TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Pattern Detection in Time-Series Satellite Data

Authors	Lukas Dreier, Lucas Lincoln, Markus Steinbach
Mentor(s)	Leonard Schlag, Dr. Clemens Schefels
	DLR
Co-Mentor	M.Sc. Laure Vuaille
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2019

Abstract

Numerous manned and unmanned satellites are under the auspices of the German Aerospace Center (DLR). These are supported by teams of ground-engineers who are responsible for monitoring and evaluating signals from afar for performance, system health, and analysis. A key challenge in this work is the amount of data - satellites may send tens of thousands of independent signals, too much for manual inspection. In this work we describe the development of a software system which enables an engineer to select an arbitrary region of data that contains some pattern of interest and be presented with periods of time from historical data in which the system experienced a similar pattern. The software has been developed in a modular fashion in the Python language; 5 unique algorithms for pattern matching and one ensemble method are included; and evaluation of performance is done using the receiver operating characteristic and area-under-curve (AUROC) measure. Performance of the tool on most patterns is adequate with an AUROC in the range 0.60 - 0.80; though for some patterns performance is on par with a random classifier and these challenges will require further algorithmic improvements. The system was designed to be robust to signal types, incomplete coverage, and inconsistent sampling, common issues with telemetry data. Insight from a parameter study of the matching algorithms is presented.

Contents

Abstract **1**

1 Introduction **3**
1.1 Formal Problem . . . 4
1.1.1 Similarity . . . 4
1.2 Distance Measures and Match Quality . . . 5
1.2.1 $\mathcal{L}_p$-type measures . . . 5
1.2.2 Match Quality . . . 6
1.2.3 Probabilistic distance and match quality . . . 6

2 Statistical Data Exploration **7**
2.1 Data Acquisition . . . 7
2.2 Data Cleansing . . . 7
2.3 Statistical Data Exploration . . . 7
2.4 Identified Challenges . . . 8
2.4.1 Testcase Tooling . . . 8

3 Approach **9**
3.1 Preprocessing Methods . . . 10
3.2 Search Methods . . . 11
3.3 Postprocessing Methods for Match-lists . . . 11
3.4 Algorithms . . . 12
3.5 Distance . . . 12
3.5.1 Wavelet . . . 12
3.5.2 DTW . . . 13
3.5.3 APCA . . . 13
3.6 Probabilistic . . . 13
3.6.1 Ensemble . . . 14
3.7 Performance Evaluation . . . 15
3.8 Program Structure . . . 16
3.8.1 Modular Design . . . 16
3.8.2 Coding Style . . . 17
3.8.3 Formal Interfaces . . . 17
3.8.4 End-user GUI . . . 17
3.8.5 Documentation . . . 17
3.9 Computational Experiments . . . 17
3.9.1 Parameter Study . . . 18
3.9.2 Performance Study . . . 19

4 Results and Discussion **19**

5 Conclusions and Future Work **21**

1 Introduction

The German Aerospace Center (DLR) is responsible for the safe and effective operation of a fleet of manned- and unmanned-satellites serving a wide variety of purposes. These satellites have differing topologies, sensing hardware, diagnostic abilities, and flight paths - a commonality in this generally variant fleet is the transmission of data to earth for ground-based system engineers to perform satellite system operations including system health monitoring, anomaly detections, predictive analysis, and performance evaluation.

The goal of this project was to establish a software suite which enables the system engineers to more efficiently perform the aforementioned duties, specifically by providing a system which enables automatic recognition of events in the historical dataset which are substantially similar to a manually selected event. This system is considered a *pattern matching* system; wherein a user may manually select a period of time-series data which is considered novel by their expert opinion and intuition but that may not be an error, system failure, or other significant anomaly. In this sense the software serves as a powerful data exploration tool for datasets which are too large for manual inspection.

Key challenges in this development are:

- **A large number of parameters**. Each satellite may provide tens of thousands of different time-series data streams.
- **An inhomogeneous dataset**. Each parameter from a satellite may have different sampling rates, resolutions, types, or other characteristics. Additionally, different satellites may have a different set of parameters; and satellites reporting the same parameters may do so inconsistently (i.e: the resolution and sampling rate of parameter α from Satellite A is not, in general, the resolution and sampling rate of parameter α from Satellite B).
- **An inhomogeneous operating environment**. The satellites experience a wide variety of conditions - there are regular periodic and diurnal variations as well as irregular variations due to solar activity, weather, sensor and power conditions, and other factors. There may also be updates to the system firmware, software, and operating modes over time. In practice this means that each period in the database matching a novel period λ may have different sampling rates; data representations; noise characteristics; bias; and coverage (data may be incomplete.)

These points were the focus for the team both in research and selection of appropriate methods as well as development of the tool and in its evaluation. The system should be robust to the types of changes identified above.

A summary of the system scope follows:

1. The system will take as input a period of data of a user selected duration from a single parameter, the start time and end time of the period determined by the user.
2. The system will output, based upon the input novel period, a list of periods which are similar to this novel period based on some metrics. The returned periods should be identifiably similar to the input period by human inspection.

3. The system should be written in Python 3 to suit the DLR working environment and users. If non-Python portions are required for performance or other reasons, they shall be wrapped in Python for ease of use.

4. The system should be performant on consumer grade computing hardware. Target operation is a modern laptop. Target performance to return a search is on the order of minutes for a dataset on the order of 1 GB per parameter.

5. Users are technically proficient personnel. The system's methods should be understandable to such a userbase and not appear to said personnel as a *black box*. With this in consideration, the system should utilize classical machine learning approaches and not deep learning or neural network approaches.

6. The system resulting from this project is targeting future adoption into the DLR workflow which has an existing UI, plotting system, and database access systems. The system resulting from this project should be well documented and follow modern Python coding practices to allow future incorporation into DLR systems.

1.1 Formal Problem

We formalize our problem as follows:

Given a sequence of N value measurements $\{x_1, x_2, ..., x_n\} \in \mathbb{R}^N$ and an associated sequence of N **strictly** monotonically increasing time values $\{t_1, t_2, ..., t_n\}$, the sequence of N tuples $D := \{(t_i, x_i)\}$ is called a *time series* and can be partitioned into (potentially overlapping) compact subsets arbitrarily. We call this set of all possible compact subsets $\mathbf{A}$.

We may further define $\mathbf{A_p}$ as the set of all compact subsets in $\mathbf{A}$ of duration p.

The most general problem is, given a user-selected query subset $Q \in \mathbf{A}$, find the most similar matching subsets from $\mathbf{A}$, where similarity is best defined by human intuition considering the question *are these two sequences similar?*.

In reality, there are a number of simplifications to the problem, some of which are suitable for our application and some of which are not. We will define those simplifications below.

We say the time series is **consistently sampled** if $t_{k+1} - t_k$ is constant $\forall k \in [1, N]$, and inconsistently sampled otherwise.

We will also say the time series has **incomplete coverage** if it is otherwise consistently sampled but is missing periods of data. In general, data from satellites may be consisent- or inconsisently sampled; and often has incomplete coverage.

Though some intution of a *similar* pattern would allow patterns to be streched or squeezed to different durations, we will only investigate **strict-duration** queries, in which we attempt to match regions of the same duration as the query. That is to say: Given a subset $Q \in \mathbf{A}$ of length l, we perform a **strict-duration** query if we seek matching subsets only in $\mathbf{A_l}$, that is, in regions of equal length to the query Q.

1.1.1 Similarity

We wish to return the most-similar regions from our database to a query sequence Q. What makes a region similar? Is a region I similar to query Q? Unfortunately the answer

often is: *I'll know when I see it.* However, because a user cannot review every possible region manually, we must develop measures of similarity which are seemingly consistent with user *intuition* of similarity.

Similarity can mean different things to different people, or even to the same person at different times, and we must remain aware of these facts as we attempt to define some measures Δ such that a sequence S is **similar to** Q if $\Delta(\vec{Q}, \vec{S}) < \epsilon$ for some ϵ

The challenge here is twofold: We must determine what an appropriate measure is Δ, and in addition we must have a *match quality* in order to rank the matches; allowing the return of only the most-similar matches. This **match quality** of (Q, S) is the result of a function $\eta_\Delta(Q, S) \in [0, 1]$ which should have the following properties:

1. $\eta_\Delta(Q, Q) \approx 1$
2. $\eta_\Delta(Q, S) > \eta_\Delta(Q, T)$ if Q is more similar to S than it is to T.

we must determine the appropriate ϵ and quality mapping η_Δ for measure Δ such that only intuitively similar sequences achieve a high match quality. First, we will discuss similarity measures Δ.

1.2 Distance Measures and Match Quality

In general, we have two types of similarity measures to consider: $\mathcal{L}_p$ type distance measures and probabilistic measures.

1.2.1 $\mathcal{L}_p$-type measures

A common measure of the distance (or inversely, similarity) between two signals is an $\mathcal{L}_p$ norm:

$$\Delta_{\mathcal{L}_p}(\vec{x}, \vec{y}) = \left(\Sigma_{i=1}^{N} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Though there is potential value to using an $\mathcal{L}_1$ norm for robustness against impulsive noise, the $\mathcal{L}_2$ Euclidean distance is optimal for Gaussian, i.i.d. measurement error [18]. Given the popularity of the Euclidean distance in time series matching literature and provided it's optimality, we selected to use it throughout our distance-based approaches.

We consider in all cases a bias-compensated Euclidean distance, in which we compare the query (Q) and sequence in question (S) after compensating each by their mean values:

$$\widetilde{\Delta}_x(Q, S) = \Delta_x(Q - Q_{avg}, S - S_{avg})$$

with

$$Q_{avg} = \frac{\Sigma_{i=1}^{N} Q_i}{n}, \quad S_{avg} = \frac{\Sigma_{i=1}^{N} S_i}{n}$$

and Q_i, S_i the ith component of Q and S, respectively.

1.2.2 Match Quality

The bias compensated $\mathcal{L}_2$ measure provides an unbounded measure of dissimilarity. Practically, we wish to work with a metric which has been normalized into a perfect score (1.0) for perfectly similar signals and a 0-score for completely non-matching signals. We have designed a mapping from Euclidean Distance to Match Quality which captures the intent of our solution.

We wish that a perfect signal - one identical to the query - equals 1.0 ($\eta_{\mathcal{L}_2}(Q, Q) = 1.0$). The Euclidean distance between these signals is 0 ($\Delta_{\mathcal{L}_2} = 0$), therefore it is suggested to have a mapping of the form

$$\frac{f(Q) - \Delta_{\mathcal{L}_2}}{f(Q)}$$

with $f(Q)$ being some arbitrary function - examples may be the maximum or mean functions. We can select the most appropriate function using the knowledge of our end-use case.

There is in general no bound to the amount of distance a signal can be from the query signal, given that each signal has a different value range we cannot make assumptions about the maximum and minimum values of a signal. However, it is not required that a 0.0 be assigned only to the worst possible match; only that any match with a 0.0 would under no conditions be considered *similar* by a ground engineer. With consultation of the advisors at the DLR; we determined that a DC signal would be the worst possible match possibly still of interest. Phrasing that in a manner which we can translate to our mapping defines $f(Q)$ to be the norm of Q, and the final quality metric is:

$$\frac{(\Sigma|Q_i|^2)^{\frac{1}{2}} - (\Sigma|Q_i - S_i|^2)^{\frac{1}{2}}}{(\Sigma|Q_i|^2)^{\frac{1}{2}}}$$

This quality metric operates nicely in that it scales appropriately with signals of differing ranges. It is a measure of how much error between signals there is in relation to how large the signal itself is.

1.2.3 Probabilistic distance and match quality

In general we introduce how probabilistic models work in section 3.6. Nevertheless, we will introduce the similarity context in probabilistic terms. Assume we have two realizations which are drawn from a probabilistic model. We fit a model for one of these realizations with the maximum likelihood method. From this we get an estimation of the hyperparameters for the assumed probabilistic model. Next we wish to check whether the underlying model is also appropriate for other realizations. Typically this is done with a likelihood ratio test, but due to its complexity and computational cost we implemented a different but similar method.

In [15] the authors fit several models and calculate the respective likelihoods for queried time series. Afterwards they decide based on the likelihood of the fit model whether it is appropriate or not. In our case we calculate the likelihood for the second realization (region of interest) based on the query's underlying model. Now we compare the ratio of likelihoods determine if the assumption (that both realizations rely on the same underlying model) is true or not. This ratio of the likelihoods gives us a measure of similarity. If this

ratio is large than the model represents the second realization very well. If it is small the model fitted for the first realization does not represent the one in question.

This ratio is used directly as the match quality. The quality could exceed 1 since it is possible that the second realization is represented better by the model than the first one. In agreement with our DLR advisors we decided that this does not negatively affect our goal - it only indicates that the realization is represented very well by this underlying probabilistic model. We ultimately use as a match quality the maximum of the described ratio or 1.0.

2 Statistical Data Exploration

Prior to literature research or implementation of any algorithms we concentrated on familiarizing ourselves with the supplied data, and identifying its potential challenges. Furthermore, we used this phase to get an idea what are the challenges of the data. The exploration phase ended in the clustering in different test cases described in 2.4.

2.1 Data Acquisition

Data was acquired from the DLR in a convenient format. We were provided the time series data of 12 parameters as TSV files. The total file size is 6.2 GB, with a range of individual series from the smallest at 9.5 MB to the largest at 1.3 GB. Every file had a consistent structure.

The largest challenge with the dataset is it's regulatory restriction: we were requested not to transfer the data to any servers outside of our own personal laptops. Namely, cloud compute providers were unavailable for our project. The restriction is manageable as the data fit comfortably in memory on a personal computer. However, for parametric optimization of our algorithms we required sometimes hundreds of executions; as a result we utilized a publicly available dataset [5] of labeled time series data. A tool was written to map the data format of this dataset to our own format. With this data we are able to perform parametric explorations on LRZ cluster computer resources, take the ultimate findings for parameter ranges and apply them to the DLR dataset on our personal computers.

2.2 Data Cleansing

As the data was provided to us in the same format and condition as it will be accessed by ground engineers while using our tool; we did not perform additional data cleaning or filtering. This best represents an accurate end-use case.

2.3 Statistical Data Exploration

We utilized a standard toolbox of statistical KPIs and investigated the following for the values for every parameter: mean, median, standard deviation, minimum, maximum, number of observations and number of unique observations. Of course, every sampled time series has a discrete range of values. Nevertheless, we wished to distinguish between

continuous and *discrete* parameters. In cooperation with our advisors from the DLR, we defined a threshold of quantization levels to differentiate a continuous and discrete parameter.

Furthermore, we reviewed the data via auto correlation which is one the main tools for time series analysis. Autocorrelation plots allowed us to understand the structure of the different parameters, i.e. parameter 4 had a nearly perfect autocorrelation and therefore, we decided to use this testcase to test the algorithms whether they detect perfect data or not. These plots can be found in Milestone Presentation #1, and are omitted here for brevity.

The second focus in our data exploration phase was the structure of the difference between subsequent timestamps (the *sampling rate*). Irregular sampling rates can be a tremendous challenge for some algorithms, and sometimes force interpolation to a fixed-sampling series. Histograms showing the variability of each parameter can again be found in Milestone Presentation #1. Note that the tool was targeting not these specific datasets, but instead hopes to serve as a matching tool robust to different parameter characteristics: from this point of view it is sufficient to say, after reviewing the representative data, that we must account for nonuniform sampling and coverage.

2.4 Identified Challenges

All the analysis and visualizations of the time series led to six key challenges our algorithms will face:

Inconsistent sampling	Variable pattern duration
Instantaneous events	Low Pattern-to-noise-ratio
Discrete or continuous signals	Patterns characterized by frequency variation

Representative plots and further discussion of these challenges may be found in the Milestone 1 meeting presentation, and are omitted here for brevity.

2.4.1 Testcase Tooling

After qualitative exploration of the data, we required labels in order to evaluate the performance of each algorithm and compare various implementations. As we were provided unlabeled datasets without a testing framework, we were required to determine a framework for labeling and testing, and further to label each pattern manually. A complete suite of tooling around labeling data; defining tests; and evaluating performance was developed. Notable contributions are:

- A standardized, markup-language format for labeling data.
- A standardized, markup-language format for saving query results.
- A set of programs to automate testing of algorithms against labeled datasets using selectable measures.
- A QT-framework based GUI to aid in the manual labeling of datasets (see Fig 1)

Based on the aforementioned challenges, and using the developed framework, we labeled 776 regions over 26 testcase patterns from 12 parameters.

Figure 1: Screenshot of tool developed for testcase labeling, reviewing, and modifying

3 Approach

Following investigation of the representative datasets, the team performed a literature review to understand current best-practices in pattern matching; and classic solutions.

Based upon this review, we characterized approaches to time series matching that may be applicable to our task into three major areas:

- **Distance-based** approaches, and would include simple methods as subsampling/filtering and comparing the Euclidean distance as well as more complex routines such as Dynamic Time Warping [13] or comparing distance in a Wavelet subspace [8]. A particularly aggressive method simplifies the pattern both temporally (reducing from 100s of data points to 1-10s of segments) and with respect to value (by downsampling to a quantization with only a few levels; selected to maximize information fidelity given the compression) - this method is known as APCA [4].

- **Probabilistic** approaches, which represent the pattern as a model and measure the likelihood a given test range comes from the same model (such as Hidden Markov Models [6] or Gaussian Mixture Models [12])

- **Symbolic** approaches, which transform the time series to a string of symbols and compare string similarity to measure distance. These are frequently used in bioinformatics, language processing, and streaming and can be very fast given some prior knowledge of the dataset at hand [10] [3] [17]. Because we desire a robust, general matching system *without* prior training, we did not pursue symbolic approaches in this work.

We will discuss in further detail Dynamic Time Warping, Probabilistic Methods, and the APCA algorithm. First, we may further decompose a time-series matching task (including any of the aforementioned algorithms) into a set of subtasks. Though not a perfect taxonomy, it serves as a useful framework for both discussing the following work; and for structuring our software module.

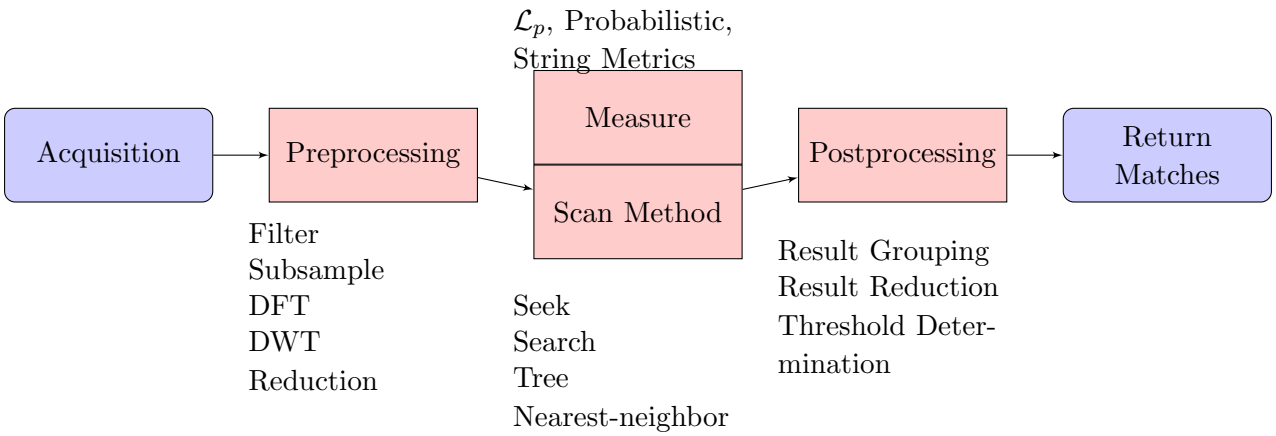

Figure 2: A breakdown of the subtasks of a time-series pattern matching task, with examples of potential methods for each stage

An overview of the time series matching is shown in Fig 2; with a few examples for each stage of the process. The Acquisition stage is outside the scope of our work - the DLR will ultimately pull data from their internal database. For our purposes, data is contained in provided TSV files, each containing a duration of data on the order of 1-2 years. The preprocessing, searching, measure, and postprocessing are now discussed.

The characterization of the different types of time series matching algorithms is mainly based on the different measures of similarity. The main idea is that you take a suitable measure for your representation of the time series. The overview of time series matching shows clearly that different stages can be combined in several ways, i.e. use a wavelet decomposition before you apply a probabilistic algorithm. This gives the user several possibilities to customize the algorithms.

One may notice that we neglected advanced machine learning methods like deep learning. This was one of the requirements of the DLR. They want to have a high explainability of the model which is not guaranteed in advanced methods. Approaches such as deep learning would have offered a bunch of new possibilities and can be studied in a separate project.

3.1 Preprocessing Methods

We consider preprocessing to include not only traditional data cleaning operations but also fundamental transformation to the data, and therefore part of the matching algorithm as a whole. Besides well-known preprocessing methods such as normalizing and standardizing, the transformation to a new representation such as the discrete wavelet transformation (DWT) or discrete Fourier transformation (DFT) are included in our preprocessing stage. The mapping to a representation as a probabilistic model is also contained here and it demonstrates that different approaches (like DTW and a probabilistic method) can be combined using our multi-staged approach.

3.2 Search Methods

There are two fundamental ways in which one can search through a time series data set to evaluate quality of each potential matching subset. First, let us define our terminology for this section.

Given the set of $\mathbf{A_p}$ compact subsets of length A from time series $\mathbf{D} := (t_i, x_i)$ (see Chapter 1.1), it is obvious to assert that each subset of A_p may be uniquely identified (indexed) by its lowest contained timestamp t_i and that we may sort them to be in monotonically increasing order in time :

$$\forall a_j, a_k \in A_p \quad j, k \in \mathbb{N}^+ \quad j < k : \min_{t_i} a_j < \min_{t_i} a_k$$

Note that this is not true for $\mathbf{A}$ in general, wherein elements must be identified by at least two components, for example starting time and length. This is a simplification made possible by restricting to a strict-duration query.

Given strict-duration query we say that we **search** for matches to query Q in $\mathbf{A_l}$ if we evaluate the match quality of at least a substantial subset of $\mathbf{A_l}$ against Q.

We **seek** for matches in $\mathbf{A_l}$ if we systematically evaluate $\eta_\Delta(Q, a_j)$ in monotonically increasing order - $\eta_\Delta(Q, a_{j+1})$ is evaluated after $\eta_\Delta(Q, a_j)$.

A seek can be performed in reverse order (newest timestamps to oldest) or trivially parallelised by partitioning the series into non-overlapping sections.

To note a few important distinctions. Seeking is clearly a type of search, and is the type utilized throughout this project. [1]

In general, we don't seek every possible region of length l in the time series (see Sec 3.7. Instead, we seek through the time series at a stride that is some fraction of the initial query length. This stride length is a parameter that can be adjusted. When this stride is fixed, we refer to it as a **fixed-walk**.

A **smart-walk** proceeds as a fixed-walk except after evaluations where $\eta_\Delta >$ thresh, wherein the stride length for the next evaluation increases to some value nx.

3.3 Postprocessing Methods for Match-lists

After a query has been performed, the result is a list of tuples, each tuple containing the range of a match exceeding the threshold and the quality of that match, η_Δ. The amount of these matches and the distribution can be a challenge for end users - it may not be useful to return 10 matches all overlapping the same pattern; the goal of the system is to use a single region tuple to alert the user to a single matching pattern.

This goal is complicated by the interplay in a variety of parameters: Lowering the match quality threshold will increase the number of regions returned; raising it may cause false negatives. Decreasing the stride length will increase the number of matches returned and result in many overlapping matches on a single pattern (especially for a time-scaling

[1] Promising methods of search that are not *seeking* take advantage of efficient datastructures (often trees or tree-like structures) and to allow faster retrieval of a match than a seek. A notable example are F-Trees, a type of R*-tree populated wherein levels are built based on the proximity of the subsets Fourier coefficients [1]. These are applicable in cases where we search many same-length patterns on one dataset; this is not our usecase.

Algorithm 1: Smart Walk

Input: Q query, $D := \{d_i\}$ data in closed subranges of length len(Q)
Parameter: X fixed stride length, Y stepover length, τ Quality Threshold
Output: Vector of match qualities $\vec{\eta}$

```
η⃗ ← list
i ← 1
do
    if η_Δ(Q, d_i) > τ then
        η⃗.append(η_Δ(Q, d_i))
        i ← i + Y
    else
        i ← i + X
    end
while Not at end of time series;
return η⃗;
```

measure such as DTW), increasing it introduces the risk that a pattern is missed because it doesn't align well in time with the stride period.

In an attempt to improve the user experience, we have designed two postprocessing routines and included them in our parametric studies of algorithm performance. In the first, *best-in-group* postprocessing, every contiguous set of matches (those whose endpoint overlaps the following startpoint) is reduced by selecting only the best match in that set. In the case that a single pattern of interest truly exists that group, the best match should be the most accurate pattern overlap.

The second method is *Combine-group* postprocessing, in which each contiguous group is reduced to a single match with a range starting at the minimum startpoint of the group and ending at the maximum endpoint of the group. The quality in this case is inherited from the best match within the group.

Finally, postprocessing of the type *none* is self explanatory - the raw list of returned regions from the algorithms is maintained.

3.4 Algorithms

Five algorithms were selected to be implemented and evaluated, with three as the focus. The algorithms were selected based on a desired for both variety (in strengths and weaknesses, as well as in fundamental method of use) and a high likelihood of success on pattern matching the supplied datasets. These algorithms include three from a *distance* category, and two from the *probabilistic* category.

3.5 Distance

3.5.1 Wavelet

Wavelet matching is implemented in a straightforward manner. The **preprocessing** stage consists of a Discrete Wavelet Transform using Haar wavelets at a depth of 4 levels. This

results in a set of coefficients in a generated wavelet basis. The **measure** is directly the Euclidean distance between these coefficients - that is, we measure the proximity in the decomposed wavelet basis, rather than reconstructing the time series from the truncated wavelet representation. This results in an algorithm with fast performance as it has vastly decreased length in search space. The **scan method** is selectable between a fixed-stride walk and a smart walk through the time series. **Postprocessing** may be selected from any of the previously described postprocessing routines. We utilize the PyWavelets package in this work [9].

3.5.2 DTW

Dynamic Time Warping is a popular time series matching procedure in which the query may be mapped surjectively to the search subset, effectively pairing each query sample to one or more search samples. Taking the optimal mapping (the mapping which results in the lowest euclidean distance) under some restrictions (endpoint continuity, maximum time scaling limits) performs a nonlinear scaling of time within the search sequence. This agrees well with intuitive ideas of similarity, in which a peak of the same size and shape, but a few samples further in time than expected, is considered by a human observer a very similar sequence. In comparison, a direct Euclidean distance sample-by-sample produces a significant penalty on small temporal deviations. The dynamic time warping routine used is a mature implementation by the authors of [14], and described therein. We omit further discussion of the technique and point the reader to our milestone #1 meeting presentation for a graphical introduction.

3.5.3 APCA

APCA [4] is an approximation of a time series that partitions it into n (a hyperparameter) segments of variable length. Each of these segments is represented by the mean of the data points it contains. The lengths of the individual segments are chosen in such a way as to achieve the most accurate approximation of the time series under the limitation of n segments. This is achieved by a discrete Haar wavelet transformation in which the largest coefficients are retained. This results in an approximation that preserves the essential structural properties, but on the other hand abandons non-essential structures such as noise. During a seek operation a region is transformed into an APCA representation by partitioning it into n segments, same as the query, with segment starting points in the same relative locations. Euclidean distance is measured between these two low-dimensional representations. Since these are relatively inexpensive operations, an APCA matching algorithm is performant with respect to runtime. Furthermore, APCA is particularly suitable for the approximation of time series with sharp edges. But less for natural or sinusoidal signals.

3.6 Probabilistic

At a first glance it may seem unintuitive that we use probabilistic models to compare time series. In this approach we consider the time series as a realization of the probabilistic model by representing the observed time series by an underlying model. The main struc-

ture of the time series is reflected in this model, where the next point in the time series is expected probabilistically.

As **preprocessing**, we must transform the time series into a probabilistic model. To get such a model the observed time series is fit with an Expectation-Maximization-Algorithm generating hyperparameters of our underlying model. Knowing these parameters one can calculate the likelihood prior of the model is correct for the realized time series. We evaluate this likelihood on the query range, and then by any chosen **scan method** compare the likelihood of each subset of interest. The **measure** is a ratio of the query likelihood and the subset of interest likelihood. One can compare it to the likelihood ratio which is used in statistical tests to accept or reject a hypothesis. This procedure stays the same for all probabilistic models. In the following we will consider two of them. It has to be stressed that it is not the same as the likelihood ratio test and does not have the statistical properties of a likelihood ratio test. Nevertheless, it fits our needs in terms of comparing the similarity.

3.6.0.1 Gaussian Mixture Model A Gaussian Mixture Model (GMM) is essentially the mixture of several Gaussian distributions. One hyperparameter is the number of Gaussian models one wants to include in the GMM. The different unique Gaussian distributions will be weighted. In particular, the number of Gaussian models represents the different states of our observed time series. Each different state of the time series can be characterized by a different Gaussian distribution, i.e. it plateaus which can be represented by different means of the Gaussian distribution or the variation gets larger which can be represented through a higher variance.

3.6.0.2 HiddenMarkovModel The Hidden Markov Model (HMM) is a more sophisticated probabilistic model. HMMs have two components. On the one hand are the observed emissions and on the other hand the hidden states. The connection between both is the emission probability. Every hidden state has a distribution for the emission probability and they form a Markov chain of different transition probabilities. This is applied to a time series in a manner analogous to the GMM. The characteristics of the observed values are modeled through the probabilistic model (in this case, a HMM). The HMM, in comparison to GMM, additionally takes the transition between states into account capturing these transitions into the model. Through the additional information the complexity rises and also the computational cost for this model.

3.6.1 Ensemble

We found each algorithm to have strengths and weaknesses in regards to patterns that are reliably matched. A natural extension was to combine the algorithms in an ensemble approach. This is common in time series forecasting work as it decreases the variance and often leads to better results than the singular algorithms.

For our ensemble approach we utilized the DTW, APCA and GMM implementation, based on their promising and complementary performance on the test datasets. Details of this study, showing the relative performance of each algorithm across datasets, are found in Milestone Presentation #2. In the ensemble approach, each algorithm is independently executed. If any algorithm considers the region as a match it is considered as an ensemble

match (an OR ensembleing process). More sophisticated ensemble procedures are conceivable. Since we want to find a robust algorithm against several challenges we relied on the simplest ensembling rule to avoid overfitting for certain test cases.

A challenging detail in the assembling is determining a match quality ($[0, 1]$) based upon the independent match qualities returned from each algorithm. Ultimately, the qualities are scaled uniformly and summed to generate an overall ensemble quality measure from each independent algorithm.

3.7 Performance Evaluation

Investigations into the appropriate performance metrics to evaluate our system yielded a single most-popular measure: The receiver operating characteristic (ROC), and to summarize into a single value from $[0, 1]$, the area-under this curve (AUROC) [11]. This measure indicates how successful a binary classifier is by plotting the true-positive rate against the false-positive rate. A perfect classifier in this space appears as a step function to 1, a random classifier is represented by the diagonal. We encourage the unfamiliar reader to briefly review [11].

Algorithm 2: ROC performance measure

```
Data: ResultMatches, LabeledMatches
Result: Percent of matches for x% of matches
Sort ResultMatches descending by quality and add midpoint
Add midpoint to ManualLabels
usedMatches = 0
for Each ResultMatch ∈ ResultMatches do
    usedMatches += 1
    Find closest LabeledMatch ∈ LabeledMatches
    Get overlap of ResultMatch and LabeledMatches
    if Overlap is at least 40% then
        if overlappedLabels >= x· Length(LabeledMatches) then
            return int(x · Length(LabeledMatches)/usedMatches
        end
    end
end
```

We have implemented our own version of this routine. We consider a return a true positive if it overlaps a labeled range at least 40%, and a false positive if it does not. A false negative is represented by any labeled pattern that is not found by the algorithm. For the number of true negatives we must determine the total population of possible regions in our time series - that is: how large is the strict-duration set $\mathbf{A}_l$?

A lower bound would be every subset we explicitly evaluated by the scan method, however this is inappropriate because our measure would then be very sensitive to changes in step size and stepover size. More intuitively, taking large steps does not remove regions from $\mathbf{A}_l$, it only preemptively assigns a negative response by the classifier. Therefore these should be included in the total population.

An upper bound for the total population is is infinite - we can check a region infinitesimally close to the last region; but this is also inappropriate. Assuming for simplicity of

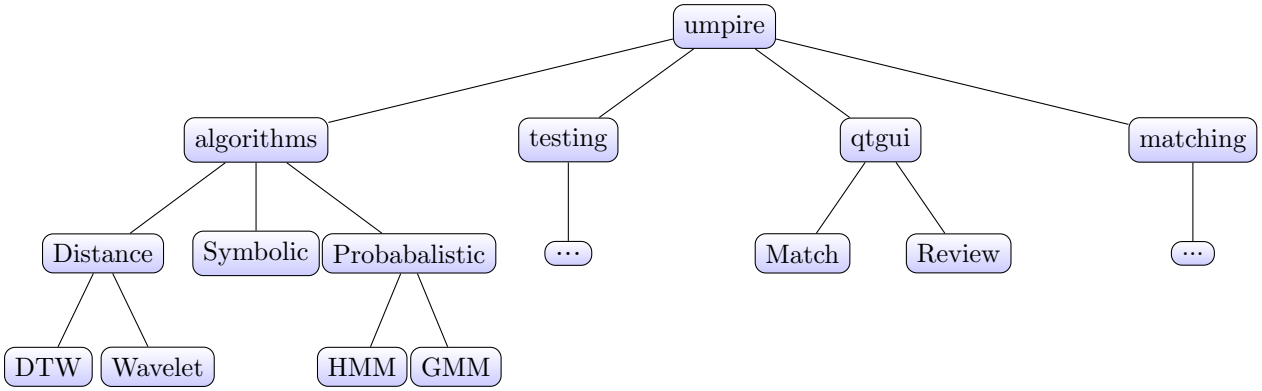

Figure 3: Simplified modular structure of Python module *umpire*

discussion a fixed sampling rate, it is tempting to use as a total population the set of every range of length l starting from a unique sample. This is not fundamentally better than the continuous case; and would clearly result in near-zero false positive rates, as the number of true negatives dwarfs the possible number of matching regions.

To resolve this and allow ourselves to use the ROC tool we again must turn to the user experience (and to the ultimate goal of the project): At what distance would regions be spaced such that a user would identify any closed region as belonging to the same pattern region, and any further spacing as belonging to a different possible pattern region? Human pattern perception is relative; as a result we suggest using a fraction of the query pattern length as the effective spacing to determine a total population. By intuition we suggest this value is somewhere in the range of 1/8 to 1/4 of the query length, and conservatively choose 1/8. Therefore we assume the total population (for the purposes of ROC plotting) as:

$$\frac{8 \cdot \text{Total Timeseries Duration}}{\text{Query Duration}}$$

3.8 Program Structure

An important goal of the team throughout this project was provide a Python module that will be readily adopted by the DLR, and extended after our project is complete. We wished to build a tool providing useful benefit to the engineers, rather than an academic codebase with a high barrier of entry for future contributions. This was accomplished by modularizing the design, following a coding style standard, formalizing our interfaces, and providing useful documentation. In addition, the DLR advisors were invited to our versioning repository at the start of the project, allowing them to follow the project as it develops and keep a log of design decisions.

3.8.1 Modular Design

The suite was designed as a Python 3.6+ module with comprised of a set of optional sublibraries. A simplified overview of this structure can be seen in Fig 3. In this structure, each submodule in the tree is available independently of its siblings. Reuse of the

`algorithms` framework does not require the `testing` submodule. Likewise, removing the `qtqui` module completely removes the QT framework from the requirements from the umpire package. In this way, we have built a complete and functional tool that in addition may be easily deconstructed and adopted by the client.

3.8.2 Coding Style

Though imperfect in execution, and in light of the absence of a defined DLR style, the team defined and largely followed a code style guide in line with current Python best practices [16]. In addition to the PEP8 best practices, we generally use:

- `lowerCamelCase` for most methods, both class initializers and functions
- `CamelCase` for dataclasses and namedtuples.
- leading underscores (`_example`) for functions and variables which should not be interacted with directly by the user, or that may escape their typically intended scope.

3.8.3 Formal Interfaces

It is critical to adopt a formal interface for the matching algorithms in this project, therefore allowing adoption into an existing DLR codebase with minimal effort. Each algorithm is defined by a class with a common interface for initialization, querying, and comparison. Likewise, testcases and match result files share a common markup-language and structure. Further details of the interfaces are beyond the scope of this report, but documented examples may be found in the `umpire.algorithms` and `umpire.testing` modules, respectively.

3.8.4 End-user GUI

In addition to the label tooling described in Section 2.4.1, the team developed a pattern-matching GUI (Fig 4) which may be used immediately by the staff at the DLR to select regions of interest from arbitrary time series, and explore the patterns matching this selected region as returned by various algorithms.

3.8.5 Documentation

The module has been implemented with self-documenting functions in the form of Python docstrings. These follow the format of Google's Style Guide, section 3.8 [7]. The advantages of following this format are numerous: clearly readable, automatic parsing by many IDEs, and perhaps most importantly for us is the ability to automatically generate API documentation using, for example, Sphinx [2].

3.9 Computational Experiments

The final evaluation of the system took place in two stages: first, a parametric study on the UCR data afforded a view into the pareto-front of settings for each algorithm. This

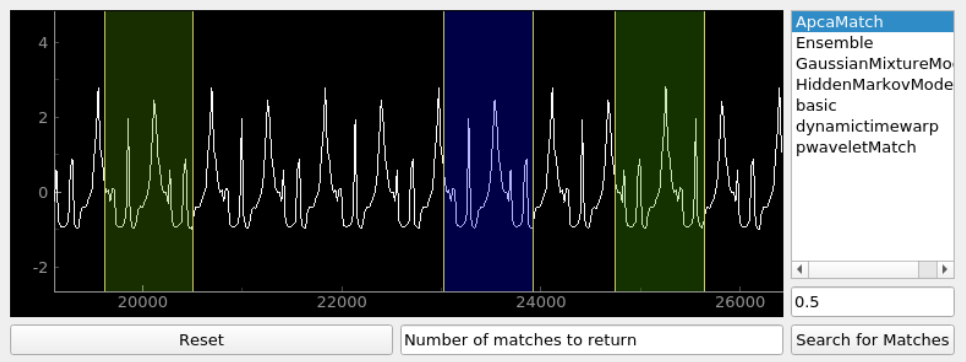

Figure 4: Screenshot of tool developed for interactively finding matched patterns of interest. This tool is able to be used immediately by ground-engineers at the DLR. It allows selection of a region of interest (blue) and returns color-coded matches (red-green mapping low-quality to high-quality matches). The algorithm to use is selected in the right-hand-side dialog box. This algorithm list is populated automatically from the available algorithms discovered in the `umpire` module, allowing for easy development and exploration of new matching routines.

allows us to make suggestions on optimal parameters for each algorithms; affords insight into how to improve the algorithms and approaches; and indicates the relative value of each component of our pipeline.

We also apply the two best performing algorithms to the satellite test data to evaluate performance.

3.9.1 Parameter Study

There are a number of parameters for each algorithm to vary - examples are the bounds for time-scaling limits in DTW; the number of segments in APCA; an appropriate depth for Wavelet decomposition; and the number of hidden states in a HMM. Conservative selections for these parameters were used, based on experimentation throughout the semester and existing values used in literature.

In addition to these parameters, there are a number of parameters which affect all algorithms; these parameters were used for an optimization study on the UCR dataset utilizing a compute cluster. These parameters are:

Symbol	Description	range
τ	Quality measure threshold, below which a potential match is discarded.	0.1 - 0.75
α	Step size fraction. The fraction of the query length (in time) which is used as the stride for a seek	$1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$
β	Stepover fraction. The fraction of the query length to step over in the case of a smart walk. $\beta = 0$ is defined to be a fixed-walk.	off, $1, \frac{1}{2}$
P	Postprocessing method	none, best, combine

This parameter study results in a total of over 6000 pattern matching searches.

Analysis of the performance at each of these iterations is performed using the AUC metric. For brevity and effective visualization, this document will present the results in aggregate: at each combination of parameters, the sum of all AUC scores across test cases is used as the final score. For 14 test cases, this would result in a perfect classifier having a score of 14. A classifier with random (coin-flip) classification on every test case would score a 7.

3.9.2 Performance Study

We do not have the computational resources available to perform a full parametric study on the DLR datasets, which may not be copied to cloud providers (including the LRZ), therefore we have performed a simplified representative study with the most promising algorithms from the UCR study (GMM and APCA) at a low match threshold (0.01) with a fixed-walk at 1/8 query length. This provides the purest result for a AUROC curve and indicates the general performance of these two algorithms on the DLR dataset.

4 Results and Discussion

A sampling of the results from the parameter study are presented in figures 5 - 7. Representative results from the performance study are found in Fig 8.

Fig 5 shows the aggregate sum on AUC scores on each of the 14 UCR testcases. We note that the baseline *Wavelet* approach is a poor classifier. In general for all algorithms a finer stepsize ($\alpha = 8$ results in a step 1/8 the query length) provides better results; likewise, a lower threshold results in more effective classification. These points must be balanced with runtime as stepsize and runtime scale linearly; and increasing the threshold improves runtime but to a lesser extent (any item below the threshold is discarded from memory).

Better performance is observed with no postprocessing (Fig 6) than with the post-processing routines. Here we observe limitations with AUROC as the metric for our performance. A greater number of false positives has a less pronouced effect on the AUROC measure than it does on user experience: a user may tolerate only a certain number of false positives before discarding the tool. As the postprocessing routines were added to improve this user-experiece, it is unsurprising to see them have a negative effect on the AUROC score.

In general, the algorithms perform satisfactory on the classification problems. In most cases the primary problem resulting in a low AUC score is that not all labeled patterns are identified by the algorithms, no matter how low the threshold is set. Two reasons appear for this. The first is that we do not perform a complete scan through the time series in infinitesimally small strides - some pattern regions may not align with the scan stride, resulting in a failure to classify this region. Increasing to a finer and finer scan increases computation time and introduces the additional issues with user experience regarding overlapping regions.

The second problem reflects the nature of the work and was indicated in our introduction: what a *similar* pattern is is determined by human intuition. As a result, any metric may fail to correlate with human judgement; and furthermore labeled datasets may not be precise and free of errors.

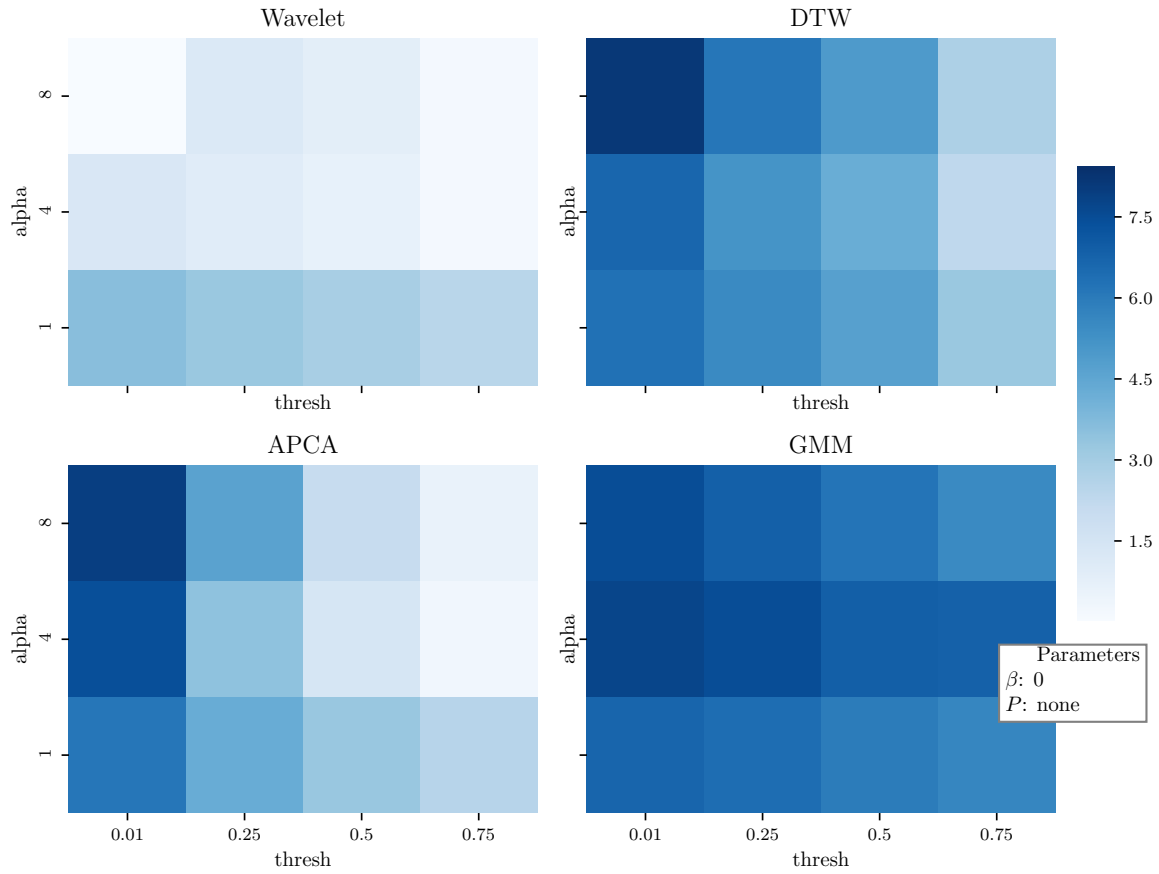

Figure 5: Parameter study results on UCR dataset. Heatmap shows aggregate AUROC score over 14 test cases. Higher scores represent a better classifier, a 14.0 is a perfect classifier on all testcases. In this plot, smart walk is disabled and there is no postprocessing.

We may gain additional insight by reviewing two reasonable, but not best-case, parameterizations in the original ROC space. The first, Fig 7 shows the performance of each algorithm at matching each of the 14 test cases in our UCR dataset. Here we see again the singular AUROC score may not reflect the user experience - the GMM method often scores well on AUROC, but its precision over the first 10% of matches has many more false positives than the equivalent APCA, which has excellent performance with its first matches. In all three plots most test cases terminate around the same TPR - note the green test case. Perhaps in this case the human who labeled the testcase did so with a different eye for similarity than the Euclidean norm, DTW distance, or likelihood ratio do.

The same comparison between APCA and GMM translates to the DLR dataset (Fig 8). APCA, when effective, generally returns reliable matches as the highest scoring regions. GMM, while often reaching 100% of matches ultimately, return many false positives to the user as the highest quality regions.

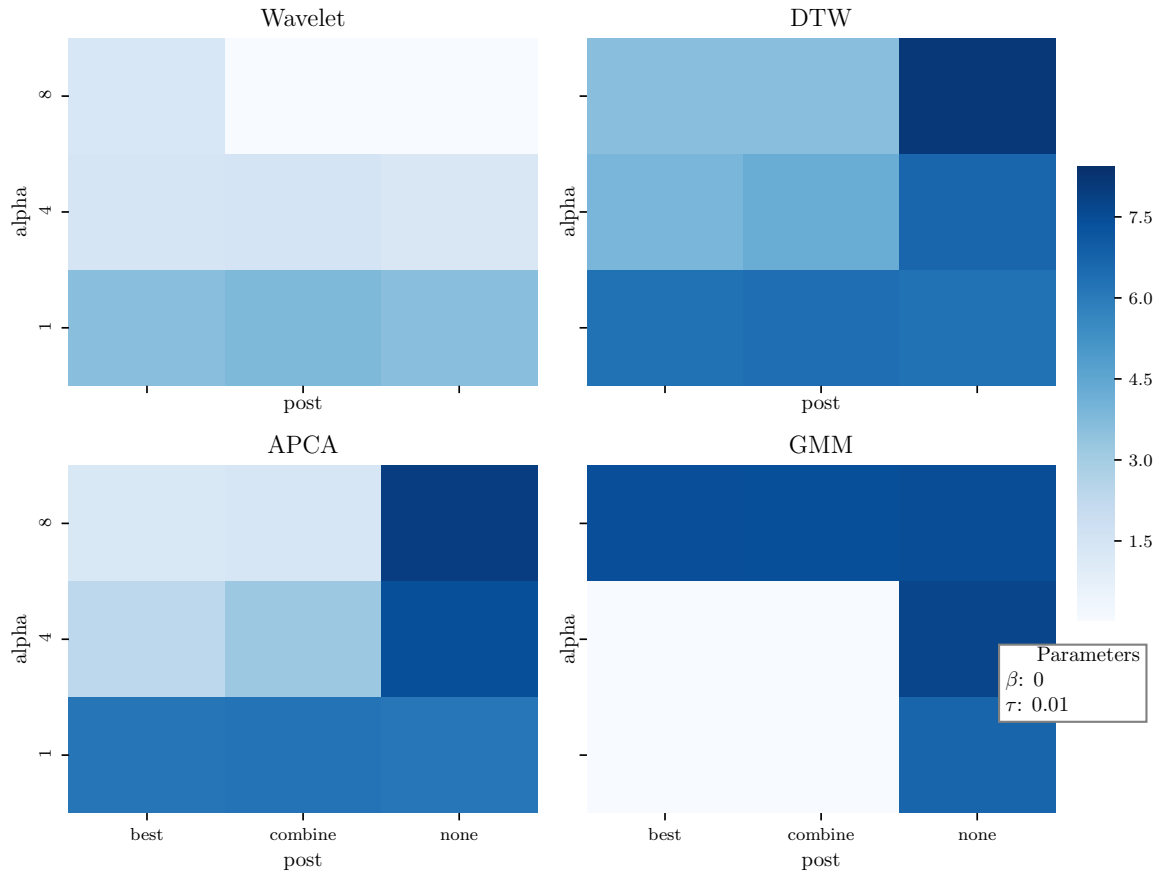

Figure 6: Parameter study results on UCR dataset. Heatmap shows aggregate AUROC score over 14 test cases. Higher scores represent a better classifier. 14.0 is a perfect classifier on all testcases. In this plot, a fixed walk is performed at different step sizes α and different postprocessing methods. The threshold is fixed and smart walk is disabled. The ROC curves from the top-right squares (excepting Wavelet) are shown in Fig 7

5 Conclusions and Future Work

In summary, we have successfully developed a software suite which gives a technical user the ability to easily label new signal datasets for testing; incorporate new matching algorithms; evaluate the performance of algorithms; implement new performance measures; and most importantly gives a nontechnical user the ability to, through a GUI, select a range of time containing a pattern of interest be presented a set of reasonable-quality matching regions in the signal's history. The goals of the project as it was presented (and summarized in the Introduction chapter) have been achieved.

Furthermore, the software has a practical, professional design and will serve as a strong foundation for future work in this area at the DLR.

6 functional pattern-matching algorithms are included in the software - one of which (DTW) is the direct application of an existing library for this purpose [13]. Algorithm performance is challenging to measure; however the initial results are promising. Already

the tool provides a useful set of results - further optimization of algorithm parameters should be undertaken by our successors.

A valuable exploration in the immediate future would be into evaluation of the software from a user perspective. Though AUROC results are higher at low thresholds and step sizes; human preferences for performant software and a clean set of results may encourage other parameterization realizations.

We wish to thank our advisors both at TUM and the DLR and hope that our work here will be utilized and built on for many years.

References

[1] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. "Efficient Similarity Search In Sequence Databases". In: vol. 730. Jan. 1993, pp. 69–84.

[2] Georg Brandl et al. *Sphinx Python document generator*. URL: `https://www.sphinx-doc.org/en/master`.

[3] Anthony J. Bagnall et al. "The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version". In: *CoRR* abs/1602.01711 (2016). arXiv: `1602.01711`. URL: `http://arxiv.org/abs/1602.01711`.

[4] Kaushik Chakrabarti et al. "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases". In: *ACM Transactions on Database Systems (TODS)* (Jan. 2002). URL: `https://www.microsoft.com/en-us/research/publication/locally-adaptive-dimensionality-reduction-for-indexing-large-time-series-databases/`.

[5] Yanping Chen et al. *The UCR Time Series Classification Archive*. `www.cs.ucr.edu/~eamonn/time_series_data/`. July 2015.

[6] Bilal Esmael et al. "Improving time series classification using Hidden Markov Models". In: *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*. 2012, pp. 502–507. URL: `https://pure.unileoben.ac.at/portal/files/1073252/Improving_Time_Series_Classification_Using_Hidden_Markov_Models.pdf`.

[7] Google Inc. *Style guides for google-originated open souce projects*. URL: `http://google.github.io/styleguide/pyguide.html`.

[8] Kin-Pong Chan and Ada Wai-Chee Fu. "Efficient time series matching by wavelets". In: *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*. Mar. 1999, pp. 126–133. DOI: `10.1109/ICDE.1999.754915`.

[9] Gregory Lee et al. "PyWavelets: A Python package for wavelet analysis". In: *Journal of Open Source Software* 4.36 (Apr. 12, 2019), p. 1237. ISSN: 2475-9066. DOI: `10.21105/joss.01237`. URL: `http://dx.doi.org/10.21105/joss.01237`.

[10] Jessica Lin et al. "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms". In: *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. DMKD '03. San Diego, California: ACM, 2003, pp. 2–11. DOI: 10.1145/882082.882086. URL: http://doi.acm.org/10.1145/882082.882086.

[11] Margaret Pepe, Gary Longton, and Holly Janes. "Estimation and Comparison of Receiver Operating Characteristic Curves". eng. In: *The Stata journal* 9.1 (2009). ISSN: 1536-867X.

[12] R.J. Povinelli et al. "Time series classification using Gaussian mixture models of reconstructed phase spaces". en. In: *IEEE Transactions on Knowledge and Data Engineering* 16.6 (June 2004), pp. 779–783. ISSN: 1041-4347. DOI: 10.1109/TKDE.2004.17. URL: http://ieeexplore.ieee.org/document/1294898/ (visited on 07/27/2019).

[13] T. Rakthanmanon et al. "Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping". In: *ACM Transactions on Knowledge Discovery from Data* 7.3 (2013). ISSN: 15564681.

[14] Thanawin Rakthanmanon et al. "Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data". In: Dec. 2011, pp. 547–556. DOI: 10.1109/ICDM.2011.146.

[15] Sara Rebagliati and Emanuela Sasso. "Pattern recognition using hidden Markov models in financial time series". In: *Acta et Commentationes Universitatis Tartuensis de Mathematica* 21.1 (July 2017), p. 25. ISSN: 2228-4699, 1406-2283. DOI: 10.12697/ACUTM.2017.21.02. URL: http://acutm.math.ut.ee/index.php/acutm/article/view/ACUTM.2017.21.02 (visited on 07/27/2019).

[16] G van Rossum. *PEP8 Style Guide for Python Code*. URL: http://www.python.org/dev/peps/pep-0008.

[17] Patrick Schäfer. "Scalable time series similarity search for data analytics". PhD thesis. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2015. DOI: 10.18452/17338.

[18] Byoung-Kee Yi and Christos Faloutsos. "Fast Time Sequence Indexing for Arbitrary Lp Norms". In: *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB'00* (Jan. 2000), pp. 385–394.

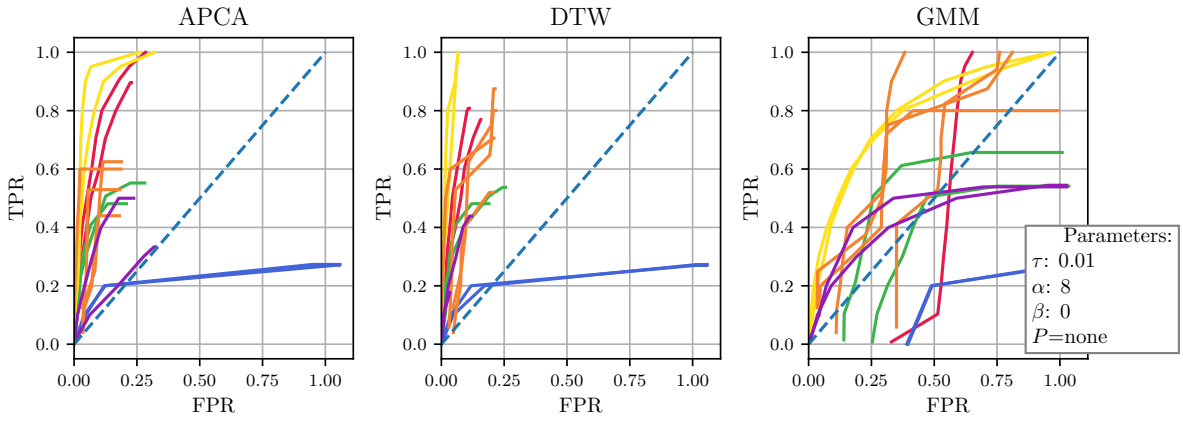

Figure 7: Results across UCR data test cases of APCA, DTW, and GMM methods using a fixed walk with no postprocessing. Graphs corrospond to data in Fig 6. Colors represent one signal, lines sharing a color are different patterns to match within that signal. The dashed line represents a binary random classifier.

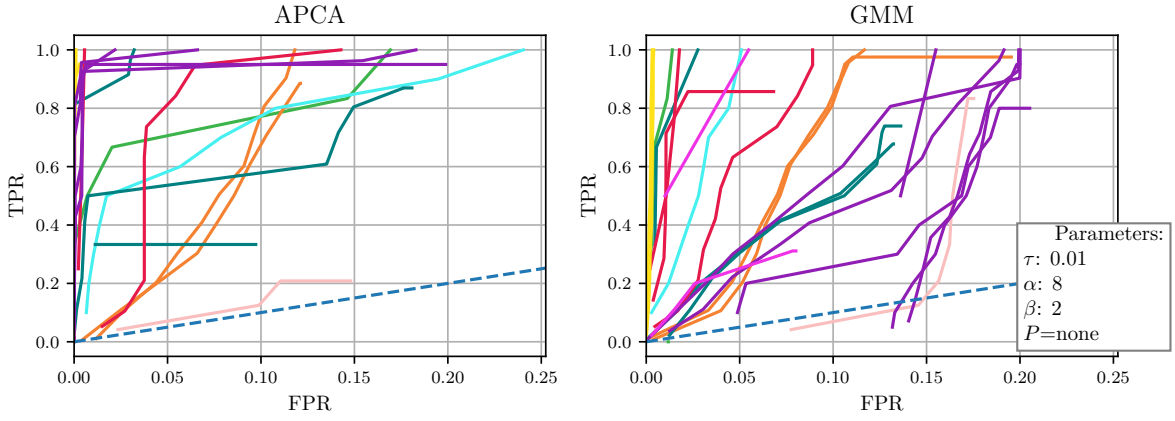

Figure 8: Results across all DLR data test cases of APCA and GMM methods using a smart walk with stepover length equal to one half the total query length. The threshold is 0.01 and the stepsize is 1/8 query length. Each line color represents one signal, lines that share colors are different test case patterns from the same signal. The dashed line represents the expected performance of a random binary classifier.