriebsverfahren des Frameworks, zusammen mit der Validierung seiner Wirksamkeit in verschiedenen Bereichen mit realen Datensätzen.



# 1. Introduction

Data scientists undergo several stages when analysing and exploring a new dataset. This process commences with data ingestion, selection and preparation, followed by feature extraction and selection, and finally, modelling and generalisation.

As discussed in [1], there are two primary objectives in data analysis:

- Inference: This involves constructing random models that adapt to the data, followed by drawing inferences about the data generation mechanisms based on the structure of these models.
- Prediction: This pertains to the ability to forecast what the responses to future input variables will be.

However, in the case of the most raw dataset, the features provided are often inadequate. Merely training and predicting by directly inputting the raw data into the model cannot achieve a higher level of accuracy. Wind [2] presented an intriguing exposition on the award-winning submissions in numerous Kaggle competitions, underscoring the significance of feature engineering in real-world scenarios.

Feature extraction and selection, often referred to as feature engineering (FE), tend to be the most time-consuming and labour-intensive steps in the data science workflow. It is a complex task that requires the manual identification of intricate patterns in the data, driven by cumulative domain knowledge, and carried out through iterative experimentation and trial-and-error [3]. Moreover, the feature engineering conducted for different datasets is frequently unscalable, posing further challenges in the process.

The fundamental building blocks of FE approach are transformation functions, capable of being applied to single or multiple attributes within a dataset to create novel features, and robust selection methodologies that allow for the curation of the newly generated features, ensuring their suitability for the targeted model.

In this paper, we propose a generic and scalable web-based framework that attempts to address the above mentioned goals through automated FE for time series data to streamline data science workflows and improve the accuracy and efficiency of inference and prediction tasks.

The workflow (see figure 1.1) can be summarized as follows:

1. Firstly, we have developed an extensible backend, supported by Python, to enable efficient data preprocessing, feature engineering, and model selection for the given dataset.
2. Secondly, we have curated a collection of feature extraction, feature selection, and modeling algorithms from reputable open-source platforms such as Ts-fresh, Tsfel, Seglearn, Kats, Sklearn and so on.
3. Then, due to the specialized and intricate nature of backend programming, we have constructed a frontend framework using React. This approach leverages a user-friendly interface, simplifying the process of automated FE.
4. And finally, we have evaluated our web framework using 15 real-world datasets sourced from diverse domains, showcasing significant performance improvements achieved within a limited timeframe.

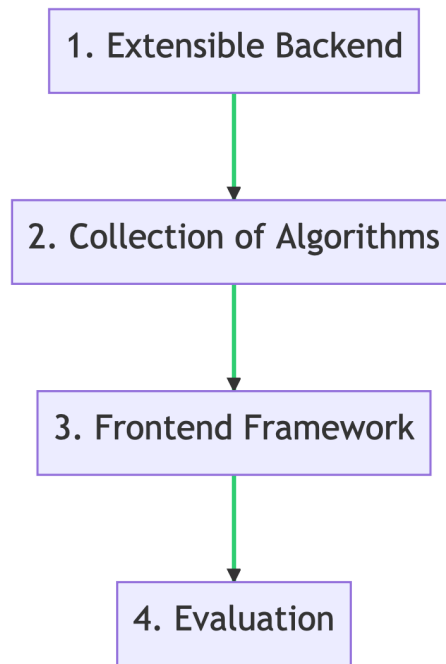


Figure 1.1: Framework building flow chart.



## 2. Background

In this chapter, we first discuss time series data, a typical representation of information that is inherently dependent on its temporal context. We then explore automated feature engineering, which revolutionizes traditional manual processes of feature extraction, contributing to more predictive models.

Moreover, we discuss the essential elements of data modeling, underscoring its role as the mathematical representation of real-world phenomena and its impact on prediction and classification tasks. Lastly, we introduce Bayesian Optimisation, an efficient technique for model tuning, enhancing both accuracy and applicability to new data. These fundamentals provide the necessary background for the methods and tools discussed later in the paper.

### 2.1 Time series data

A time series is a sequence of observations taken sequentially in time [4]. In order to use a set of time series

$$D = \{\chi_i\}_{i=1}^N$$

as input for supervised machine learning algorithms, each time series  $\vec{\chi}_i$  needs to be mapped into a well-defined feature space with problem specific dimensionality  $M$  and feature vector  $\vec{\chi}_i = (\chi_{i,1}, \chi_{i,2}, \dots, \chi_{i,M})$ . In principle, one might decide to map the time series of set  $\mathcal{D}$  into a design matrix of  $N$  rows and  $M$  columns by choosing  $M$  data points from each time series  $\chi_i$  as elements of feature vector  $\vec{\chi}_i$  [5].

In other words, the aforementioned description can be represented using the following matrix format:

$$\begin{pmatrix} \chi_{0,1} & \chi_{0,2} & \chi_{0,3} & \dots & \chi_{0,M} \\ \chi_{1,1} & \chi_{1,2} & \chi_{1,3} & \dots & \chi_{1,M} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \chi_{N,1} & \chi_{N,2} & \chi_{N,3} & \dots & \chi_{N,M} \end{pmatrix}.$$

Furthermore, the total number of such two-dimensional matrices equals the overall quantity of data points present in the dataset. That is, if a trivial time series

dataset is converted into a matrix form it should be a three-dimensional matrix  $A \in \mathbb{R}^{\mathbf{S} \times \mathbf{N} \times \mathbf{M}}$ , where:

- The first dimension (**S**) represents the number of datasets, indicating the count of individual samples.
- The second dimension (**N**) represents the window size of the time series, denoting the number of time steps within each sample.
- The third dimension (**M**) represents the feature dimension, indicating the number of features at each time step.

Such a three-dimensional matrix  $A$  provides a convenient representation, integrating time series data into a unified structure, facilitating subsequent data processing and analysis. Each sample can be accessed by its index (**S**, **N**, **M**) to retrieve specific feature values at a particular time step.

## 2.2 Automated feature engineering

The emergence of Automated Machine Learning (AutoML) has underscored the critical goal of simplifying and automating feature engineering. AutoML is devoted to crafting algorithms and models designed to autonomously carry out tasks that have historically demanded significant human intervention [6].

Automated FE, as a vital component of AutoML, refers to an auto-generative process that extracts and creates novel features from raw data to elevate machine learning model performance. The potential of this approach to drastically cut down the time and specialized knowledge required for feature engineering, thereby expediting the evolution of machine learning models, has garnered substantial scholarly attention in recent years.

Despite the significant strides made in the realm of automated FE recently, several challenges persist. These encompass handling high-dimensional data, ensuring the explainability of generated features, and ascertaining the most effective combinations of features.

## 2.3 Machine learning model

In automated FE, models play a key role in tasks and accelerate the data science process. By utilizing models in automated FE, prediction performance can be significantly improved. Models capture complex patterns and trends in time series data, enabling more accurate predictions.

In our framework, we have integrated four models from different domains, namely  $k$ -nearest neighbors, logistic regression, random forest, and finally the multilayer-perceptron. These models represent diverse algorithms and concepts, providing our framework with a comprehensive and diversified predictive capability.

Further detailed discussions and explorations regarding these models will be presented in chapter 5.

## 2.4 Bayesian optimisation

The model of our framework is built on scikit-learn and integrates hyperparameter Optimisation based on a range of parameters chosen by the user. When the user chooses a wide range of parameters, it can be very time consuming to use brute force to obtain the best results. Therefore, we introduce Bayesian optimisation as an alternative method.

Bayesian Optimisation derives from Bayes' theorem [7]. Given evidence data  $E$ , the posterior probability  $P(\mathcal{M}|E)$  of a model  $\mathcal{M}$  is proportional to the likelihood  $P(E|\mathcal{M})$  of observing  $E$  given model  $\mathcal{M}$ , multiplied by the prior probability of  $P(\mathcal{M})$  [8]:

$$P(\mathcal{M}|E) \propto P(E|\mathcal{M})P(\mathcal{M}). \quad (2.1)$$

In our framework, the optimisation objective is to find the best combination of model parameters, which yields the highest score in the context of supervised learning, i.e.,

$$\hat{p} = \arg \max_{p \in S} f(p),$$

where  $S$  denotes the search parameter space of parameter  $p$ . This formula 2.1 is used to update the probability distributions of different model parameter combinations, enabling the most promising parameter combinations to be selected for the next step in the evaluation. Through iterative updates, the focus is narrowed to parameter combinations with higher probabilities, ultimately leading to the discovery of the optimum.

In other words, by applying Bayesian Optimisation, we are able to search for local optima within a defined range of trial numbers. It allows us to systematically explore the parameter space and discover the best combination of model parameters for a specific dataset and problem.

With Optuna [9], our framework is capable of performing hyperparameter optimisation for models. The following is a pseudocode:

---

### Algorithm 1 Bayesian Optimisation with Optuna

---

**Require:** Dataset  $\mathcal{D}$ , Model  $\mathcal{M}$ , Parameter ranges  $\mathcal{P}$ , Optimisation rounds  $T$

**Ensure:** Optimal parameter set  $\hat{p}$

- 1: Create a new Optuna study
  - 2: Define the objective function:  $\hat{p} = \arg \max_{p \in \mathcal{P}} f(p)$
  - 3: Set  $f(p)$  to a specific model evaluation metric
  - 4: **for**  $t = 0$  to  $T$  **do**
  - 5:     Generate a new trial with the study
  - 6:     Optimize the parameter set  $p$  from  $\mathcal{P}$  using Bayesian Optimisation
  - 7:     Set the model parameters to  $p$
  - 8:     Train the model  $\mathcal{M}$  on the dataset  $\mathcal{D}$  using the parameter set  $p$
  - 9:     Inform the selection of the next trial based on the objective function value  $f(p)$  and its corresponding parameter set  $p$
  - 10: Extract the best parameter set  $\hat{p}$  from the study within  $T$
  - 11: **return**  $\hat{p}$
-



## 3. Pre-processing

Data pre-processing is an important step in the data analysis and machine learning workflow. It involves cleaning, transforming and sliding windows of raw time series data to bring in models for training and analysis more efficiently. Each of these steps will be described in detail in the next few sections.

### 3.1 Data cleaning

Data cleaning is usually the first step in pre-processing, which deals with errors and missing values in the raw data.

Since our framework is designed to facilitate automated processes, it is important to validate the integrity of newly acquired datasets. This includes checking for the presence of `NaN` or missing values. If in that case we choose to replace them with average values. In addition, we would like to have direct access to the dimensions of the dataset as well as the target columns in order to generate anonymised column names for the data import.

We use the `read_csv`<sup>1</sup> function from the pandas library to load the dataset. Currently, we can read datasets in `.txt`, `.csv`, and `.data` formats.

### 3.2 Data transformation

Once the dataset has been cleaned, the next step is data transformation. Raw data often contain noise and outliers, which can negatively impact the performance of the model, leading to inaccurate predictions or classifications. Data transformation, a significant step in data preprocessing, helps to mitigate these disruptions. Our framework provides normalization and min-max scaling techniques for data transformation. These methods adjust the scales of the features to a uniform numerical range, thereby reducing the disturbance caused by variations in the scales of different features.

---

<sup>1</sup>[https://pandas.pydata.org/pandas-docs/version/1.5/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/version/1.5/reference/api/pandas.read_csv.html)

### 3.3 Sliding window

Sliding window is a temporary approximation over the actual value of the time series data [10]. In the realm of time series data analysis, the sliding window process is typically characterized by two crucial factors: the window size and window stride. These are often determined by a specific sampling rate (commonly expressed in Hz) and sampling duration (measured in seconds). For example, in the context of a 20Hz dataset, we receive 20 data samples each second. Hence, in light of the dataset's characteristics and the objectives of the analysis, selecting a window size of 20 (samples/second) x 5 (seconds) = 100 would be an appropriate choice.

As shown in the figure 3.1, we constructed a dataset containing 80 data points. By setting the window size to 20 and the step size to 10, we can divide the data set into 7 windows. In each window, various statistical operations can be performed on the data points in that window, such as calculating the mean, median, and other measures.



Figure 3.1: A sketch of sliding window on time series data with a window size of 20 and a window stride of 10.

In our framework, we use the `sliding_window_view`<sup>2</sup> method from NumPy to process sliding windows over the time series dataset. The code 3.1 implementation is as follows:

Listing 3.1: Sliding window algorithm

```

1 from numpy.lib.stride_tricks import sliding_window_view
2 from scipy import stats
3 def sliding_window(df, segment_length, step_distance, sensor_cols,
4   target):
5     """
6     The returned array should be in the form of a 3-dimensional.
7     (n_samples, segment_length, n_features)

```

<sup>2</sup>[https://numpy.org/devdocs/reference/generated/numpy.lib.stride\\_tricks.sliding\\_window\\_view.html](https://numpy.org/devdocs/reference/generated/numpy.lib.stride_tricks.sliding_window_view.html)

```

7
8 Parameters
9 -----
10 df : pd.DataFrame
11     the input frame should already be pre-processed, e.g. tagging
12     target..;
13     It should only contain the feature and target columns.
14 segment_length: int
15     window size
16 step_distance: float
17     the overlap of window size between (0,1).
18 sensor_cols: list
19     all features in the data that require feature engineering.
20 target: str
21     aka. column to be forecast.
22
23 Returns
24 -----
25 X: ndarray y: ndarray
26     the 3D array after the sliding window(based on the specific
27     segment length and step distance);
28     an array of the most common value for the passed array.
29 """
30 step = round(segment_length * float(step_distance)) # window
31     size * overlap
32 X_slide = sliding_window_view(df[sensor_cols], (segment_length
33     , len(sensor_cols)))[:, :step, :] # 4d array
34 y_slide = sliding_window_view(df[target],
35     segment_length)[:, :step, :] # 2d array
36 X = X_slide.reshape(X_slide.shape[0] * X_slide.shape[1],
37     segment_length, len(sensor_cols))
38 # Return an array of the modal (most common) value for the
39     passed array.
40 y = stats.mode(y_slide.transpose(), axis=0)[0][0] # the shape
41     is now equal to the sample of X
42 return X, y

```

For each window, the corresponding target is determined by employing a mode (most common value) strategy. This approach effectively captures the most frequent target value in each window, providing a robust and representative target for each windowed segment of the dataset.





## 4. Technology

### 4.1 Automated feature engineering technology

In this section, we will discuss the Python machine learning libraries (tsfresh [5], tsfel [11], kats, seglearn [30], cesium [34]) that we have utilized in our framework. These libraries provide a comprehensive set of tools and functionalities for time series analysis and feature extraction. They offer a wide range of capabilities for time series analysis, feature extraction, and modeling. Leveraging these libraries, we enhance the feature representation of our design matrix and empower our framework to handle diverse time series data efficiently.

Indeed, the focus of each feature extraction library is not identical. This implies that the time series characterization methods  $f_j$  applied to respective time series  $\chi_i$  yield distinct feature vectors, which is why, at the same time, we can use multiple feature extraction packages to extract the maximum number of features and then use feature selection methods to filter out features that are not very important to the target.

#### 4.1.1 TSFRESH

The Python-based machine learning library tsfresh is a rapid and standardized tool used for automatic extraction and selection of time series features, which has been integrated into our framework as well.

Tsfresh provides 63 time series characterization methods, which compute a total of 794 time series features. A design matrix of univariate attributes can be extended by time series features from one or more associated time series. Alternatively, a design matrix can be generated from a set of time series, which might have different number of data points and could comprise different types of time series [5]. It provides three basic feature extraction interfaces: “minimal”, “efficient”, and “comprehensive”. Additionally, we have utilized its feature selection method based on statistical hypothesis testing, offering two variations. One variation emphasizes selecting the union of features beneficial for each individual class (see code 4.2), while the other variation focuses on selecting the top `n_significant` features most strongly correlated with all of the target variables (see code 4.3). This enables us to restrict the

number of features selected during the feature selection process and retain only the most important ones.

Listing 4.1: TSFRESH features

```

1 from tsfresh import extract_features
2 from tsfresh.feature_extraction import EfficientFCParameters,
  MinimalFCParameters, ComprehensiveFCParameters
3 extraction_settings_dict = {
4     'minimal': MinimalFCParameters(),
5     'efficient': EfficientFCParameters(),
6     'comprehensive': ComprehensiveFCParameters()
7 }

```

Listing 4.2: TSFRESH feature selection\_var1

```

1 from tsfresh import select_features
2 import pandas as pd
3
4 def features_selector_variant1(x_train, x_test, y_train):
5     y_train_pd = pd.Series(y_train)
6     relevant_features = set()
7     for label in y_train_pd.unique():
8         y_train_binary = y_train_pd == label
9         x_train_filtered = select_features(x_train, y_train_binary)
10        relevant_features = relevant_features
11            .union(set(x_train_filtered.columns))
12
13    x_train_filtered = x_train[list(relevant_features)]
14    x_test_filtered = x_test[list(relevant_features)]
15
16    return x_train_filtered, x_test_filtered

```

Listing 4.3: TSFRESH feature selection\_var2

```

1 from tsfresh import select_features
2 import math
3 import pandas as pd
4 def features_selector_variant2(x_train, x_test, y_train):
5     '''
6     n_significant is set to 20-percent of the number of classes (
7     rounded up). This means that, after correction, at least
8     this many features in each class should be considered
9     significant in relation to the target. If a feature does not
10    meet this criterion in all classes, then it will not be
11    selected as significant.
12    '''
13    n_unique_classes = pd.Series(y_train).nunique()
14    x_train_filtered_multi = select_features(x_train,
15    pd.Series(y_train), multiclass=True, n_significant=math.ceil(
16    n_unique_classes * 0.2))
17
18    x_test_filtered_multi = x_test[x_train_filtered_multi.columns]
19    return x_train_filtered_multi, x_test_filtered_multi

```

### 4.1.2 TSFEL

Tsfel features a diverse suite of feature extraction methods that are categorized into three main domains: “temporal”, “spectral”, and “statistical”. It offers over 60 differ-

ent features that encompass autocorrelation, energy, entropy, statistical moments, spectral analysis, peak analysis, and many more [11]. In addition, we integrated TSFEL’s internal Pearson correlation coefficient-based feature selection method <sup>1</sup> to filter out highly correlated features.

Listing 4.4: TSFEL features

```

1 import tsfel
2 domains = ['statistical', 'spectral', 'temporal']
3 cfg_file = {}
4 def extract_features(domain):
5     cfg_file.update(tsfel.get_features_by_domain(domain))

```

Listing 4.5: TSFEL feature selection

```

1 import tsfel
2 def features_selector_tsfel(x_train, x_test):
3     # redundancies and noise should be removed.
4     # Compute pairwise correlation of features using pearson method
5     redundant_features = tsfel.correlated_features(x_train)
6
7     x_train_filtered = x_train.drop(redundant_features, axis=1)
8     x_test_filtered = x_test.drop(redundant_features, axis=1)
9     # Sort the columns of the training and test data to ensure the
10    features are in the same order
11    x_train_filtered = x_train_filtered.sort_index(axis=1)
12    x_test_filtered = x_test_filtered.sort_index(axis=1)
13
14    return x_train_filtered, x_test_filtered

```

### 4.1.3 KATS

Kats is released by Facebook’s Infrastructure Data Science team. The time series feature (TSFeature) extraction module in Kats can produce 65 features with clear statistical definitions <sup>2</sup>.

Kats divides the features that can be extracted into groups such as “statistics”, “acfpacf\_features”, “bocp\_detector” and so on.

Our framework integrates several groups, allowing users to select as per their requirements.

Listing 4.6: KATS features

```

1 from kats.tsfeatures.tsfeatures import TsFeatures
2
3 features_to_use = ['acfpacf_features', 'statistics', 'bocp_detector',
4                  'cusum_detector', 'level_shift_features', 'robust_stat_detector',
5                  'special_ac', 'trend_detector']

```

### 4.1.4 SEGLEARN

Seglearn provides about 30 features <sup>3</sup>, and since these features are all statistically based, there are no features from other domains. We have grouped these statistical features together under the name “seglearn”.

<sup>1</sup>[https://tsfel.readthedocs.io/en/latest/\\_modules/tsfel/utis/signal\\_processing.html](https://tsfel.readthedocs.io/en/latest/_modules/tsfel/utis/signal_processing.html)

<sup>2</sup><https://facebookresearch.github.io/Kats/>

<sup>3</sup>[https://dmbec.github.io/seglearn/feature\\_functions.html](https://dmbec.github.io/seglearn/feature_functions.html)

Listing 4.7: SEGLEARN features

```

1 from seglearn.feature_functions import all_features
2
3 features_to_use = {
4     'mean': mean,
5     'median': median,
6     'gmean': gmean,
7     'hmean': hmean,
8     'vec_sum': vec_sum,
9     'abs_sum': abs_sum,
10    'abs_energy': abs_energy,
11    'std': std,
12    'var': var,
13    'mad': median_absolute_deviation,
14    'variation': variation,
15    'min': minimum,
16    'max': maximum,
17    'skew': skew,
18    'kurt': kurt,
19    'mean_diff': mean_diff,
20    'mean_abs_diff': means_abs_diff,
21    'mse': mse,
22    'mnx': mean_crossings,
23    'hist4': hist(),
24    'mean_abs_value': mean_abs,
25    'zero_crossings': zero_crossing(),
26    'slope_sign_changes': slope_sign_changes(),
27    'waveform_length': waveform_length,
28    'emg_var': emg_var,
29    'root_mean_square': root_mean_square,
30    'willison_amplitude': willison_amplitude()}

```

### 4.1.5 CESIUM

Cesium provides a large number of feature extraction methods, divided into “Cadence/Error”, “General” and “Lomb-Scargle (Periodic)”<sup>4</sup>. We have selectively employed a selection of these feature extraction methods, depending on the type of our dataset, as follows:

Listing 4.8: CESIUM features

```

1 from cesium import featurize
2
3 features_to_use = [
4     "all_times_nhist_numpeaks",
5     "all_times_nhist_peak1_bin",
6     "all_times_nhist_peak2_bin",
7     "all_times_nhist_peak3_bin",
8     "all_times_nhist_peak4_bin",
9     "all_times_nhist_peak_1_to_2",
10    "all_times_nhist_peak_1_to_3",
11    "all_times_nhist_peak_1_to_4",
12    "all_times_nhist_peak_2_to_3",
13    "all_times_nhist_peak_2_to_4",
14    "all_times_nhist_peak_3_to_4",
15    "all_times_nhist_peak_val",

```

<sup>4</sup>[https://cesium-ml.org/docs/feature\\_table.html](https://cesium-ml.org/docs/feature_table.html)

```
16     "avg_double_to_single_step",
17     "avg_err",
18     "cad_probs_1",
19     "cad_probs_10",
20     "cad_probs_20",
21     "cad_probs_30",
22     "cad_probs_40",
23     "cad_probs_50",
24     "cad_probs_100",
25     "cad_probs_500",
26     "cad_probs_1000",
27     'cads_avg',
28     'cads_kurtosis',
29     'cads_med',
30     'cads_skew',
31     'cads_std',
32     "mean",
33     'med_double_to_single_step',
34     "med_err",
35     'std_double_to_single_step',
36     "std_err",
37     "amplitude",
38     'anderson_darling',
39     'flux_percentile_ratio_mid20',
40     'flux_percentile_ratio_mid35',
41     'flux_percentile_ratio_mid50',
42     'flux_percentile_ratio_mid65',
43     'flux_percentile_ratio_mid80',
44     "max_slope",
45     "maximum",
46     "median",
47     "median_absolute_deviation",
48     "minimum",
49     'percent_amplitude',
50     "percent_beyond_1_std",
51     "percent_close_to_median",
52     'percent_difference_flux_percentile',
53     'qso_log_chi2_qsonu',
54     'qso_log_chi2nuNULL_chi2nu',
55     'shapiro_wilk',
56     "skew",
57     "std",
58     'stetson_j',
59     'stetson_k',
60     "weighted_average",
61     "fold2P_slope_10percentile",
62     "fold2P_slope_90percentile",
63     "freq1_amplitude1",
64     "freq1_amplitude2",
65     "freq1_amplitude3",
66     "freq1_amplitude4",
67     "freq1_freq",
68     "freq1_lambda",
69     "freq1_rel_phase2",
70     "freq1_rel_phase3",
71     "freq1_rel_phase4",
72     "freq1_signif",
73     "freq2_amplitude1",
74     "freq2_amplitude2",
```

```

75     "freq2_amplitude3",
76     "freq2_amplitude4",
77     "freq2_freq",
78     "freq2_rel_phase2",
79     "freq2_rel_phase3",
80     "freq2_rel_phase4",
81     "freq3_amplitude1",
82     "freq3_amplitude2",
83     "freq3_amplitude3",
84     "freq3_amplitude4",
85     "freq3_freq",
86     "freq3_rel_phase2",
87     "freq3_rel_phase3",
88     "freq3_rel_phase4",
89     "freq_amplitude_ratio_21",
90     "freq_amplitude_ratio_31",
91     "freq_frequency_ratio_21",
92     "freq_frequency_ratio_31",
93     "freq_model_max_delta_mags",
94     "freq_model_min_delta_mags",
95     "freq_model_phi1_phi2",
96     "freq_n_alias",
97     "freq_signif_ratio_21",
98     "freq_signif_ratio_31",
99     "freq_varrat",
100    "freq_y_offset",
101    "linear_trend",
102    "medperc90_2p_p",
103    "p2p_scatter_2praw",
104    "p2p_scatter_over_mad",
105    "p2p_scatter_pfold_over_mad",
106    "p2p_ssqr_diff_over_var",
107    "scatter_res_raw",
108 ]

```

## 4.2 Discussions

Firstly, it is clear that there will unavoidably be a small number of redundant feature methods within the feature extraction package, as can be seen in methods such as mean, median, std, etc.

Thanks to subsequent feature selection methods, we can eliminate these highly correlated repetitive features and retain only a representative set of features before feeding them into the model.

Secondly, we try to achieve a good trade-off between accuracy and computational overhead. The calculation of a large number of features increases the computation time and thus affects efficiency. We therefore offer users a full range of possibilities by providing feature extraction packages that include multiple methods (e.g., `tsfresh`) as well as compact and efficient feature extraction packages (e.g., `seglearn`).

### 4.2.1 Overfitting

Overfitting is a common problem in supervised machine learning, where models are susceptible to fitting the training data too perfectly. This can undermine their ability to generalize well to unseen data in the testing set. Because of the presence of noise, the limited size of training set, and the complexity of classifiers, overfitting happens [39]. Therefore, it's crucial to extract more "informative" features.

An "informative" feature should satisfy the following conditions:

- **Relevance to the target:** An informative feature should have a high relevance with the target, which means that changes in the feature should significantly affect the target.
- **Contains informative information:** If all the values of a feature are the same, then the feature cannot provide any useful information.
- **Low noise levels:** If a feature contains too much noise, it could interfere with the model, leading to worse performance.

### 4.2.2 Curse of dimensionality

The concept of the "curse of dimensionality" was proposed by Richard Bellman in 1961 [12]. In machine learning and data analysis, the curse of dimensionality typically refers to the decline in performance of many algorithms as the number of features (or dimensions) increases, especially when the number of features approaches or exceeds the number of samples. This is mainly because data points tend to become very sparse in high-dimensional spaces, making it more difficult to identify effective patterns and regularities.

More generally, the curse of dimensionality represents all the phenomena that occur with high-dimensional data, often resulting in reduced accuracy and poorer performance in machine learning [13].





# 5. Model

In this chapter, we will delve into the basic concepts and principles of the models that were previously introduced in Section 2.3. We will explore the similarity-based classification method of  $k$ -nearest neighbors, the generalised linear model of logistic regression, and the features and advantages of random forests as an ensemble learning method. In addition, we will look at the neural network structure of multilayer perceptrons.

By getting a deeper understanding of the principles and workings of these models, we will better understand their application within our framework, helping us to select the most appropriate models to solve various time series problems.

## 5.1 $K$ -nearest neighbors

The  $k$ -nearest neighbors ( $k$ NN) is a powerful non-parametric classification algorithm that operates without making any assumptions about the underlying data distribution. It is widely recognized for its simplicity and effectiveness. As a supervised learning algorithm, it leverages a labeled training dataset, where data points are already classified into different classes, to make predictions on unlabeled data instances [14].

The right side figure 5.1 represents a simple  $k$ NN model. When  $k = 1$ , due to the higher number of orange triangles compared to green squares, the unlabeled blue data points are classified by the  $k$ NN model as orange triangles. However, when  $k = 2$ , as there are more green squares, they are considered as blue squares.

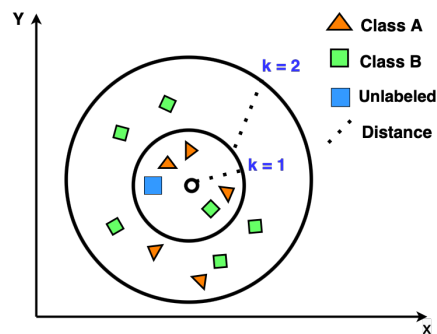


Figure 5.1: A simple  $k$ NN-model with  $k = 2$ .

So the principle of the  $k$ NN algorithm in supervised learning is as follows:

1. Determine the number of nearest neighbors.
2. For each unlabeled sample to be classified, calculate its distance to each labeled sample in the training set. Common distance metrics include Euclidean distance, Manhattan distance, etc.
3. Find the  $k$ -nearest neighbours.
4. Assign class containing the maximum number of nearest neighbours.

## 5.2 Logistic regression

The basic principle of logistic regression is to linearly combine the input features and then map the linear output to the range  $[0, 1]$  using the logistic function [15], also known as the sigmoid function, defined as follows:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}. \quad (5.1)$$

Since the output range of the sigmoid function is  $[0,1]$ , we can set the output probability threshold to 0.5 in practice for binary classification prediction. If the output probability is greater than or equal to 0.5, we predict the sample as a positive case; if the output probability is less than 0.5, we predict the sample as a negative case.

For a problem with  $K$  classes ( $K > 2$ ), we can train  $K$  binary logistic regression classifiers, each classifier treating one class as the positive class and the remaining classes as the negative class. This strategy is used to solve the multi-class classification problem.

This following figure 5.2 illustrates the application of logistic regression to solve the multi-class classification problem in the Iris dataset <sup>1</sup>. The sample points are distributed in a coordinate system based on the length and width of the sepals. The model then partitions these points into three distinct regions. Sample points located in the same region are considered to belong to the same class. For example, samples in the purple region are classified as setosa, while those in the yellow region are classified as virginica, and so on.

## 5.3 Random forest

The random forest was first proposed by Leo Breiman from the University of California in 2001 [16].

The model comprises numerous fundamental classifiers (decision trees), each operating independently of one another. When a sample is fed into this composite classifier, the classification of the sample is determined by the majority voting result from each individual classifier.

The entire process of classification based on the random forest is shown in the Figure 5.3.

According to the [16] and the figure 5.3, the construction of a random forest involves the following steps:

<sup>1</sup><https://archive.ics.uci.edu/dataset/53/iris>

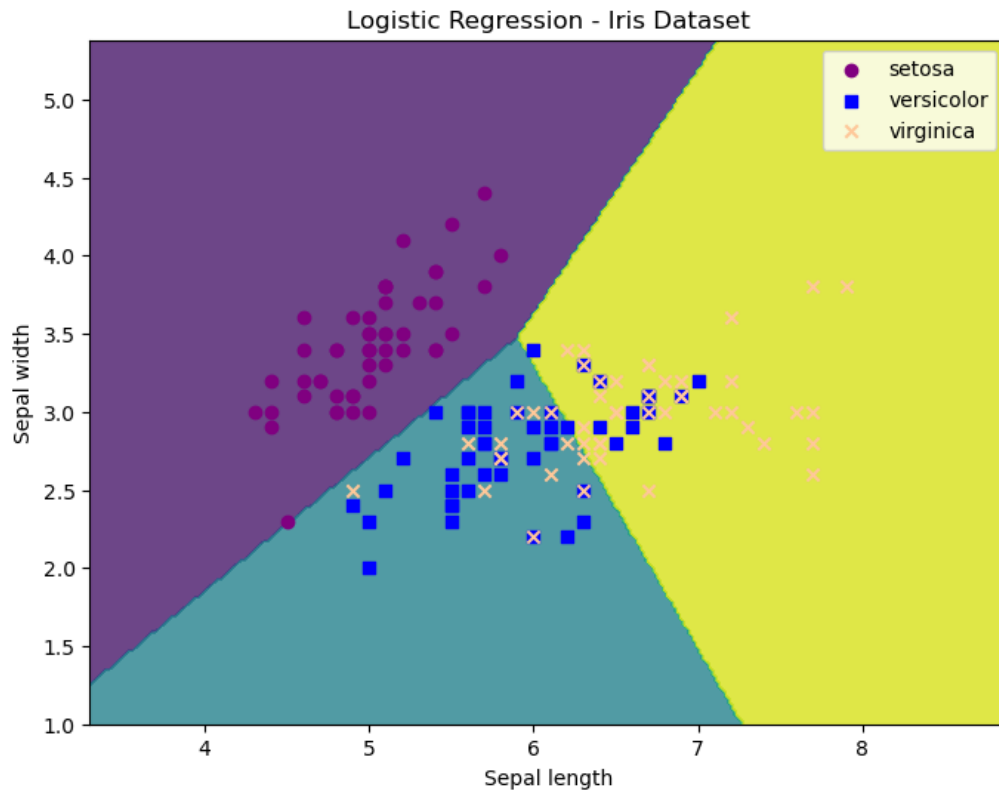


Figure 5.2: Logistic regression with iris dataset.

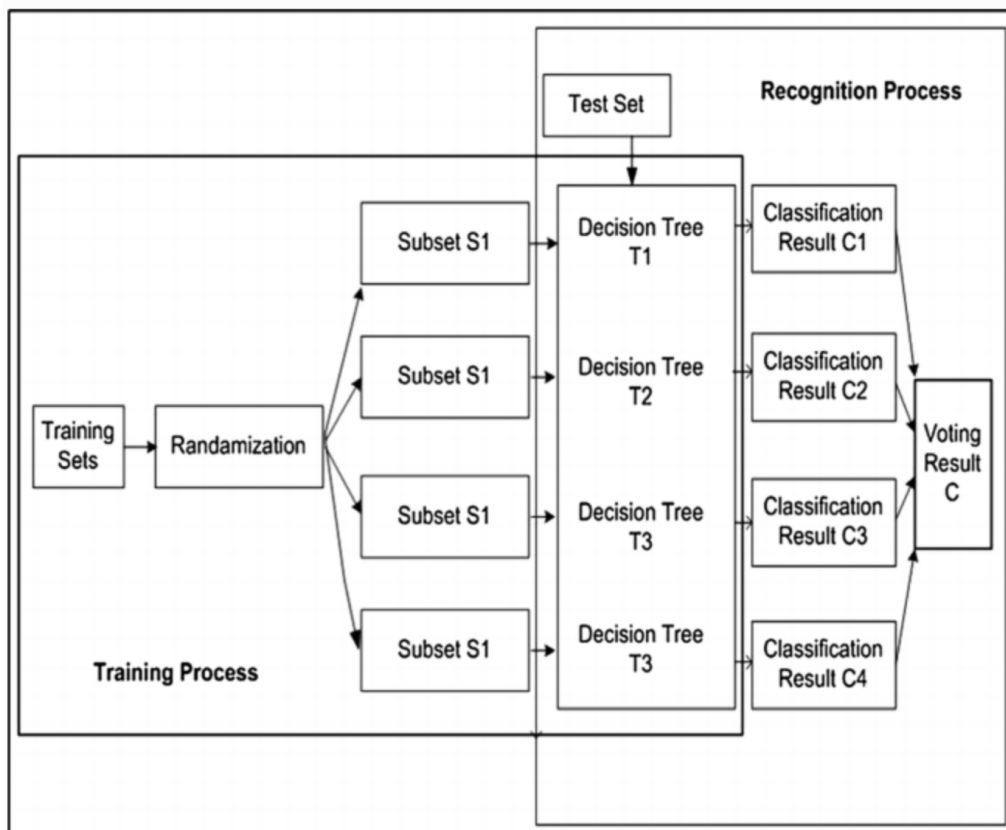


Figure 5.3: Conceptual framework of random forest classifier [17, 18]

1. Determine an appropriate value for the number of elements in each feature subset.
2. Generate a random subset  $\Theta_k$  from the whole set depending on the value at step 1, independent of the past random subset  $\Theta_1, \dots, \Theta_{k-1}$  but with the same distribution.
3. Train the dataset using the selected subsets, generating a decision tree for each training set group.
4. Repeat steps 2 and 3 until a specified number of trees have been generated. This process results in a collection of tree-structured classifiers denoted as:  $\{h(\mathbf{x}, \Theta_k), k = 1, 2, 3, \dots\}$ , where  $\mathbf{x}$  represents the input features.
5. Cast a unit vote for the most probable class at each instance in the test set, with each tree in the forest contributing to the voting process.
6. Assign the class with the most votes across all trees as the final prediction for each instance.

## 5.4 Multilayer perceptron

The perceptron is one of the fundamental building blocks of neural networks, and it serves as the foundation for the multilayer perceptron (MLP), which is the most widely known and frequently used type of neural network [19].

The perceptron usually consists of 4 parts, the input value; the weights and bias; the net input function and the activation function as shown in figure 5.4.

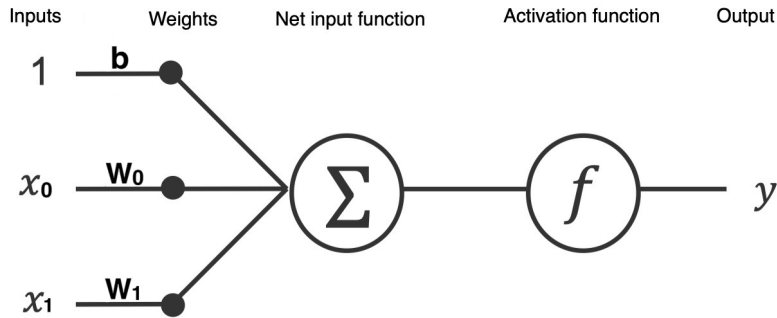


Figure 5.4: The structure of a single perceptron.

It comprises several input nodes. Each input node is associated with a specific weight, and these inputs are multiplied by their respective weights. The results of these multiplications are then summed up and passed through an activation function, which ultimately determines the output of the perceptron. I.e.,

$$y = f \left( \sum_{i=1}^k x_i \omega_i + b \right), \quad (5.2)$$

where  $k$  represents the number of inputs and  $b$  specifically denotes the bias.

The MLP consists of multiple layers of perceptrons (neurons), including an input layer, one or more hidden layers, and an output layer. The hidden layers allow the network to learn complex patterns by combining the features learned in the input layer [20]. Each perceptron in these layers performs a weighted summation of its inputs, similar to a single-layer perceptron, but followed by the application of a nonlinear activation function. This feed-forward architecture is characterized by its lack of loops, ensuring that the computations of any given perceptron are not influenced by its own output. The signals in the MLP typically flow through the network in a unidirectional manner, progressing from the input layer, through the hidden layers, and finally to the output layer [19]. The figure 5.5 shows a simple multilayer perceptron with 2 hidden layers.

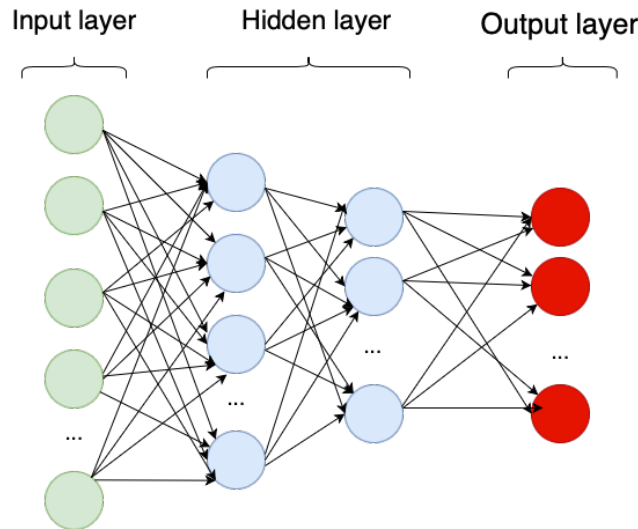


Figure 5.5: A multilayer feed-forward neural network.

Suppose our neural network has  $L$  hidden layers, we can represent the output of each layer as  $O^{(l)}$ , where  $l$  denotes the index of the layer,  $l = 0$  corresponds to the input layer and  $l = L + 1$  corresponds to the output layer. We can represent the output of neuron  $j$  at layer  $l$  as  $O_j^{(l)}$ .

For the hidden layer, the output of each neuron is calculated according to the formula 5.2, which can be expressed as

$$O_i^{(l)} = f \left( \sum_{j=1}^{N^{(l-1)}} w_{ji}^{(l)} O_j^{(l-1)} + b_i^{(l)} \right),$$

where

- $O_i^{(l)}$  is the output of the  $i$ -th neuron in the  $l$ -th layer.
- $f$  is activation function.
- $w_{ji}^{(l)}$  is the weight of the  $j$ -th neuron in the previous layer to the  $i$ -th neuron in the current layer.

- $O_j^{(l-1)}$  is the output of the  $j$ -th neuron in the previous layer.
- $b_i^{(l)}$  is the bias of the  $i$ -th neuron in the  $l$ -th layer.
- $N^{(l-1)}$  is the number of neurons in the previous layer.

### 5.4.1 Activation function

In a neural network, the activation function has two main roles:

- **Introduce non-linearity:** To handle complex tasks, neural networks often need to solve non-linear problems. Activation functions are typically non-linear, enabling neural networks to approximate any non-linear function, which enhances the expressive power of the networks.
- **Determine neuron activation:** The activation function decides the degree to which a neuron should be activated in response to a given input.

Our framework uses the MLP from scikit-learn, which includes four types of activation functions, namely <sup>2</sup>:

- **Identity:**  $f(x) = x$ .
- **Logistic:**  $f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$ .
- **Tanh:**  $f(x) = \tanh(x)$ .
- **ReLU:**  $f(x) = \max(0, x)$ .

In practice, additional activation functions such as leaky ReLU( $\cdot$ ) and softmax( $\cdot$ ) are also frequently utilized. Each of these has its own advantages and suitable scenarios. For example, the identity function is useful to implement linear bottleneck, while the softmax function is often used in the output layer for multi-class classification problems.

---

<sup>2</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

## 5.5 Discussions

In the previous sections, we have discussed the fundamental concepts of four models:  $k$ NN, logistic regression, random forest, and MLP. However, one crucial point has been overlooked: how to evaluate the performance of a model?

There are numerous evaluation methods available, as mentioned in [21], including micro average F1-score, macro average F1-score, dodrans, entropy, among others. In this section, we will focus on F1-based scores and specifically highlight the macro average F1-score employed in our framework.

### 5.5.1 F1-score

F1-score is defined as the harmonic average of recall and precision. The mathematical formula is defined as

$$F1 = \frac{2 \times (\text{Recall} \times \text{Precision})}{(\text{Recall} + \text{Precision})}, \quad (5.3)$$

where

- Recall is defined as the ratio of observations that are predicted positively correct to the total number of observations in an actual class; that is,

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \mathbf{\text{False Negatives}}}.$$

- Precision is defined as the ratio of observations that are predicted positively correct to the total number of observations predicted positively:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \mathbf{\text{False Positives}}}.$$

### 5.5.2 Macro average F1-score

The macro average F1-Score provides an equal weight for each class, irrespective of their individual sample counts. It considers the F1-score for each class independently and then calculates their average [21]. This approach allows us to evaluate the model's performance without considering class imbalances. By giving equal importance to each class, macro average F1-score highlights the overall effectiveness of the model in handling diverse classes and can help identify areas where the model may struggle across different classes.

As mentioned in [35], there are two ways to implement macro average F1-score. In our framework, under the premise of utilizing scikit-learn <sup>3</sup>, we adopt the first approach, which involves calculating the arithmetic mean over harmonic means. Referring to 5.3, the mathematical expression can be defined as follows:

$$F1_M = \frac{1}{n} \sum_{i \in \text{Classes}} F1_i.$$

<sup>3</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)





## 6. Related work

AutoML has emerged as a hot topic with both industrial and academic interest. In recent years, numerous studies and methodologies have been proposed in this area, reflecting the growing recognition of the importance of AutoML. In the following sections, some of the current work in the field of AutoML will be described.

### 6.1 Novel feature selection methods

In general, feature selection methods can be categorized into three types: filter-based, wrapper-based, and embedded [33]. The feature selection methods used within our framework are all filter-based, as illustrated in listings 4.2, 4.3, and 4.5.

Tan et al. [36] proposed a new method called DimReM to deal with high dimensional problems in the search space. The method uses an evolutionary algorithm (EA), which belongs to the wrapper-based feature selection technique. DimReM provides an effective way to identify and delete unimportant features by taking advantage of the information from evolution and due to the re-evaluation of population, DimReM ensures that the classification performance never gets worse when a feature is deleted in each time [36].

In the realm of embedded feature selection methods, the elastic net has proven to be a viable strategy [22]. The elastic net is a regularized regression technique that incorporates an  $\ell_2$  penalty to circumvent the inherent limitations of the Least Absolute Shrinkage and Selection Operator (LASSO) [23].

However, the elastic net can lead to inconsistent feature selection, as the selected features and their weights may vary when the training set data samples change. Yu et al. [40] proposed an enhancement to the traditional elastic net process. Their methodology involves ranking features based on the probability of a feature being selected after multiple data splits and subsequent elastic net-based feature selection. This improvement mitigates the problem of the elastic net's inability to rank the importance of features consistently.

## 6.2 Model hyperparameter tuning

The work of Yao, Wang, et al. [24] fundamentally states three questions: what is AutoML, why AutoML, and how to implement AutoML. Notably, their research discusses a multitude of methods for model selection and hyperparameter optimisation. Beyond the Bayesian Optimisation adopted in our framework, they also consider classification-based optimisation (CBO), greedy algorithms, evolutionary algorithms, and reinforcement learning (RL), etc.

Yu et al. [25] proposed an algorithm based on CBO, named **RACOS**, and demonstrated that **RACOS** performs better than Bayesian optimisation in the task of hyperparameter tuning.

Greedy search is a natural strategy to solve multi-step decision-making problem. It follows a heuristic that makes locally optimal decision at each step, although it cannot find the global optimum, but it can usually find a local optimum which approximates the global optimum in a reasonable time cost.

Evolutionary algorithms, inspired by biological evolution, involve generation steps of crossover and mutation. Crossover combines two distinct individuals from the previous generation, typically favoring promising individuals, to produce a new one. Mutation slightly alters an individual to form a new one. Through these processes, exploiting via crossover and exploring via mutation, the population is expected to evolve towards improved performance [29].

RL [26] is a very general and strong optimisation framework, which can solve problems with delayed feedbacks. A special case of RL, i.e., bandit-based approach, where rewards are returned for each action without a delay, is introduced to AutoML for hyperparameter optimisation [27, 28]. In this context, employing a bandit-based approach, the “multiple arms” denote the various possible combinations of hyperparameters. The reward associated with each “arm” can be defined as the performance of the model on the validation set.

## 7. Programming environments

Our framework contains several feature extraction packages, each of which may have their own pre-conditions and dependencies. Careful management of version information for these packages is essential to avoid conflicts and to ensure the stability and compatibility of our framework.

And combining fastAPI and React technologies, we achieve seamless interaction between the front-end and back-end.

Finally, we have listed the specific version information of key packages here for reference and utilization.

- **Programming**

- PyCharm 2022.3
  - \* Python 3.8.0
    - Tsfresh 0.20.0
    - Tsfel 0.1.4
    - Seglearn 1.2.5
    - Kats 0.2.0
    - Cesium 0.12.1
    - Optuna 3.0.4
    - Pandas 1.5.3
    - Numpy 1.22.3
    - Scikit-learn 1.2.2
    - Scipy 1.7.3
    - Matplotlib 3.6.1
    - Fastapi 0.97.0
- Webstorm 2022.3.1
  - \* JavaScript ECMAScript 6+
  - \* React 18.2.0
  - \* Node 18.12.1
  - \* Material UI v5.14.1



## 8. Design of user interface

In this chapter, we will dive into a newly developed user interface (UI) front-end framework tailored to our feature-engineered AutoML pipeline. The framework is designed to simplify the often complex process of feature extraction, selection, transformation and optimisation, enabling both newcomers and experienced data scientists to streamline their workflow.

### 8.1 Welcome page

When a user enters the website, they are first taken to the welcome page 8.1. After waiting, the user will be redirected to the main page 8.2. This page will also be used in the future for registration and login.

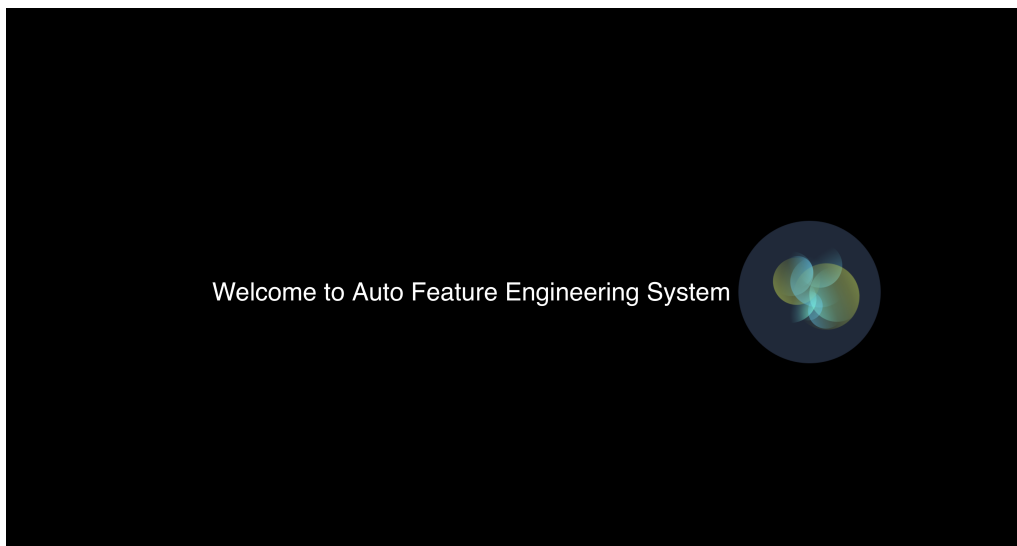


Figure 8.1: The animation on the welcome page.

## 8.2 Main page

This page will be our main execution page. In the center of the page is a pipeline, where the user needs to follow the steps to complete the appropriate actions.

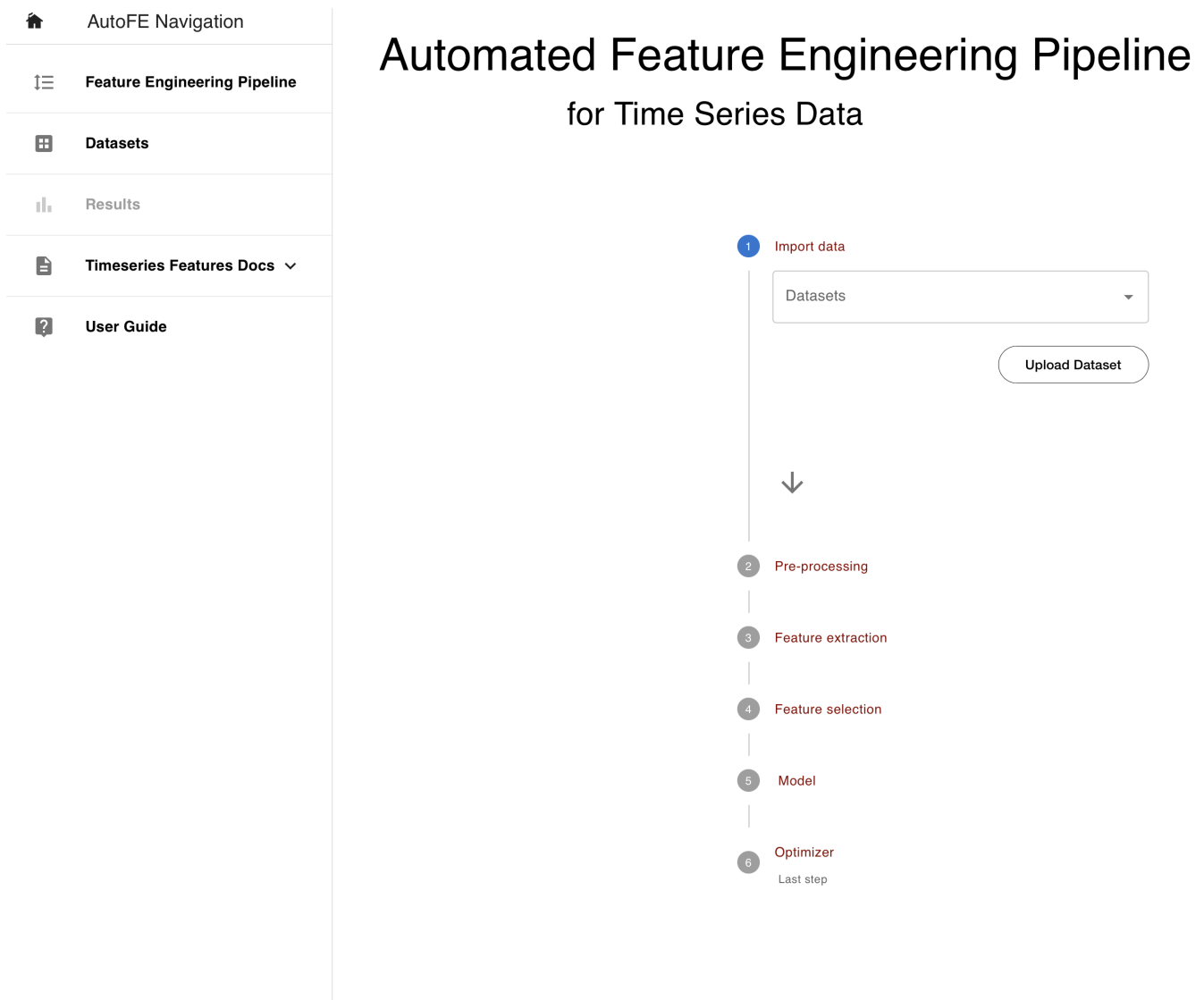


Figure 8.2: The main page of feature engineering pipeline.

### 8.2.1 Step 1: import data

The first step in the pipeline is, of course, to import the dataset. We provide two approaches:

- **Default** (figure 8.3 left): Users can select the dataset of the built-in system and start running it directly.
- **Customized** (figure 8.3 right): We also allow users to upload customized datasets by clicking on “upload dataset” button.

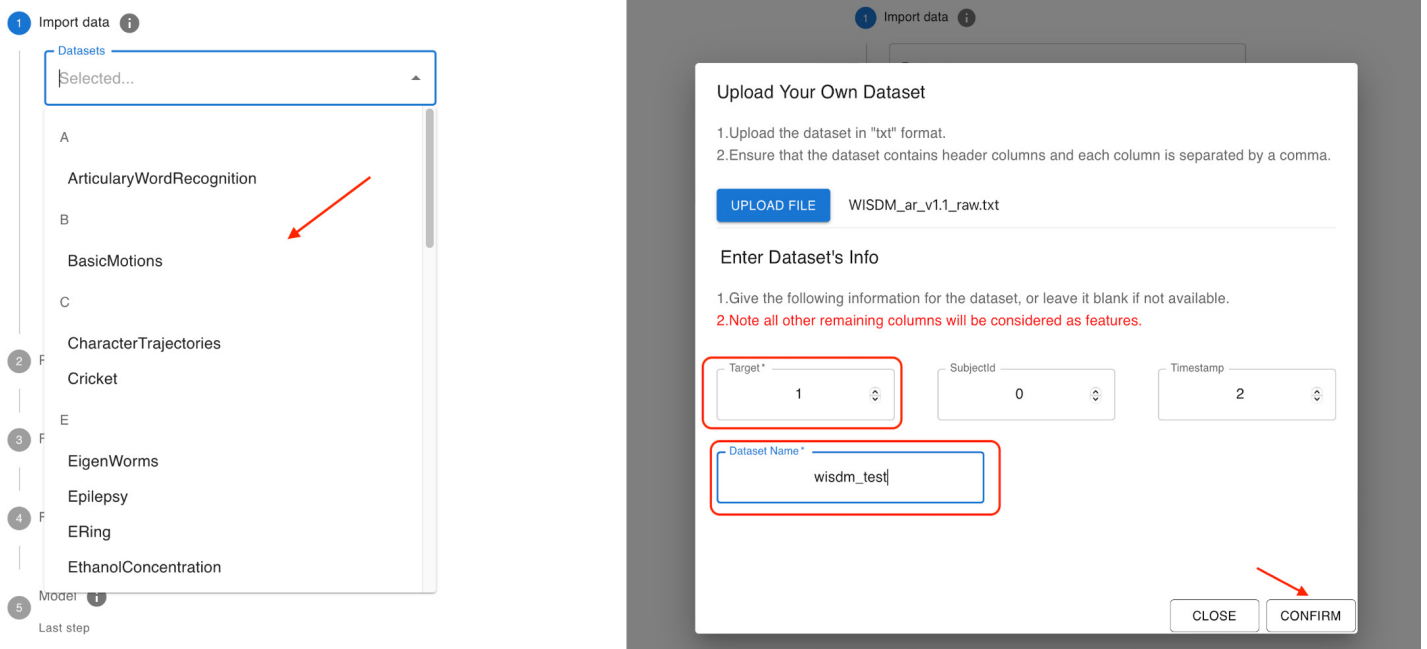


Figure 8.3: The default and customized mode of importing data.

## 8.2.2 Step 2: pre-processing

In this step we expect the user to provide some basic information about the dataset. Most of it is about the sliding window (section 3.1) operation. We also provide two approaches:

- **Default** (figure 8.4 left): All built-in datasets have fixed (unchangeable) window size/stride and frequency sampling.
- **Customized** (figure 8.4 right): Custom parameter attributes for specific datasets (especially those uploaded by users themselves).

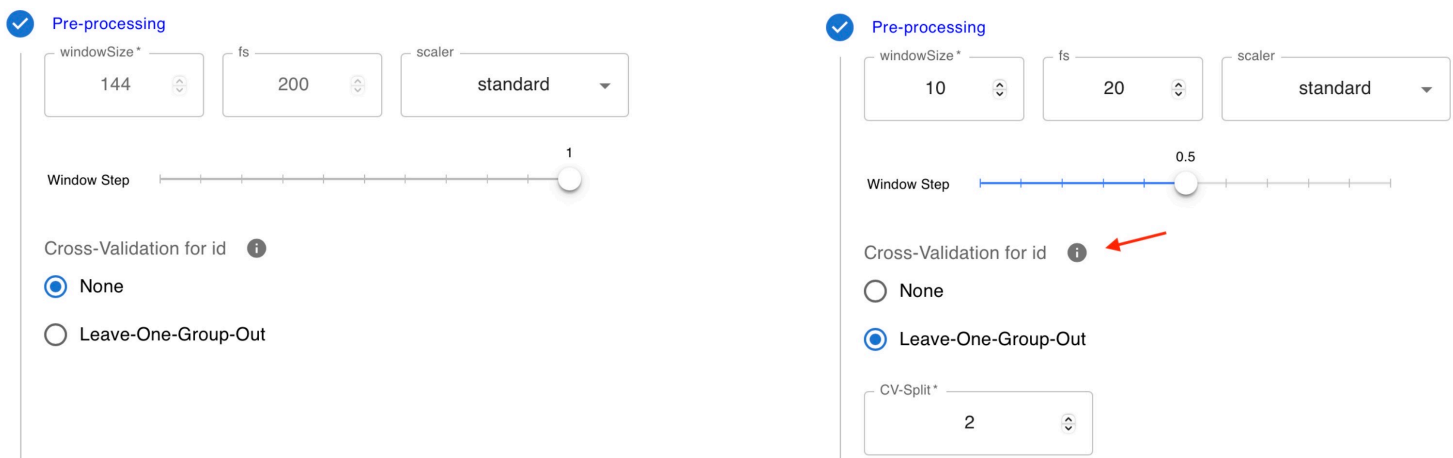


Figure 8.4: The default and customized mode of pre-processing.

For datasets integrated into the system, we still allow users to modify certain information, such as the scaler. Additionally, we offer a cross-validation method known as “Leave-One-Group-Out (LOGO)”, as illustrated in figure 8.5.

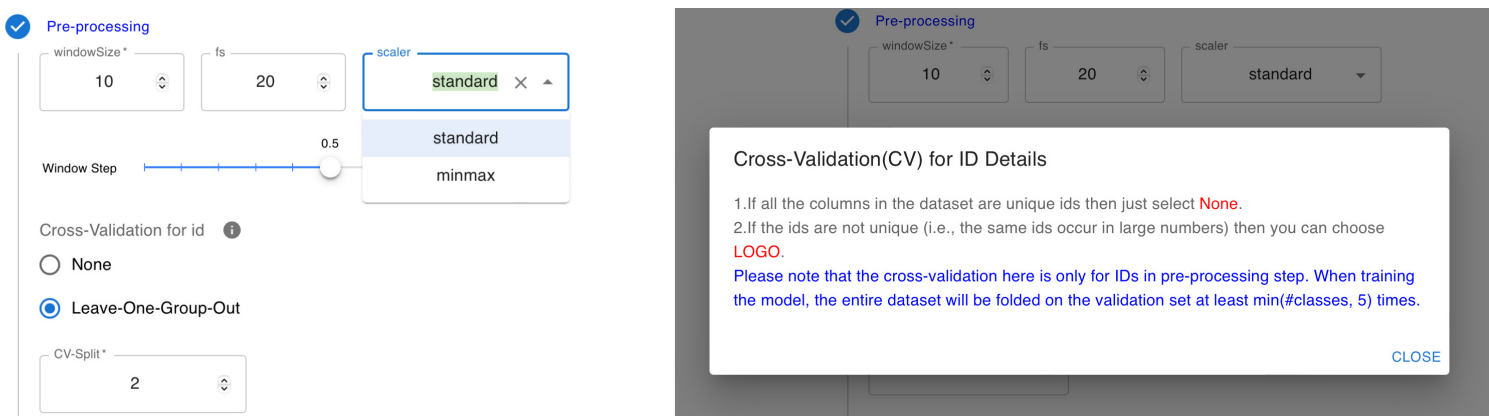


Figure 8.5: The details of the scaler setting and the dialog for the “cross-validation for id” information icon.



### 8.2.3 Step 3 & 4: feature extraction and feature selection

These two steps are the feature extraction and selection that we mentioned in chapter 4. As shown in the figure 8.6, the user can select a single feature package or combine multiple feature packages together. The feature selection method is the same.

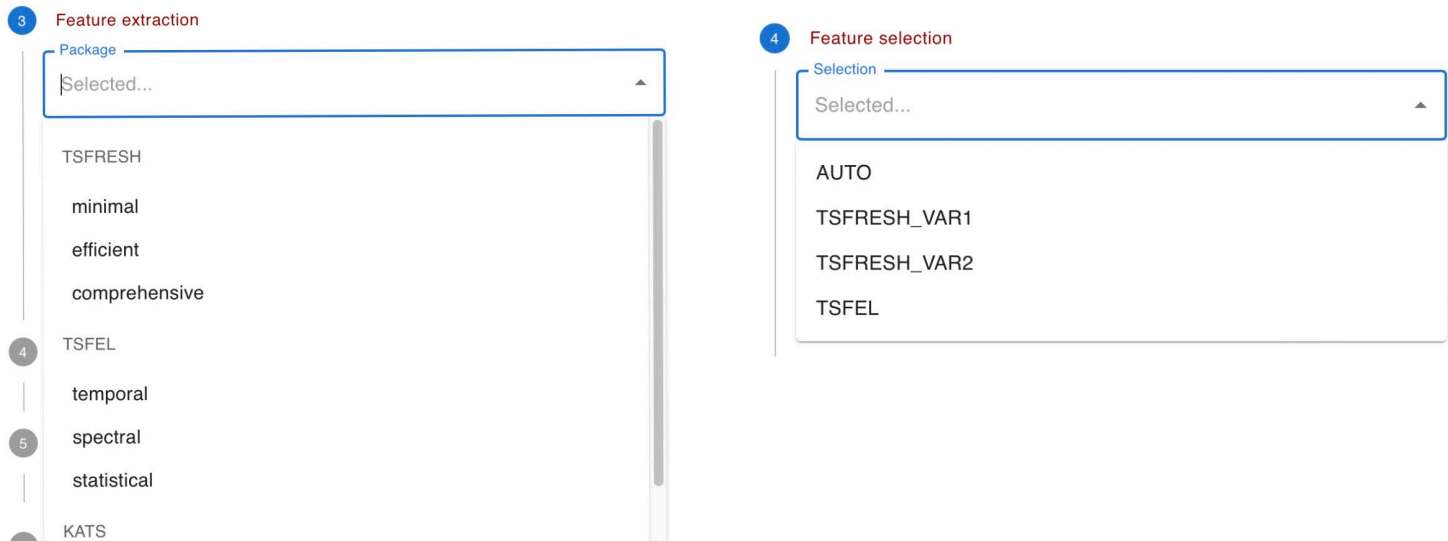


Figure 8.6: The step of feature extraction and feature selection.

### 8.2.4 Step 5: model

Our framework now offers 4 models to choose from. Users can click on a specific model button to perform hyperparameter tuning. If users wish to use these model, they need to also click the checkbox, the detailed operation can be seen in the figure 8.7.

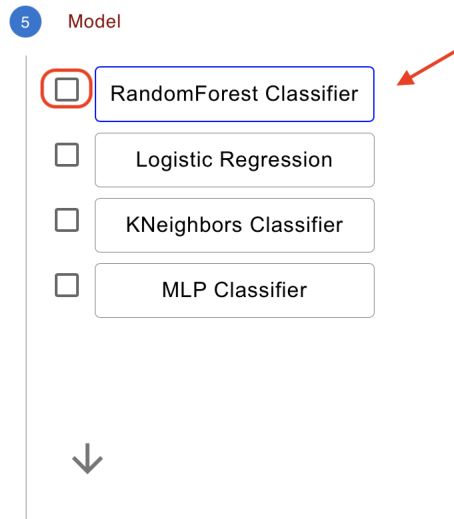


Figure 8.7: The overview of the model.

Each model is designed to operate similarly, and the following is an example of a random forest model. The left figure 8.8 shows the current settings dialog. Hints are given through the text. The user can easily do the same as in the right figure 8.8.

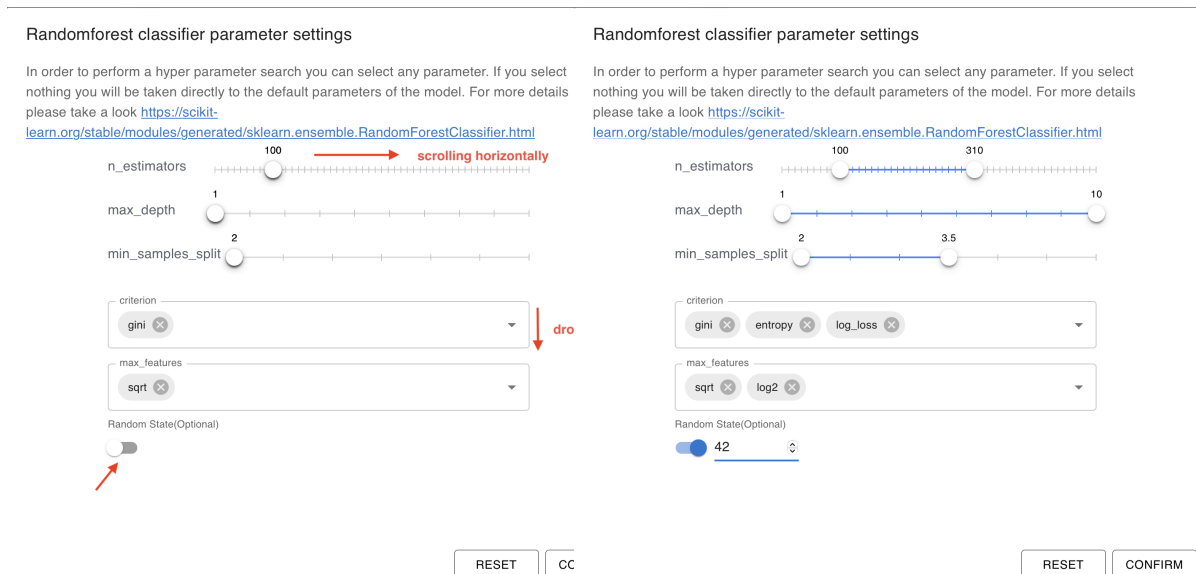
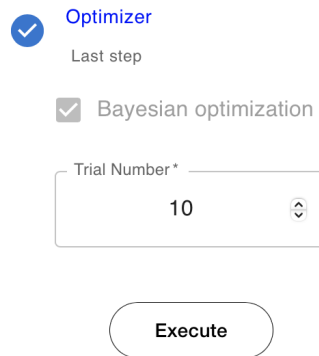


Figure 8.8: The original and changed random forest model parameter settings.

### 8.2.5 Step 6: optimizer

This is the final step in the pipeline. As you can see in figure 8.9, for feature selection and hyperparameter optimization of the model we currently only have Bayesian Optimisation. But it still requires the user to choose the number of trials needed. After clicking the “Execute” button, the program will start executing automatically.



The screenshot shows a user interface for the optimizer. At the top, there is a blue checkmark icon next to the word "Optimizer" and the text "Last step". Below this is a checked checkbox next to "Bayesian optimization". A dropdown menu labeled "Trial Number\*" is open, showing the value "10". At the bottom of the interface is a rounded rectangular button labeled "Execute".

Figure 8.9: The optimizer of the pipeline.

Upon completion of all steps by the user, the title of each step will transition from red to blue. This indicates that the user has not omitted any required inputs. Once the backend processes have been executed, the results will be returned.

## 8.3 Navigation bar

The navigation bar (figure 8.10) is built into every page. Users can choose what they want to see, and the main page we just introduced is the first item in the navigation bar.

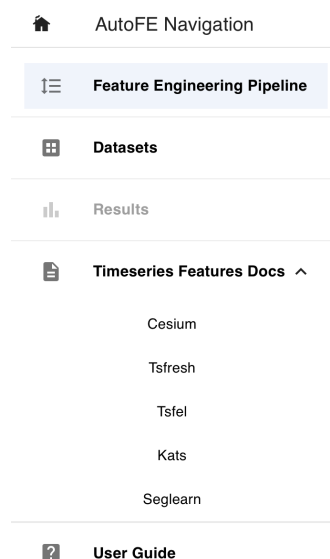


Figure 8.10: The navigation bar of the website.

## 8.4 Datasets page

On this page (figure 8.11) we give some basic information about the built-in datasets such as train cases, test cases, and so on. The dataset can be viewed directly by clicking on the dataset. The table also supports querying, sorting, etc.

**Built-in Datasets Information**

The following datasets are built-in the system. We obtained them from the official websites of [UCR](#) and [UCI](#). Clicking on them will provide you with some basic information. For detailed information, please visit the official websites directly.

Dataset	Train Cases	Test Cases	Dimensions	Length	Classes	Type
<b>ArticularyWordRecognition</b> This is the EMA dataset which contains data collected from multiple native English native speakers producing 25 words. Twelve sensors were used in data collection, each providing X, Y and Z time-series positions with a sampling rate of 200 Hz.	275	300	9	144	25	MOTION
<b>BasicMotions</b> The data was generated as part of a student project where four students performed four activities whilst wearing a smart watch. The watch collects 3D accelerometer and a 3D gyroscope. The data order is accelerometer x, y, z then gyroscope x, y, z. There are classes: walking, resting, running and badminton. Participants were required to record motion a total of five times, and the data is sampled once every tenth of a second, for a ten second period.	40	40	6	100	4	HAR
<b>CharacterTrajectories</b> The characters here were used for a PhD study on primitive extraction using HMM based models. The data consists of 2858 character samples. The data was captured using a WACOM tablet. 3 Dimensions were kept - x, y, and pen tip force. The data has been numerically differentiated and Gaussian smoothed, with a sigma value of 2. Data was captured at 200Hz.						
<b>Cricket</b> Cricket requires an umpire to signal different events in the game to a distant scorer/bookkeeper.						

1-5 of 15

Figure 8.11: The dataset page with basic details about built-in datasets.

## 8.5 Time series features document page

In this page, we have provided a comprehensive list of all the feature extraction packages available, as mentioned in chapter 4. Detailed features and their descriptions are presented for user reference.

AutoFE Navigation

- Feature Engineering Pipeline
- Datasets
- Results
- Timeseries Features Docs ^
  - Cesium
  - Tsfresh
  - Tsfel
  - Kats
  - Seglearn
- User Guide

### Cesium Features

The following features are a detailed description of the cesium feature extraction library used in our framework. For more detailed information please refer to the cesium website [view the link](#).

Feature Name	Description
all_times_nhist_numpeaks	Number of peaks (local maxima) in histogram of all possible delta_t's.
all_times_nhist_peak1_bin	Return the (bin) index of the ith largest peak. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak2_bin	Return the (bin) index of the ith largest peak. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak3_bin	Return the (bin) index of the ith largest peak. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak4_bin	Return the (bin) index of the ith largest peak. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_1_to_2	Compute the ratio of the values of the ith and jth largest peaks. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_1_to_3	Compute the ratio of the values of the ith and jth largest peaks. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_1_to_4	Compute the ratio of the values of the ith and jth largest peaks. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_2_to_3	Compute the ratio of the values of the ith and jth largest peaks. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_2_to_4	Compute the ratio of the values of the ith and jth largest peaks. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_3_to_4	Compute the ratio of the values of the ith and jth largest peaks. Peaks is a list of tuples (i, x[i]) of peak indices i and values x[i], sorted in decreasing order by peak value.
all_times_nhist_peak_val	Peak value in histogram of all possible delta_t's.
avg_err	Mean of the error estimates.

Figure 8.12: The time series features document page, listing all feature extraction packages for time series.

## 8.6 User guide page

We have always prioritized fostering an optimal interaction with our users. Hence, we offer a “quick start” guide aimed at enhancing the user experience. Both text-based and video guides are provided to cater to various user needs.

### 8.6.1 User guide panel

This is a comprehensive textual description for each step in the pipeline. Users can simply click on any step to access the corresponding documentation or functionality they wish to explore. Figure 8.13 provides an overview of the user guide panel, while Figure 8.14 offers an example of detailed view.

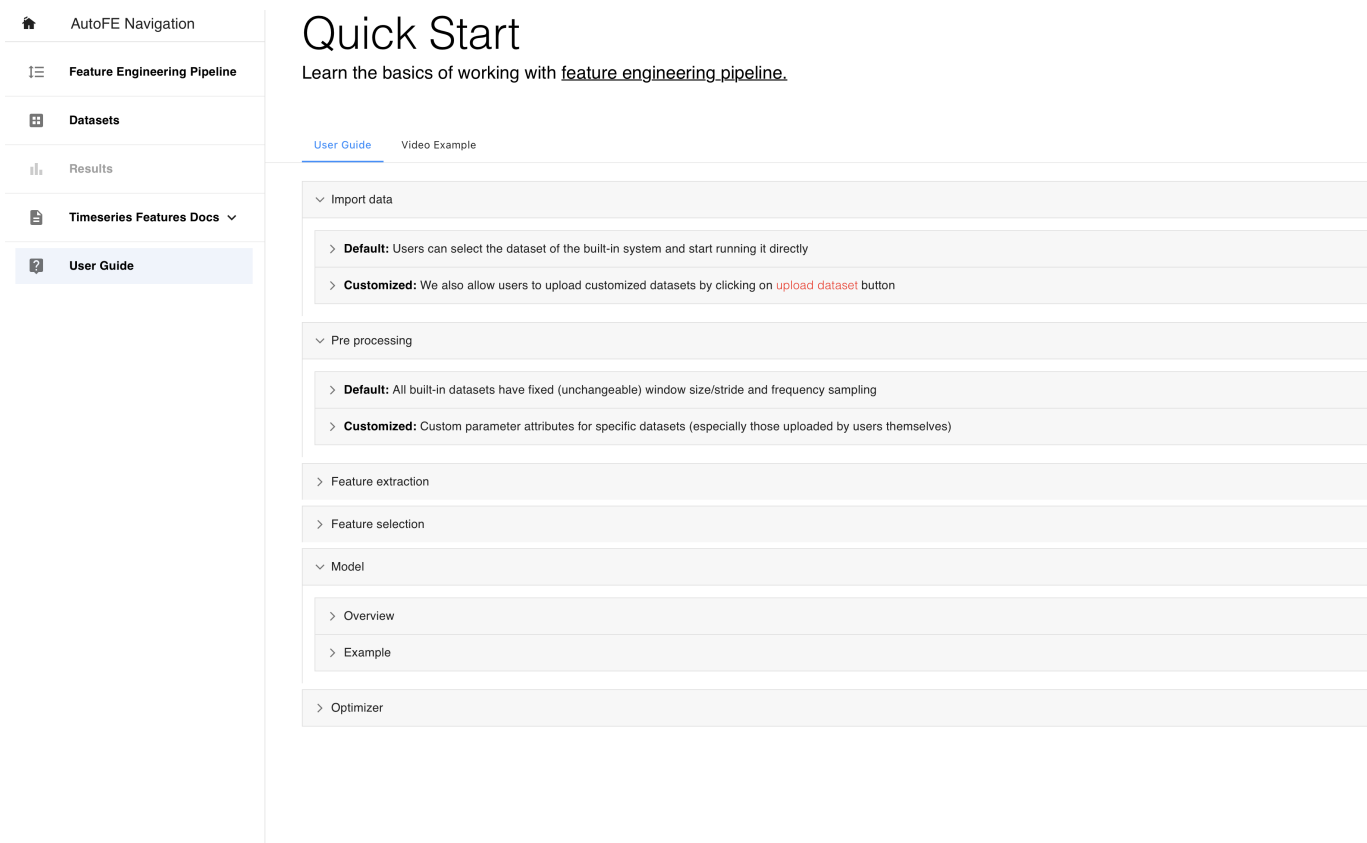


Figure 8.13: The initial panel of the user guide page, offering a comprehensive textual description for each pipeline step. Clicking on a step leads users to its corresponding documentation or functionality.

## Quick Start

Learn the basics of working with [feature engineering pipeline](#).

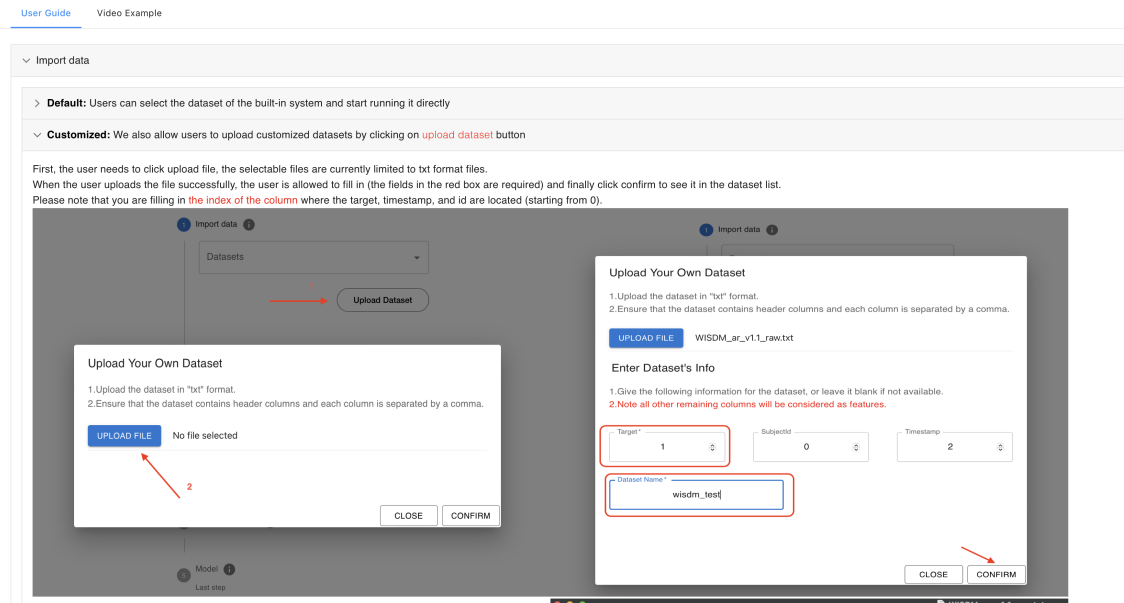


Figure 8.14: An example of user guide panel.

### 8.6.2 Video panel

The video covers every step, starting from “import data” to viewing the results upon completion. Essentially, it’s a comprehensive walkthrough of the full procedure.

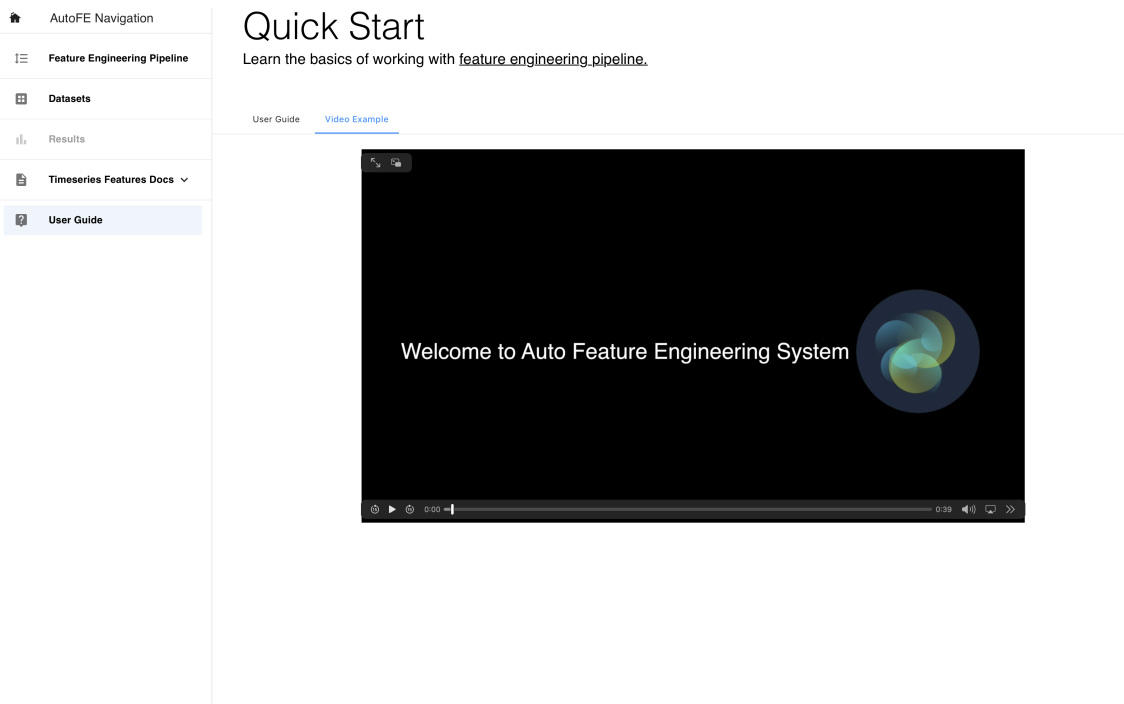


Figure 8.15: Video walkthrough from 'import data' to final result viewing on the user guide page.

## 8.7 Results page

On the Results page, we display the results derived from the backend execution, as illustrated in figure 8.16. This is divided into two sections. The first section explicitly shows the scores of the model, along with its parameters, which have been validated on the validation set, when applied to the test set. The second section, on the other hand, presents the optimization history and detailed optimization data.

We utilized the “ArticulatoryWordRecognition” [38] dataset along with randomly selected models, their parameters, feature extraction methods, and feature selection methods. We demonstrated the following 4 panels using 10 trial numbers.

### 8.7.1 Optimization history panel

In this panel 8.16, we primarily display images of the optimization history for each trial. It can be observed that for the trial indexed at 1 (indexing starts from 0), the performance on the validation set has already reached its peak. Subsequently, the model, its parameters, and feature selection method from this trial will be employed on the test set.

## Trial Study History

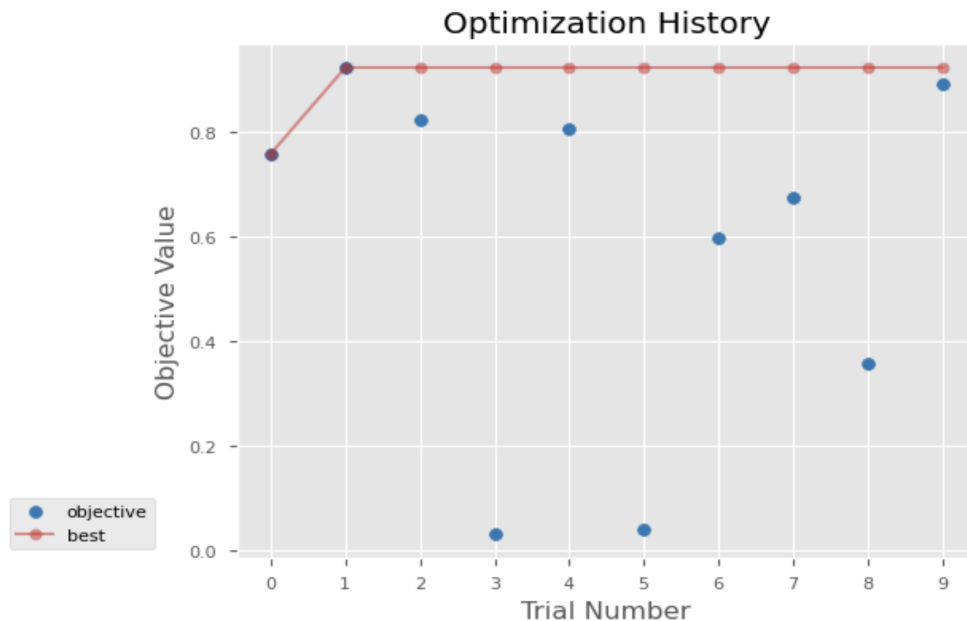
with RandomForestClassifier(criterion='entropy', max\_depth=3, n\_estimators=138, random\_state=42) and TSFRESH\_VAR2, the macro average f1 score in test set is : **0.9209891776183007**

optimization history for each trial

model hyper parameter importances

feature importances

trial study history (parameter settings)



Optimization history for each trial

- The blue sticker represents the f1 macro score of the current trial
- The red sticker represents the best f1 macro score up to the current trial(inclusive)
- If the red sticker and blue sticker overlap, the current trial is the best score

Figure 8.16: Optimization history panel highlighting the peak performance at trial number 1.



## 8.7.2 Model hyperparameter importances panel

This panel is primarily designed to highlight the importance of hyperparameters for a singular model. If multiple models are being optimized simultaneously, then the significance of this chart diminishes. However, when optimizing only a single model, one can discern the specific importance of each parameter within the model as illustrated in right figure 8.17.

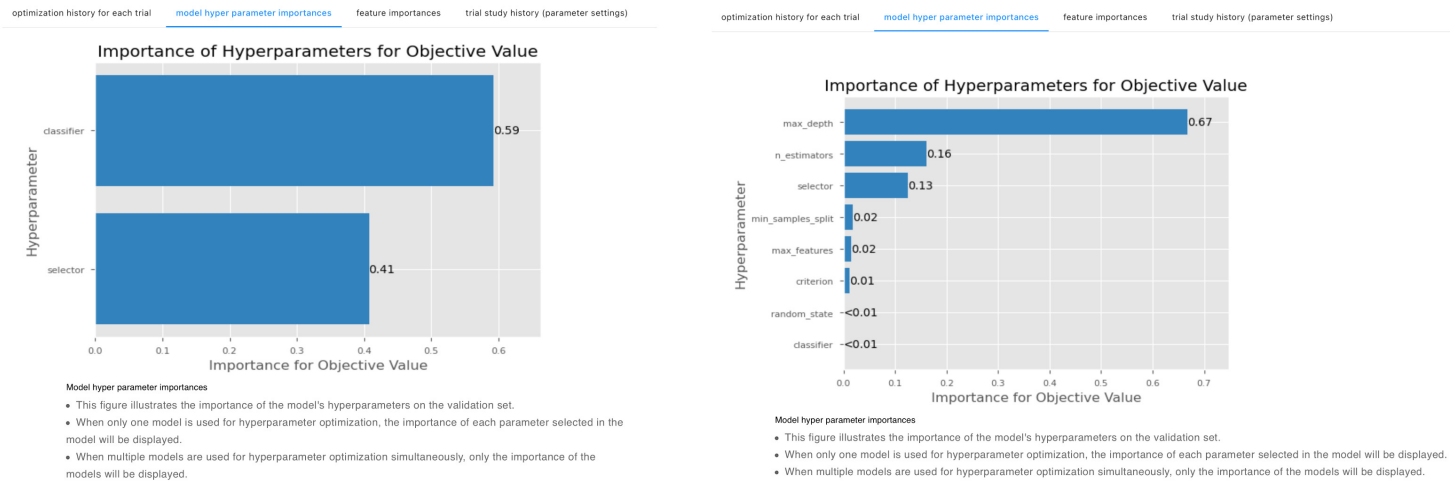


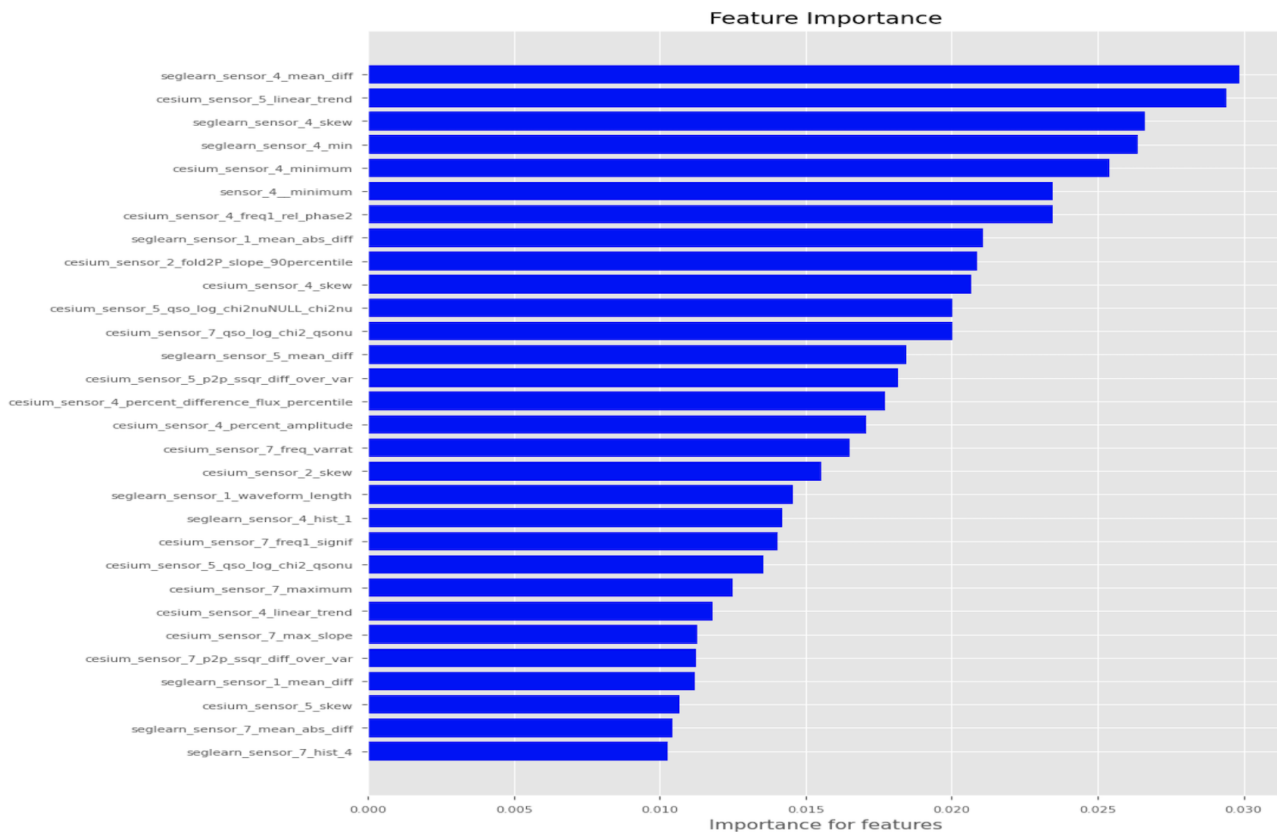
Figure 8.17: Hyperparameter importances in multi vs. singular model optimisation.

### 8.7.3 Feature importances panel

In this panel, our primary objective is to illustrate the importance of up to the top 20 features, specifically in relation to the test set. For the random forest, the features' importance is ranked directly based on the tree models. In the case of MLP and KNN, we employed a permutation feature importance method, using a 5-fold cross-validation, to rank the features. For the logistic regression model, we used coefficient weights to order the feature importance.

Permutation feature importance is a technique wherein the importance of a feature is determined by evaluating model performance on data where the said feature's values have been randomly shuffled. If the model's performance drops significantly upon shuffling, this indicates that the feature in question holds significant importance. By employing a 5-fold cross-validation alongside, we ensure the robustness of our ranking. On the other hand, for logistic regression, the magnitude of each feature's coefficient in the model provides a direct measure of its importance.

optimization history for each trial    model hyper parameter importances    **feature importances**    trial study history (parameter settings)



#### Features importances

- Feature importances provide insights into the relative significance of features used by a model for making predictions.
- They offer a quantitative measure, typically derived from the trained model itself, indicating how much each feature contributes to the model's predictions.

Figure 8.18: An example of a feature importances panel focusing on the top 20 features.

### 8.7.4 Trial study history panel

This panel serves as a detailed supplement to panel 8.7.1. The figure 8.19 displays the specific models used, their respective parameters, and the chosen feature selection methods for all trials, providing a comprehensive reference for users.

optimization history for each trial    model hyper parameter importances    feature importances    trial study history (parameter settings)

In the table below you can see the learning process of AutoML.

- For some unfilled parameters, it does not exist for the current model.
- The scores of each trial in the table are based on the results of the train-validation set.
- It is supported to sort the scores.

No.trial	f1_macro_score ↓	params_classifier	params_selector	params_C	params_active
1	0.9231238095	RandomForestClassifier	TSFRESH_VAR2		
9	0.8901714286	RandomForestClassifier	TSFRESH_VAR1		
2	0.8224761905	RandomForestClassifier	TSFEL		
4	0.8064761905	KNeighborsClassifier	TSFRESH_VAR1		
0	0.7581079365	RandomForestClassifier	TSFRESH_VAR1		
7	0.6735238095	LogisticRegression	TSFRESH_VAR2	1.0535087639	
6	0.5962031746	RandomForestClassifier	TSFRESH_VAR2		
8	0.3561904762	KNeighborsClassifier	TSFRESH_VAR1		
5	0.0404634921	MLPClassifier	TSFEL		relu
3	0.0299809524	KNeighborsClassifier	TSFEL		

1-10 of 10 < >

Figure 8.19: Detailed trial study history panel showing specific models, parameters, and feature selection methods for each trial.



# 9. Evaluation

## 9.1 Hypothesis

Our framework provides 5 different open source feature extraction packages, as we mentioned in the chapter 4.

As each feature extraction package has its own different focus. We thus want to verify whether combining multiple feature extraction packages outperforms using only one package (baseline)

In other words, we think that multiple feature extraction packages can provide more informative features.

## 9.2 Metric

The metric we employed is based on the content discussed in section 5.5.2, i.e., the macro average F1-score.

## 9.3 Datasets

To evaluate the effectiveness of our hypothesis, we conducted experiments using 15 datasets from a broad range of domains:

- **UCR <sup>1</sup> time Series classification archive:**
  - **Human Activity Recognition(HAR):** BasicMotions, Cricket, Epilepsy, ERing, Libras, RacketSports, UWaveGestureLibrary
  - **Motion:** ArticularyWordRecognition, CharacterTrajectories
  - **Electrocardiogram(ECG):** StandWalkJump
  - **Electroencephalogram/Magnetoencephalography(EEG/MEG):** SelfRegulationSCP1, SelfRegulationSCP2, HandMovementDirection

---

<sup>1</sup><http://www.timeseriesclassification.com/dataset.php>

- **Audio Spectra:** SpokenArabicDigits
- **Others:** EthanolConcentration

As for the selection of datasets, we did not have any specific criteria. Due to the constraints of local computing resources and time, we were unable to process very large datasets. Table 9.1 presents the details of each dataset we used.

Table 9.1: A summary of the used datasets in the UCR Multivariate Time Series Classification archive.

ID	Dataset	Train Cases	Test Cases	Dimensions	Length	Classes
1	ArticularyWordRecognition	275	300	9	144	25
2	BasicMotions	40	40	6	100	4
3	CharacterTrajectories	1422	1436	3	182	20
4	Cricket	108	72	6	1197	12
5	Epilepsy	137	138	3	206	4
6	ERing	30	30	4	65	6
7	EthanolConcentration	261	263	3	1751	4
8	HandMovementDirection	320	147	10	400	4
9	Libras	180	180	2	45	15
10	RacketSports	151	152	6	30	4
11	SelfRegulationSCP1	268	293	6	896	2
12	SelfRegulationSCP2	200	180	7	1152	2
13	SpokenArabicDigits	6599	2199	13	93	10
14	StandWalkJump	12	15	4	2500	3
15	UWaveGestureLibrary	120	320	3	315	8

## 9.4 Design of experiments

For each feature extraction package (tsfresh, tsfel, cesium, kats, seglearn), we conducted individual extractions to serve as our baseline.

For our experimental group, we initially used combinations of feature extraction packages taken two at a time, excluding tsfresh. This exclusion was due to the extensive number of features extracted by tsfresh, which could potentially lead to issues of overfitting (see section 4.2.1) and the curse of dimensionality (see section 4.2.2).

We carried out 100 trials using all the model parameters that have been pre-defined in the frontend, testing their macro average F1-score.

For model hyperparameter details:

- **Random forest:**
  - `n_estimators`: [50, 200]
  - `max_depth`: [1, 10]
  - `min_samples_split`: [2, 5]
  - `criterion`: [“gini”, “entropy”, “log\_loss”]
  - `max_features`: [“sqrt”, “log2”]
  - `random_state`: 42

- KNN:
  - `n_neighbors`: [1, 100]
  - `p`: [1, 5]
  - `weights`: [“uniform”, “distance”]
- Logistic regression:
  - `C`: [0.5,1.5]
  - `tol`: [0.0001, 0.0005]
  - `penalty & solver`:
    - \* (l2, lbfgs)
    - \* (None, lbfgs)
    - \* (None, newton-cg)
    - \* (None, newton-cholesky)
    - \* (None, sag)
    - \* (None, saga)
    - \* (l1, liblinear)
    - \* (l1, saga)
    - \* (l2, liblinear)
    - \* (l2, newton-cg)
    - \* (l2, newton-cholesky)
    - \* (l2, sag)
    - \* (l2, saga)
- MLP:
  - `hidden_layer_size`: Consists of 1 to 5 hidden layers. The number of neurons in each layer is formed as an arithmetic sequence from the input layer to the output layer.
  - `activation`: [“relu”, “logistic”, “tanh”]
  - `solver`: [“adam”, “lbfgs”, “sgd”]

### 9.4.1 Benchmark results

As can be observed in table 9.2, except for the datasets marked with (\*), our combination of two feature extraction packages did not surpass the baseline. However, for the majority of datasets, our hypothesis holds true. Even on some datasets, the scores combining the two feature extraction packages were consistently higher than all baseline scores.

Table 9.2: Benchmark classification results (in terms of macro average F1-score) for the baseline and combined tuple feature extraction packages (without tsfresh) approaches of each dataset. The (potentially tied) best score achieved for a dataset is in bold.

ID	Baseline					Combined packages						
	Tsfresh	Tsfel( $T_2$ )	Cesium	Seglearn	Kats	$T_2C$	$T_2S$	$T_2K$	CS	CK	SK	
1	0.969	0.970	0.973	0.966	<b>0.980</b>	<b>0.990</b>	0.977	0.986	0.983	0.980	<b>0.990</b>	
2	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	0.975	<b>1.0</b>	
3	<b>0.982</b>	0.974	0.970	0.949	0.971	0.977	<b>0.981</b>	0.975	0.959	0.977	0.974	
4	<b>0.972</b>	<b>0.972</b>	<b>0.972</b>	<b>0.972</b>	0.957	0.972	<b>0.986</b>	0.972	<b>0.986</b>	<b>0.986</b>	0.972	
5	0.986	0.986	0.957	0.958	<b>1.0</b>	0.972	0.979	0.993	0.965	<b>1.0</b>	<b>1.0</b>	
6*	<b>0.981</b>	0.899	0.903	0.903	0.861	0.903	0.836	0.891	0.914	<b>0.941</b>	0.923	
7	0.209	<b>0.228</b>	0.167	0.169	0.167	0.203	<b>0.232</b>	0.194	0.169	0.182	0.189	
8	0.271	<b>0.313</b>	<b>0.313</b>	0.293	0.285	0.279	<b>0.314</b>	<b>0.314</b>	0.264	0.251	0.237	
9*	<b>0.866</b>	0.778	0.796	0.830	0.848	0.786	0.777	0.810	0.803	<b>0.841</b>	0.821	
10*	<b>0.883</b>	0.872	0.802	0.805	0.740	0.818	0.856	0.848	<b>0.865</b>	0.828	0.804	
11*	<b>0.880</b>	0.826	0.750	0.797	0.748	0.823	0.819	0.823	<b>0.836</b>	0.775	0.771	
12	0.494	<b>0.645</b>	0.472	0.547	0.522	0.635	<b>0.645</b>	0.640	0.477	0.465	0.477	
13	0.453	<b>0.885</b>	0.802	0.819	0.798	<b>0.919</b>	0.879	0.878	0.859	0.877	0.827	
14	0.3892	0.280	0.278	0.328	<b>0.400</b>	0.275	0.265	<b>0.463</b>	0.207	0.172	0.275	
15*	<b>0.807</b>	0.724	0.690	0.767	0.776	<b>0.789</b>	0.749	0.748	0.768	0.768	0.786	



To further validate our hypothesis, we noticed that the datasets which did not meet the criteria had the highest scores from `tsfresh`. This might be due to the other feature extraction packages not extracting enough informative features, making it difficult to achieve comparable scores regardless of the combinations used. Therefore, in this step, we also included `tsfresh` in our combinations and conducted the same experiments. The results can be seen in table 9.3.

Now, as you can see, all tuples of feature extraction packages that included `tsfresh` performed better than `tsfresh` on its own and also surpassed the other baseline packages.

Table 9.3: Benchmark classification results (in terms of macro average F1-score) for the baseline and combined tuple feature extraction packages (with tsfresh) approaches of each dataset. The (potentially tied) best score achieved for a dataset is in bold.

ID	Baseline					Combined packages				
	Tsfresh( $T_1$ )	Tsfel( $T_2$ )	Cesium	Seglearn	Kats	$T_1T_2$	$T_1C$	$T_1S$	$T_1K$	
6	<b>0.981</b>	0.899	0.903	0.903	0.861	0.948	<b>0.985</b>	0.959	0.955	
9	<b>0.866</b>	0.778	0.796	0.830	0.848	0.849	<b>0.877</b>	0.866	0.860	
10	<b>0.883</b>	0.872	0.802	0.805	0.740	<b>0.914</b>	0.902	0.890	0.902	
11	<b>0.880</b>	0.826	0.750	0.797	0.748	0.873	0.874	<b>0.891</b>	0.877	
15	<b>0.807</b>	0.724	0.690	0.767	0.776	0.771	<b>0.820</b>	0.813	0.788	

# 10. Conclusion and Future Work

## 10.1 Conclusion

This is our first attempt at building a web-based, modular feature engineering pipeline. By refining each small step in the pipeline, users can select the feature extraction packages they need (or combine multiple packages), choose their preferred feature selection method, and specify the hyperparameters for each model. The program can then automatically execute the specified number of trials. Users simply need to wait a while and they can see the visualized results directly on the web page.

We provide essential user interaction. For example, if users are unfamiliar with the built-in datasets in the system, they can directly click on the “datasets” button in the sidebar to view detailed information. In addition, we offer all the specific features about the feature extraction methods from the source document for user inspection, as shown in chapter 4. We also allow users to upload their own datasets for feature engineering (currently, only those with an attribute type of numerical are supported).

## 10.2 Future work

Currently, the entire framework runs locally, which can be time-consuming when extracting a large number of features, especially for high-dimensional time series data. If the framework could be set up on a server, it could leverage the server’s computational power to significantly enhance computational efficiency. This would necessitate the construction of a cloud computing architecture.

As mentioned in our “Related Work” (see section 6), we plan to incorporate additional feature selection methods to effectively choose more informative features, thus avoiding phenomena such as overfitting (see section 4.2.1) or the curse of dimensionality (see section 4.2.2). Hyperparameter optimisation in our model is not limited to Bayesian Optimisation; we aim to implement a broader array of optimisation methods for users to choose from.

In terms of the model itself, our framework currently supports 4 different models, but there is room for expansion. In the future, variants of models such as Support Vector Machines (SVMs) [31], Long Short-Term Memory Networks (LSTMs) [32], and Transformers [37, 41] can be added, thus expanding the applicability and versatility of our system.

At present, our framework does not implement a database. For the interactive web interface with users, our web pages could offer more efficient interactions. For example, each user could have their own account with login credentials, granting access to a personalized interface containing frequently used datasets. The system could also feature functionalities such as sending notifications via email.

# Bibliography

- [1] L. Breiman et al., “Statistical modeling: The two cultures (with comments and a rejoinder by the author),” *Statistical science*, vol. 16, no. 3, pp. 199–231, 2001.
- [2] D. K. Wind. Concepts in predictive machine learning. Master’s thesis, Technical University of Denmark, 2014.
- [3] U. Khurana, D. Turaga, H. Samulowitz and S. Parthasarathy, ”Cognito: Automated Feature Engineering for Supervised Learning,” 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), Barcelona, Spain, 2016, pp. 1304-1307, doi: 10.1109/ICDMW.2016.0190.
- [4] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, *Time Series Analysis: Forecasting and Control*, fifth ed., John Wiley Sons, Hoboken, New Jersey, 2016.
- [5] M. Christ, N. Braun, J. Neuffer, A.W. Kempa-Liehr, Time series Feature extraction on basis of scalable hypothesis tests (tsfresh – A Python package), *Neurocomputing* 307 (Sep. 2018) 72–77, doi:10.1016/J.NEUCOM.2018.03.067.
- [6] Yao Q, Wang M, Chen Y, et al. Taking human out of learning applications: A survey on automated machine learning[J]. arXiv preprint arXiv:1810.13306, 2018.
- [7] E. Brochu, V. M. Cora, and N. D. Freitas. (2010). A tutorial on Bayesian optimization of expensive cost functions,with application to active user modeling and hierarchical reinforcement learning.  
<https://arxiv.org/pdf/1012.2599.pdf>.
- [8] Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., Deng, S.-H., 2019. Hyper-parameter optimization for machine learning models based on bayesian optimizationb. *J. Electro. Sci. Technol.* 17 (1), 26–40.  
<http://dx.doi.org/10.11989/JEST.1674-862X.80904120>.
- [9] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’19)*. Association for Computing Machinery, New York, NY, USA, 2623–2631.  
<https://doi.org/10.1145/3292500.3330701>.
- [10] Yahmed,Y.B., Bakar.A.a., RazakHamdan,A., Ahmed, A., Abdullah, S.M.S.(2015). Adaptive sliding window algorithm for weather data segmentation. *Journal of Theoretical and Applied Information Technology*, 80(2), 322-333.

- [11] M. Barandas, D. Folgado, Fernandes, L, et al., “TSFEL: Time Series Feature Extraction Library,” *SoftwareX* 11, 2020.
- [12] Bellman, R.: *Adaptive Control Processes: A Guided Tour*. Princeton University Press (1961).
- [13] Verleysen, Michel François, Damien. (2005). *The Curse of Dimensionality in Data Mining and Time Series Prediction*. *Lecture Notes in Computer Science*. 3512. 758-770. 10.1007/11494669\_3..
- [14] K. Taunk, S. De, S. Verma and A. Swetapadma, ”A Brief Review of Nearest Neighbor Algorithm for Learning and Classification,” 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 1255-1260, doi: 10.1109/ICCS45141.2019.9065747.
- [15] Peng, Joanne Lee, Kuk Ingersoll, Gary. (2002). *An Introduction to Logistic Regression Analysis and Reporting*. *Journal of Educational Research - J EDUC RES*. 96. 3-14. 10.1080/00220670209598786.
- [16] Breiman, L. *Random Forests*. *Machine Learning* 45, 5–32 (2001). <https://doi.org/10.1023/A:1010933404324>.
- [17] Song, Q., Liu, X., Yang, L.: *The random forest classifier applied in droplet fingerprint recognition*. In: 2015 12th International Conference on FSKD, pp. 722–726. IEEE, August 2015.
- [18] Parmar, A., Katariya, R., Patel, V. (2019). *A Review on Random Forest: An Ensemble Classifier*. In: Hemanth, J., Fernando, X., Lafata, P., Baig, Z. (eds) *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*. ICICI 2018. *Lecture Notes on Data Engineering and Communications Technologies*, vol 26. Springer, Cham. [https://doi.org/10.1007/978-3-030-03146-6\\_86](https://doi.org/10.1007/978-3-030-03146-6_86).
- [19] Popescu, Marius-Constantin Balas, Valentina Perescu-Popescu, Liliana Mastorakis, Nikos. (2009). *Multilayer perceptron and neural networks*. *WSEAS Transactions on Circuits and Systems*. 8.
- [20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [21] David Harbecke, Yuxuan Chen, Leonhard Hennig, and Christoph Alt. 2022. *Why only Micro-F1? Class Weighting of Measures for Relation Classification*. In *Proceedings of NLP Power! The First Workshop on Efficient Benchmarking in NLP*, pages 32–41, Dublin, Ireland. Association for Computational Linguistics.
- [22] H. Zou, T. Hastie, *Regularization and variable selection via the elastic net*, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2) (2005) 301–320.
- [23] R. Tibshirani, *Regression shrinkage and selection via the lasso*, *Journal of the Royal Statistical Society: Series B (Methodological)* 58 (1) (1996) 267–288.
- [24] Yao, Quanming Wang, Mengshuo Escalante, Hugo Jair Isabelle, Guyon Hu, Yi-Qi Yu-Feng, Li Tu, Wei-Wei Qiang, Yang Yang, Yu. (2018). *Taking Human out of Learning Applications: A Survey on Automated Machine Learning*.

- [25] Yu, Yang Qian, Hong Hu, Yi-Qi. (2016). Derivative-Free Optimization via Classification.
- [26] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 1998.
- [27] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.
- [28] A. Gyoergy and L. Kocsis, “Efficient multi-start strategies for local search algorithms,” *Journal of Artificial Intelligence Research*, vol. 41, pp. 407–444, 2011.
- [29] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.
- [30] David Burns and Cari Whyne. “Seglearn: A Python Package for Learning Sequences and Time Series”. In: *Journal of Machine Learning Research* 19 (Mar. 2018).
- [31] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20 (1995), pp. 273–297.
- [32] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [33] Jundong Li et al. “Feature selection: A data perspective”. In: *ACM computing surveys (CSUR)* 50.6 (2017), pp. 1–45.
- [34] Brett Naul et al. “cesium: Open-source platform for time-series inference”. In: *arXiv preprint arXiv:1609.04504* (2016).
- [35] Juri Opitz and Sebastian Burst. “Macro F1 and Macro F1”. In: *ArXiv abs/1911.03347* (2019).
- [36] Ping Tan, Xin Wang, and Yong Wang. “Dimensionality reduction in evolutionary algorithms-based feature selection for motor imagery brain-computer interface”. In: *Swarm and Evolutionary Computation* 52 (2020), p. 100597. ISSN: 2210-6502.
- [37] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [38] Jun Wang et al. “Word Recognition from Continuous Articulatory Movement Time-series Data using Symbolic Representations”. In: *Proceedings of the Fourth Workshop on Speech and Language Processing for Assistive Technologies*. Grenoble, France: Association for Computational Linguistics, Aug. 2013, pp. 119–127.
- [39] Xue Ying. “An Overview of Overfitting and its Solutions”. In: *Journal of Physics: Conference Series* 1168.2 (2019), p. 022022.
- [40] Shaode Yu et al. “Elastic Net based Feature Ranking and Selection”. In: *ArXiv abs/2012.14982* (2020).
- [41] Haoyi Zhou et al. “Expanding the prediction capacity in long sequence time-series forecasting”. In: *Artificial Intelligence* 318 (2023), p. 103886. ISSN: 0004-3702.