Masaryk University
Faculty of Informatics

Visual Analysis of Big Time Series Datasets

Master's Thesis

Ondrej Kurák

Brno, Spring 2021

Masaryk University
Faculty of Informatics

Visual Analysis of Big Time Series Datasets

Master's Thesis

Ondrej Kurák

Brno, Spring 2021

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ondrej Kurák

Advisor: doc. RNDr. Barbora Kozlíková, Ph.D.

Acknowledgements

I thank my supervisor, doc. Barbora Kozlíková and Mgr. Juraj Pálenik and prof. Helwig Hauser from the VisGroup of the Univesity of Bergen for their guidance and help. I would also like to thank my colleagues from Gauss Algorithmic a.s. for their support and advice.

Abstract

Because the increasing size of time series datasets makes their manual exploration impossible, there is naturally an increasing demand for fast analytics tools. In this work, we provide the readers analysis of distance measures used for time series, clustering methods for large datasets, and approaches for visual exploration of time series datasets. Further, we propose a novel approach for transforming datasets of time series with multiple components of different lengths into a feature space representation, based on the Feature DTW transformation. In the last part, we demonstrate our approach, in combination with other methods, on the analysis of a large dataset of power consumption curves using clustering and anomaly detection techniques.

Keywords

time series, dynamic time warping, Feature DTW, clustering, anomaly detection, visual analysis

Contents

List of Tables

List of Figures

Introduction

Nowadays, there are only a few aspects of everyday life that are not affected by modern technologies, and the rise of technologies like *Internet of Things* lowers this number even more. These technologies produce large amounts of data with temporal stamp, also recognized as time series. As the amount and speed, at which we are getting new data, are making it impossible to process and analyze them manually, we are in need for fast analytics tools for time series data.

Our work focuses on methods for analyzing large datasets of long time series, demonstrating it on a power consumption dataset. We are using the approach to transform the dataset into a spatial feature space representation. Using this representation, we prepare multiple visualizations for exploring the structures within the dataset and detailed views for examining time series details. To supply the analysis, we apply clustering and anomaly detection methods. This work is divided into four chapters.

Chapter 1 will discuss metrics used for time series, their complexity, advantages and disadvantages, and their application in practical analysis. Afterwards, it explores the representations of time series datasets in artificial feature space. Finally, it examines clustering and anomaly detection methods available for large datatasets.

Chapter 2 will discuss techniques for visual representation and analysis of time series datasets. Firstly, it analyzes the dimensionality reduction techniques for transforming the datasets into low-dimensional embeddings. Secondly, it goes through algorithms for downsampling time series with a specific focus on visual analysis.

Chapter 3 applies the methods from the first two chapters to analyze a large power consumption dataset, while introducing a novel approach for time series transformation.

Chapter 4 is a conclusion of our work, containing the discussion of the benefits and drawbacks of our novel approach about the novel approach for time series transformation, and possible future research directions in this field.

1 Time Series Data Mining For Visual Analysis

Even though most of the data around us have a temporal nature, we often transform them into spatial data by either fixing time or ignoring time completely. It is not because the time aspect is not essential, though it makes all tasks significantly more challenging as it increases the number of data points and adding relations between them. For example, it is straightforward to compare temperatures in two places at a fixed time. However, once we want to compare their temperature profiles throughout the year, we cannot omit the temporal aspect of the data. Because the temporal aspect adds relations between points and increases the dimensionality of the data, the analysis becomes significantly more difficult. This effect is multiplied once we work with a dataset of time series [1]. Mostly there, we see an opportunity where visual exploration could help.

This chapter will discuss the right choices for measuring the distance between time series and examine how to effectively apply them in the subsequent visual analysis. Then it goes through the clustering and anomaly detection methods available for large datasets. However, before we dive into the chapter, we will clarify the basic terminology.

Having a set of objects X, we define the *distance function* as:

$$d : X \times X \to R_0^+ \tag{1.1}$$

The resulting non-negative real value is called a *distance* or *dissimilarity*. We call a distance function a *metric* if and only if it satisfies the following three axioms:

1. $\forall x, y \in X : d(x, y) = 0 \Leftrightarrow x = y$ (identity of indiscernibles)
2. $\forall x, y \in X : d(x, y) = d(y, x)$ (symmetry)
3. $\forall x, y, z \in X : d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality)

A *semi-metric* is a distance function that satisfies the first two axioms, but it does not guarantee the triangle inequality. *A time series* is then defined as a sequence of data points (X_t) indexed by time (t).

1.1 Distance Measures for Time Series

In a typical scenario, a distance metric computes the distance for one specific data type (categorical, spatial, numerical). However, time series are a combination of multiple data types as each point has an order in time, and we have to take it into account when computing the distance.

To the best of our knowledge, there are over 30 used distance measures for time series today. As there is no "universally best" metric [2], we will go through the different types of distance measures used for time series, following the categorization proposed by Esling and Agon [3].

1.1.1 Shape-Based Distance Measures

Shape-based distance measures compare the overall shape of the time series [3]. They are the closest derivation of commonly used distance measures adjusted for the time series data, making them easy to implement and understand. Regardless of their simple nature, they generally yield very competitive results and are considered a gold standard of time series distance measures [4, 5, 6, 2, 7].

Euclidian Distance and Other L_p Norms

The most basic and widely used distance measures for spatial and temporal data are Euclidian distance (ED) and other L_p norms [8, 9].

A distance measure on a set of objects X is a function $d : X \times X \rightarrow [0; \infty)$. Given the two time series X and Y, where $X = (x_1; x_2; ...; x_n)$ and $Y = (y_1; y_2; ...; y_m)$, we define L_p norms as listed in Table 1.1.

Norm	Definition
L_1 - Manhattan	$\sum_{i=1}^{n} \lvert x_i - y_i \rvert$
L_2 - Euclidian	$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
L_p - Minkowski	$\sqrt[p]{\sum_{i=1}^{n} (x_i - y_i)^{\frac{1}{p}}}$
L_∞ - Infinite	$\max_{i=1..n} \lvert x_i - y_i \rvert$

Table 1.1: L_p norms for time series.

Although the L_p norms perform well, especially on larger datasets [2, 4, 10], they require time series of equal length and tend to be fragile to the noise, shifts, and different speeds in time series [3].

Dynamic Time Warping

An elegant solution to the above-mentioned limitations of the L_p norms are elastic measures, especially the Dynamic Time Warping (DTW) [11]. DTW is a method to compute a distance between a pair of time series. When computing DTW distance between time series X and Y, it finds the best alignment between them as a minimal cost warping path ($W = w_1, ..., w_k, ..., w_K$) in $m \times n$ matrix. Elements of the matrix represent the cost to align two corresponding points ($(x_i - y_j)^2$). The warping path starts in the bottom-left corner and ends in the upper-right one (Fig. 1.1a). It is formally defined as:

$$DTW(X;Y) = \underset{W=w_1,...,w_k,...,w_K}{\arg\min} \sqrt{\sum_{k=1;w_k=(i;j)}^{K} (x_i - y_j)^2} \tag{1.2}$$

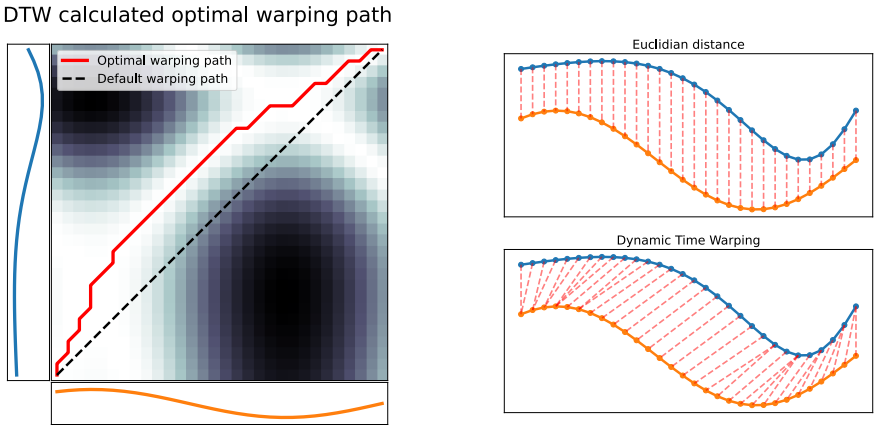
When comparing DTW with the Euclidian distance, DTW finds the best possible alignment between time series (Fig. 1.1b). That allows us to reasonably compare time series that could have different lengths, speeds, and shift [11, 12].

Dynamic time warping is considered one of the best distance measures for time series and regularly outperforms other more sophisticated approaches [2, 4, 6, 5], but its use comes with two main concerns:

1. In practice, DTW is solved by dynamic programming with $O(n^2)$ time complexity.

2. DTW is a semi-metric – it does not guarantee triangle inequality, limiting the number of optimization techniques and structures we can use.

The time complexity of this algorithm becomes an issue when operating with long time series and their large amounts. As a solution to the DTW algorithm's time complexity, it is possible to either use lower and upper bounding constraints on the maximal allowed warping path [13] or use a temporal constraint on the warping window



(a) (b)

Figure 1.1: (a) displays the optimal warping path between two time series. The darkness of color encodes the distance between the corresponding data points (darker is further away). (b) shows the alignment of data points between two time series using the Euclidian distance and DTW.

size [12]. Among the most used are Sakoe-Chiba [12], Itakura [14], and upper bound to discard complex paths [15]. Another option is to use an approximate algorithm proposed by Salvador and Chan [16] called FastDTW, where the warping path is recursively projected and optimized from a lower resolution. This algorithm has linear time complexity. Using the bounding or approximate algorithm comes with faster computational speed and could help with better accuracy due to generalization [7].

DTW is a semi-metric that makes it rather problematic to use with many machine learning algorithms that require a metric space. As there does not exist any complete solution to this issue, there are only partial answers, like Feature DTW from Kate [7]. As we want to use a wide variety of algorithms for visual exploration, we will discuss this in more detail in the following section.

SpADe, DISSIM, and OSB

In 2007, three new distance measures for time series were proposed. The first one is the Spatial Assembling Distance (SpADe), a distance measure used for streaming data that recognizes matching patterns concerning shifting and scaling in temporal and amplitude axes [17].

For measuring similarity between trajectories with different sampling rates, Frentzos et al. [18] proposed the DISSIM as an approximation of the integral of Euclidian distance.

Because data tend to be noisy and include many outliers, Latecki et al. [19] proposed Optimal Subsequence Bijection (OSB) for similarity search. This method automatically finds the most suitable subsequence for a sensible comparison. Its disadvantage is a higher computational cost.

1.1.2 Other Distance Measures

Except for the shape-based metrics used for time series data, other methods could be used for the distance calculation. These types of measures focus on specific properties shared among time series.

Edit-based distances were originally designed for string comparison. The idea is to find the minimal number of basic string operations (insertion, deletion, and substitution) by which we can transform one string to another. The most popular are techniques measuring the similarity by comparing the longest common subsequences (LCSS) [20, 21, 22]. They are robust to outliers and noise in the data. Similar options are the Edit Distance on Real Sequence (EDR) [23] and Edit Distance with a Real Penalty (ERP) [24]. EDR finds the minimal number of edit operations to convert time series and penalizes unmatched regions' gaps by their lengths. On the other hand, ERP is a combination of DTW and EDR approaches. It uses the Manhattan distance as a penalization for local shifting. Edit-based distances tend to perform well, but generally are outperformed by DTW-like measures [5, 6].

Feature-based distance measures use features extracted from the original time series, such as correlation [25] or coefficients from discrete Fourier transformation and discrete wavelet transformation [26]. It is possible to use them in specific applications but they overly show worse results than other techniques [4].

Model-based measures are an option for very long sequences, but they require prior knowledge of the time series generating process [3]. After choosing and fitting the parametric temporal model on time series, the distance measure is a likelihood that time series came from the same model. Most standard models are using Hidden Markov models or ARMA models [27].

Compression-based strategies have been succesfully applied in bioinformatics and medical data applications [28, 29, 30]. This approach uses the fact that concatenation and compression of similar time series should produce a higher compression ratio than for very diverse ones.

Many of these techniques can outperform more common shape-based measures, particularly in specific domains, but lack wide usage and universality of shape-based approaches. Even though we would primarily use the shape-based measure in this thesis, the proposed techniques for time series analysis can use any underlying distance measure.

1.2 Artificial Feature Space Representation

From the previous section, we can presume that shape-based distance measures, especially Dynamic Time Warping and its derivatives, are robust and accurate. The original DTW algorithm's time complexity can be reduced significantly by using bounding or approximation. However, we still need to consider its semi-metric nature because the triangle inequality is required for many machine learning algorithms [31]. It significantly reduces the number of available algorithms that we could use, primarily as datasets' size increases. Partial answer to this problem is building a feature space representation, which we could consider to be in a metric space.

1.2.1 Feature DTW Transformation

Kate [7] introduced an idea to build a feature space representation by using DTW. He transforms the original dataset of time series into a new feature space on which it is possible to apply any technique that is used in a metric space.

The transformation works as follows. Having dataset X consisting of time series $(X_1; X_2; ...; X_n)$, we define the Feature DTW transformation as function

$$f_{fDTW} : X \times X \to M \tag{1.3}$$

where $M = (m_{ij}) \in \mathbb{R}_{\geq 0}^{n \times n}$ is a distance matrix with $m_{ij} = DTW(X_i; X_j)$.

In his paper, Kate sees an improved accuracy in classification tasks to widely used techniques using DTW directly. Further, he shows that using DTW with bonding and approximation increases the accuracy probably due to better generalization, and points out that any distance measure could be used instead of DTW. Another notable advantage of using the Feature DTW transformation is that it is possible to combine DTW obtained features with other measures and features. In his work, Kate [7] combined DTW features with Euclidian ones, which led to an additional approvement in terms of accuracy. This technique's main drawback is space complexity as the distance matrix M size is n^2 regarding the number of time series.

1.2.2 Prototyped Feature DTW Transformation

The result of Feature DTW transformation on two similar time series should produce very similar and correlated feature vectors. A statistical point suggests that we are adding very little new information while increasing the number of dimensions in the new feature space. An increase in dimensionality leads to a rise in computational and space requirements, while having multiple significantly correlated features is a problem for many machine learning algorithms. Iwana et al. [32] showed that using only a fraction of the original distance matrix M could obtain comparable or better results than using the whole matrix.

Having the dataset $X = (X_1; X_2; ...; X_n)$ and a set of time series $P \subseteq X$, $P = (P_1; P_2; ...; P_m)$ called *prototypes*, we define the transformation as function

$$f_{Iwana-fDTW} : X \times X \to K \tag{1.4}$$

where $K = (k_{ij}) \in \mathbb{R}_{\geq 0}^{n \times m}$ is a distance matrix with $k_{ij} = DTW(X_i; P_j)$.

As this method strongly depends on prototypes, their selection is crucial for the method's success. They proposed a supervised technique using the *AdaBoost* algorithm and weak learners on one feature

at a time. Using this technique, they were able to use a smaller number of components while increasing the classification accuracy. Because this method is available only for supervised problems, they briefly discussed multiple statistical strategies to chose possible prototypes.

All of the approaches described in their work require prior computation of the whole distance matrix, which is problematic for larger datasets. In this thesis, we will propose methods that do not require a full distance matrix in advance but rather compute it interactively.

1.3 Clustering

Another part of our work is to determine the clusters in a dataset. Because we are using the Feature DTW transformation, we are not limited to clustering specific only for time series, but we can use a wide variety of clustering algorithms used for spatial data. Because we will be working with large datasets in our analysis, we will list only algorithms that are scalable to a large number of samples and features.

We will separate the clustering algorithms into three categories:

1. Distance-based methods
2. Methods using the nearest-neighbor graph
3. Density-based methods

1.3.1 Distance-based Methods

Distance-based methods are directly using the distance between the data points to determine their affiliations to the clusters. The two main representatives of this category, that scale well with dataset's size, are *K-means* and *Agglomerative Hierarchical Clustering* methods.

K-means

The *K-means* method [33] is a simple clustering algorithm that separates the data into n clusters, minimizing the within-cluster sum of squares. It produces convex and isotropic clusters, and it is easily scalable to large datasets. The disadvantages of this method are that we

have to define the number of clusters in advance, and in some cases, the clusters in data are neither convex nor isotropic. Nevertheless, it is a solid starting point for any cluster analysis.

Agglomerative Hierarchical Clustering

The *Agglomerative Hierarchical Clustering* approach [34, 33] uses a bottom-up strategy to join the two most similar data points or clusters into a single bigger one. The primary representation is a tree-like structure, where leaves are single data points, nodes represent clusters, and the root is a single unified cluster. The algorithm selects two closest nodes based on the merge strategy in each iteration and joins them into a single one, until the given number of clusters is met.

There are four main merge strategies called *linkages*:

1. *Single linkage* – the minimal distance among all tuples of data points from two clusters.

2. *Complete linkage* – the maximal distance among all tuples of data points from two clusters.

3. *Average linkage* – the average distance between all pairs of data points from two clusters.

4. *Ward linkage* – minimization of the within-cluster sum of squares, similar to the K-means.

This clustering method is scalable with the increasing number of samples and, based on the linkage type, provides different cluster types. The ward linkage produces the most even-sized clusters among the merging strategies, but is usable only with Euclidian metrics, while also having much higher time complexity. If we want to use non-Euclidian metrics, the average linkage is a feasible alternative. Both single linkage and complete linkage are very efficient and scale well on large datasets. The drawback is that they are fragile to noise in the data, as they are using only the distance between two points from clusters.

1.3.2 Methods Using the Nearest Neighbor Graph

Another family of clustering algorithms is using the pre-constructed affiliation graph from data points. Usually, it uses the nearest neighbor graph. The two main representatives are the Affinity Propagation and Spectral clustering.

Affinity Propagation

The *Affinity Propagation* [35] sends messages between data points in the affiliation graph to determine cluster centers and the number of clusters. The disadvantage is that it is usable only for smaller datasets because of its quadratic time complexity, making it unusable for our datasets of interest.

Spectral Clustering

The *Spectral Clustering* [36] starts with the nearest neighbor graph represented as a similarity matrix, where each row represents a data point and its graph distance to the other points. After selecting the number of clusters n, it computes the optimal graph cuts to create a connected component for every cluster. The drawback of spectral clustering is that it can be efficiently computed only for a small number of clusters.

1.3.3 Density-based Methods

The last group of clustering algorithms, density-based methods, uses the difference between dense and sparse regions to determine the optimal clustering. The main challenge of these methods is to determine the correct density estimate of the dataset, which is problematic for high-dimensional datasets. As the number of dimensions increases, the volume of the space rises exponentially, thus making the dataset quickly sparse. This phenomenon is also known as the *Curse of Dimensionality* [37].

Due to the expected size of datasets of our interest, and usage of the Feature DTW transformation, we will be interest and the usage with high-dimensional data. Therefore, we are not considering the most common density-based techniques such as *Gaussian Mixture*

Models [38] and *Mean Shift* [39], as they make assumptions about the underlying data distribution, or computationally expensive.

DBSCAN

The first density-based clustering algorithm in our work is the *Density-Based Spatial Clustering of Applications With Noise* (*DBSCAN*) [40]. This algorithm separates the dataset into dense clusters divided by sparse regions and outlying data points, considered as noise. Firstly it separates data points into three groups: core points, non-core points close to the core points, and noise.

We say that a point is a core point if and only if in its surrounding area with radius ϵ there are at least X other data points (the minimal number of data points). The non-core points close to the core points have a core point within the radius ϵ, yet they do not meet the minimal data points requirement within ϵ. Noise points do not have any core points in their surrounding area. Hence, the DBSCAN algorithm automatically detects the number of clusters and only requires the radius ϵ and the minimal number of samples X. The disadvantage of DBSCAN is that it cannot detect clusters with very different density, as the radius is same for each point.

OPTICS

The *Ordering Points To Identify the Clustering Structure* (*OPTICS*) [41] is a partial solution of the DBSCAN's problem with clustering regions with different densities. OPTICS uses the minimal number of neighbors X, and instead of a single radius like in DBSCAN, it uses the maximal radius to consider ϵ. For each point, it computes the core distance, the minimal radius, in which is the point is considered as a core point, and reachability distance to every other point, which is either the core distance of the other point or the distance between them, whichever of them is bigger. If two neighboring points have a reachability distance smaller than the core distance, we consider them to be a part of the same cluster.

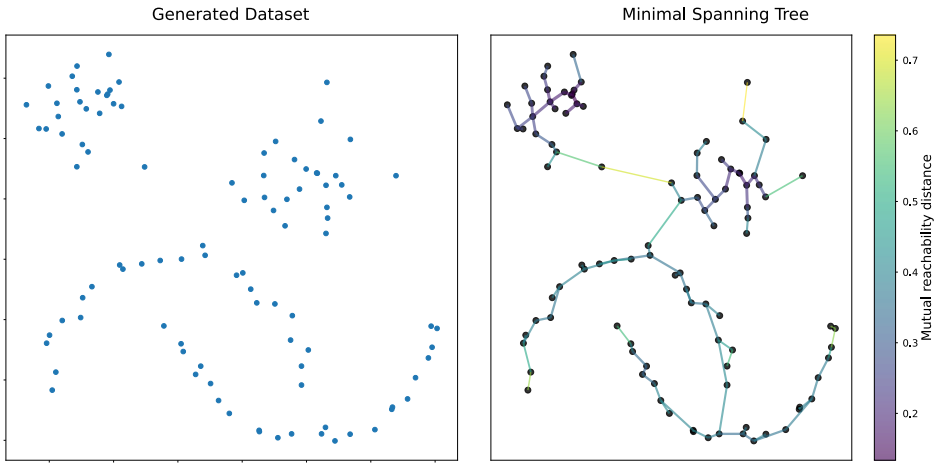
Because OPTICS computes both core distance and reachability, it has a much higher computational cost than DBSCAN. With the usage

of spatial indexing trees, we could avoid a costly computation of the full distance matrix, making it applicable to larger datasets.

HDBSCAN

The *Hierarchical Density-based Spatial Clustering of Applications with Noise* (*HDBSCAN*) [42] is another clustering algorithm originating in the DBSCAN. Like DBSCAN and OPTICS, it starts by computing the core distance of every point, which is a minimum radius in which there are at least X data points (minimal number of data points) and the mutual reachability distance for every pair of data points – the largest value among their core distances and their mutual distance.

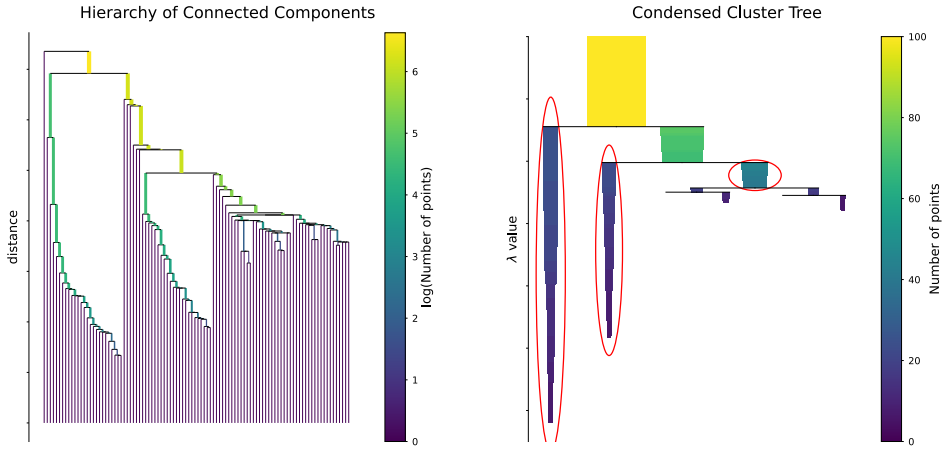
Afterwards, it finds the minimal spanning tree of a weighted graph, where data points are vertices, and edges between them have the weight of their mutual reachability distance (Fig. 1.2b).



(a) (b)

Figure 1.2: The minimal spanning tree from HDBSCAN (a) using the mutual reachability distance on an artificially generated dataset (b).

The next step clusters the tree vertices by single-linkage hierarchical clustering using the edge weights. In Fig. 1.3a, we can see the dendrogram visualizing such clustering, where on the y axis we can see the *mutual reachability distance* at which the nodes merge.



(a) (b)

Figure 1.3: HDBSCAN's single linkage hierarchy of connected components (a) and the most stable clusters in condensed cluster tree (b).

The next step converts the tree into a hierarchy of connected components by sorting the edges and merging them in an ascending order (Fig. 1.3a). To determine the number of clusters in the data, DBSCAN computes the stability of each cluster. As a measure, we are using the λ value, which is the reversed *mutual reachability distance* ($\lambda = \frac{1}{distance}$). For every cluster, HDBSCAN defines the distance when it is created as λ_c and the distance when it splits into multiple clusters as λ_d. Then for every point p in the cluster, it computes the value λ_p, which is the value when the point leaves the cluster either by cluster split or complete separation. The stability of a cluster C is then defined as:

$$\sum_{p \in C} (\lambda_p - \lambda_c) \tag{1.5}$$

Finally, HDBSCAN travels through the tree, starting from the leaves going up to the root node. For every node (cluster), if its stability is greater than the sum of its children's stabilities, HDBSCAN marks the cluster as a real cluster and unmarks all its descendants. Once the algorithm reaches the root node, it stops and returns all marked clusters as final data clustering (Fig. 1.3b).

Even though HDBSCAN consists of multiple computational steps, there are highly optimized implementations [43], allowing HDBSCAN to be effectively used for large datasets.

1.4 Anomaly Detection

In the last part of our analysis, we want to detect and study the anomalous data points within a dataset. Here we will focus on two methods, *Isolation Forest* and *Lightweight On-line Detector of Anomalies*, which are widely used and efficient even for large high-dimensional datasets.

1.4.1 Isolation Forest

One of the widely used anomaly detection algorithms for large datasets is the *Isolation Forest* [44]. The central concept of this approach is that anomalous data points can be isolated faster than the rest of the dataset.

To find the anomalies, it constructs the forest of binary isolation trees. A binary isolation tree is a simple decision tree with restricted depth, where each node of the tree selects the random feature and random value in its range as a separator. The anomaly score of a data point is the average depth it reached in trees in the isolation forest.

Because of its simple nature, the Isolation Forest scales very well with the number of samples and number of features, while also achieving competitive results.

1.4.2 Lightweight On-line Detector of Anomalies

The *Lightweight On-line Detector of Anomalies* (*LODA*) is an anomaly detection method designed to process huge high-dimensional real-time data streams. LODA uses a combination of randomly generated sparse projections and histograms to determine the anomaly score of each data point.

In the first step, LODA generates X random projection vectors with only $\sqrt{d}$ non-zero taken from $N(0,1)$ values, where d is input data's dimensionality. Then, it constructs a histogram on every projection vector. The final anomaly score of each point is the negative average logarithm of probability obtained throughout all histograms.

Because LODA is using sparse projections and histograms as very simple density estimators, it has very fast training and execution time. Another advantage is that through the vectors' sparsity, it is possible to discover which feature contributed most to the data points anomaly score. We compute the *one-tailed two-sample t-test* between probabilities from histograms on projections with and without a specific feature. The higher the value of the *t-test*, the stronger is the feature significance.

1.5 Chapter Summary

In this chapter, we discussed distance measures for time series. Among them, shaped-based distances, particularly the Euclidian distance and Dynamic Time Warping (DTW) seem like the most promising ones. The main drawback of DTW is its semi-metric nature. To use the advantages of DTW in a metric space, we can use Feature DTW transformation to create a feature space representation of DTW. This representation then forms the input for other machine learning algorithms and can be combined with other distance measures and feature extraction algorithms.

There are several options for clustering of large datasets:

- *K-means* is suitable when we know the number of clusters and want evenly sized convex clusters. It is a good starting technique as it scales very well with the dataset size.
- *Hierarchical agglomerative clustering* is suitable for medium to large datasets based on the chosen linkage. *Ward linkage* produces the most evenly sized clusters but has the highest complexity and is defined only in the Euclidian space. For non-Euclidian cases, the *average linkage* is a good alternative. While the *single linkage* is very efficient and can be used on large datasets, with the capability to produce non-convex clusters it is fragile to noise in the data.
- *OPTICS* and *HDBSCAN* are suitable for cases when we do not know the number of clusters in our data but want to specify only the minimal number of clusters. As these methods are density-based, we have to be aware of the curse of dimensionality, and consider transforming the dataset into a low-dimensional representation.

In the field of anomaly detection, we recommend the *Isolation Forest* and *LODA* approaches. Both methods are ensembles of simple estimators, giving them high-speed performance regardless of the dataset size. *LODA* also provides us with the capability for scoring the features based on their significance in the anomaly score.

2 Visual Exploration of Time Series Datasets

In data science, it is very rare to get a well-defined and detailed problem description. A much more common task is to analyze the data and to detect interesting patterns, outliers, or other phenomena that are fulfilling a complex set of criteria, which results in multiple processing steps. As we cannot comprehensively represent all necessary information only by numbers, we use data visualization.

This chapter looks at methods for comprehensible and effective visualization of comprehensibly and effectively visualize a time series datasets. These methods enable to study these datasets in detail.

2.1 Dimensionality Reduction

Usual datasets that we encounter in computer science have often tens or hundreds of dimensions, far beyond our imagination's capabilities. It makes it unintuitive for us to examine the dataset while also increases computational requirements for its analysis and visualization. For a visual exploration of the dataset, it is necessary to project our data into a lower-dimensional space. In this section, we will focus on the dimensionality reduction techniques, that are specifically used in data visualization.

2.1.1 Principal Component Analysis

The oldest and probably the most used dimensionality reduction technique in computer science is the *Principal Component Analysis* (PCA) [45]. PCA is a linear transformation that determines the orthogonal axes in which the dataset has the largest variances and then projects it onto these axes. They are represented as orthogonal eigenvectors with eigenvalues that correspond to the variance on the vector. Formally we define the PCA transformation as:

$$PCA(X) = X \times W_L^T = M \tag{2.1}$$

where X $(m \times n)$ is the original dataset with m data points, each having n features, W_L^T $(n \times l)$ is the transposed matrix of l eigenvectors, and M $(m \times l;\ l \leq n)$ is the lower-dimensional representation of X.

Usually we choose eigenvectors with the largest eigenvalues that are corresponding to directions with the largest variances.

Because PCA comes from statistical data analysis, its primary purpose is to capture maximal statistical information in lower-dimensional representation, not data visualization. We usually use eigenvectors with the largest variance to project data into two or three-dimensional embeddings when we want to use it for data visualization. It means that we mainly show global structures without enough detail to see the data's local behavior (Fig. 2.1b). In combination with incredible speed and efficiency of modern PCA implementations, it is a perfect first step for every visual exploration of multi-dimensional datasets.

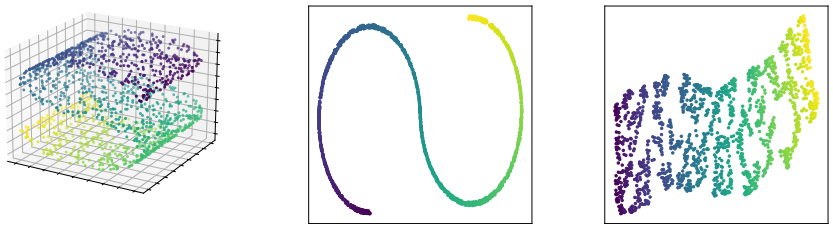
2.1.2 t-SNE

t-SNE or *t-Distributed Stochastic Neighbor Embedding* [46] is a popular manifold representation learning technique that is particularly efficient for visualizing high dimensional datasets [47]. It transforms the multi-dimensional data into a low-dimensional space, emphasizing the preservation of local similarities and distances from the high-dimensional space. It does so by converting affinities of data points into probabilities. Gaussian joint probabilities represent the higher-dimensional space's affinities, and Student's t-distributions represent the embedded space's affinities. Because it very well preserves the local structures in a dataset (Fig. 2.1c), it is one of the most popular techniques for data visualization [48].

However, the usage of t-SNE comes with several disadvantages:

- It is quite computationally expensive. The common practice uses some other fast dimensionality reduction methods, such as PCA, to reduce the dataset into a reasonable number of parameters and use t-SNE afterwards. Another option is to use optimized approximate Barnes-Hut t-SNE [49], which is only available for two or three-dimensional embeddings.

- As it is focusing on preserving mainly the local structure of the data, it does not guarantee to preserve the global structure correctly. For example, the number and distances between clusters are not always reliable.



(a) S-shape dataset (b) PCA embedding (c) t-SNE embedding

Figure 2.1: Comparison of PCA (b) and t-SNE (c) applied to the S-shape dataset (a), commonly used for benchmarking the dimensionality reduction techniques.

- t-SNE is a stochastic algorithm, and each its run every run could return slightly different results.

2.1.3 UMAP and densMAP

Uniform Manifold Approximation and Projection (UMAP) [50] is one of the more recent manifold dimension reduction techniques. Similar to t-SNE, it is excellent for visualization but also as a general non-linear dimensionality reduction transformation. The underlying idea of UMAP comes from the Riemannian geometry and algebraic topology. It makes three assumptions about the data:

1. Dataset has a uniform distribution on the Riemannian manifold.
2. The Riemannian metric is locally approximately constant.
3. The manifold is locally connected.

Using these assumptions, UMAP models the manifold as a fuzzy topological structure, and the resulting low-dimensional representation is the closest possible representation with an equivalent topological structure. In other words, it captures the local similarities of data with respect to their global structure (Fig. 2.2b).

Usage of UMAP over the t-SNE comes with several advantages:

- It scales way better than t-SNE. It is faster, more efficient, and not restricted to two or three-dimensional embeddings so that it can process even high-dimensional sparse data.
- Because UMAP is a general dimensionality reduction technique, it is possible to use it in a preprocessing step.
- Although both methods mainly capture the local similarities in data, UMAP shows better results in preserving the global structure.
- UMAP can use non-metric distance measures.

Despite their visualization qualities, both t-SNE and UMAP neglect information about the local density of the original dataset. As they mainly look at the n closest neighbors but not at their density, it could lead to misleading visualizations where the cluster's size and shape are primarily representing how many points are in it, rather than the underlying distribution.

(a) PCA (b) UMAP (c) densMAP

Figure 2.2: Comparison of PCA (a), UMAP (b), and densMAP (c) on the Fashion MNIST dataset [51].

Narayan et al. [52] introduced density-preserving data visualization derivatives called *den-SNE* and *densMAP*. Since both methods converge by iteratively optimizing their objective functions, they added a new term called *local radius* to represent local densities. In other words, this term represents an average distance to the closest neighbors. We can see the difference in Fig. 2.2, where we applied the UMAP

and densMAP on the Fashion MNIST dataset [51]. The densMAP's embedding is more spread out due to density preservation.

2.2 Time Series Downsampling for Visualization

Plotting time series containing many points comes with several pitfalls. We cannot draw all data points in time series without overplotting due to restricted space, making it almost impossible for us to examine them in detail. The second pitfall is that with the rising number of objects to render, we need more power. Because we do not want to create a misleading plot, finding the closest representation with visible structures while minimizing information loss is essential.

2.2.1 Simple Downsampling and Piecewise Aggregate Approximation

The most straightforward solution to overplotting in spatial data is sub-sampling the dataset, where its time series equivalent is called downsampling. If we try to sub-sample the data in the same way as in the spatial domain, we quickly encounter problems with the temporal aspect.

As we cannot randomly select points, the simplest downsampling algorithm picks every n-th data point. This algorithm is very fast and we can use it on specific types of time series because we are losing $\frac{1}{n}$ of the original information.

Piecewise Aggregate Approximation (PAA) is a simple downsampling algorithm that, instead of removing data points, aggregates them into smaller representations [53]. The basic idea is to split time series into approximately equal-sized buckets and aggregate their data points. We can use any aggregation function like average, mode, or median, based on the application. As the aggregation function combines all data points in the bucket, PAA captures much more information than selecting every n-th data point. Because of the simple nature of PAA, the whole process is fast and scalable.

2.2.2 Largest Triangle Downsampling

Already mentioned algorithms have excellent properties from a computational perspective but not from a visualization standpoint. As they remove or aggregate data points by a fixed value or range, we can lose some visually important information.

Because of this issue, Steinarsson proposed the Largest Triangle downsampling algorithms designed for time series downsampling for visual representation [54]. These algorithms select data points by their effective area, similarly to the *Visvalingam–Whyatt algorithm*. Having data points X_a, X_b, X_c such as $a < b < c$ and indices a, b, c are the positions in time series, the effective area of the X_b is the area of a triangle $X_aX_bX_c$ (Fig. 2.3). The indices a, b, c are chosen differently in every algorithm, and for the final downsampling, we are using the data points with the largest effective area.

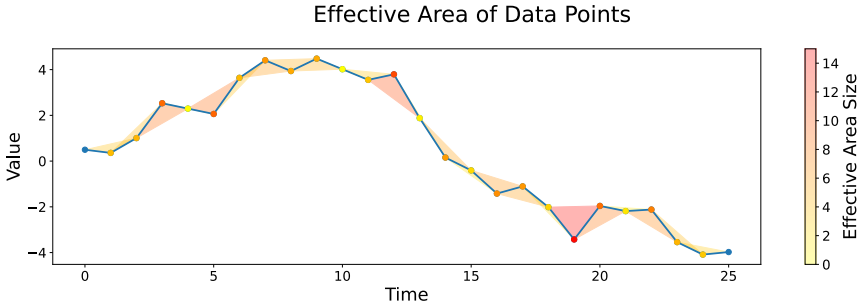


Figure 2.3: Effective area when X_a, X_b, and X_c are directly successive data points (there are no data points between them). The color of the effective area corresponds to the color of a data point, first and last datapoints are different as they are part of the downsampled result.

Largest Triangle One Bucket Algorithm

The simplest algorithm proposed by Steinarsson is *the Largest Triangle One Bucket* (LTOB) algorithm. First, all points get rank by their effective area, which we compute using directly successive data points. Afterwards, it removes data points with zero effective areas and splits points into buckets based on the number of data points we want to have in the downsampled time series. For every bucket, it selects point

with the highest rank (Fig. 2.4). This method's disadvantage is that it only uses the effective area computed from two adjacent points, which can potentially lead to misleading representations.

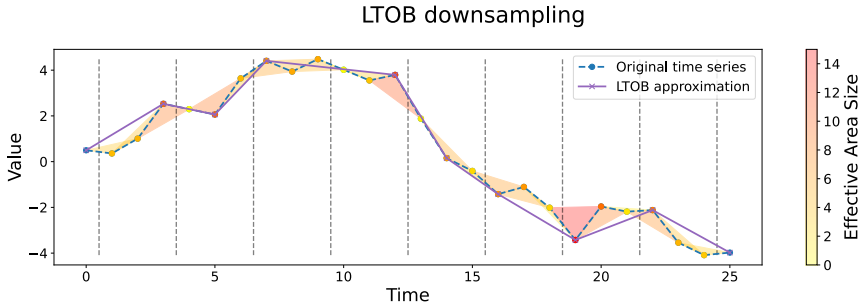


Figure 2.4: For every bucket, the LTOB downsampling algorithm selects data points with the largest effective area within the bucket.

Largest Triangle Three Buckets Algorithm

The Largest Triangle Three Buckets (LTTB) algorithm partially solves the problem of the LTOB and searches a much larger area for point exclusion. First it separates data points into equally-sized buckets, where the first and the last data point get their own buckets, as we do not want to exclude them. Second, it computes the effective area for every point by iterating through the buckets from left to right, taking three directly successive buckets A, B, C at a time. For every point $X_b \in B$, it computes the effective area as an area S of a triangle $X_a X_b X_c$, where $X_a \in A$, $X_c \in C$, such as:

$$S = \max_{X_a \in A,\ X_c \in C} S_{X_a X_b X_c} \tag{2.2}$$

Finally, for every bucket, it selects the point with the largest effective area (Fig. 2.5).

This algorithm is robust, reasonably efficient, and solves the problems of LTOB. The only problem could be the bucket selection when working with data with values non-uniformly spread over time.

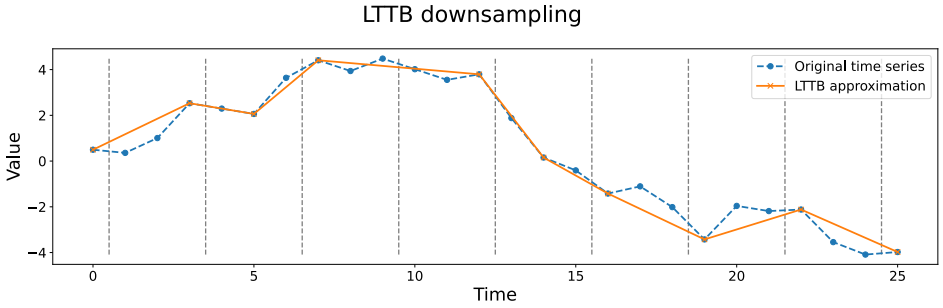


Figure 2.5: LTTB downsampling algorithm.

Largest Triangle Dynamic Algorithm

Because both methods above rely on equally-sized buckets, the last algorithm Steinarsson proposed is the *Largest Triangle Dynamic* (LTD) algorithm. First, it uses the predefined number of equally-sized buckets. Then it computes the interval *calmness* as the mean square error of linear regression on that bucket, including the last point from the previous interval and the first point in the next one. Afterwards, it joins two adjacent intervals with the smallest error and divides the one with the largest error, so the number of buckets remains the same. This iterative process converges to the optimal bucket sizes. The number of iterations is empirically determined, but in the original paper, the author recommends starting with one-tenth of the original time series's size. Afterwards, it uses the LTTB algorithm to select one point from each bucket.

TLD solves the problem of equally-sized buckets but requires a lot of computational time, which makes it hard to apply it to long time series.

Datashader

All of the approaches mentioned so far transform time series into a representation with fewer points, which we then can use for visualization. But as we downsample time series, we lose information based on the number of points we eventually plot. For example, it is not uncommon to have time series consisting of hundred thousand data points, and usually, we are downsampling to less than one thousand data

points for visualization. It is impossible to capture the information precisely and even with the best possible downsampling algorithms, we lose a lot of information, especially the initial data density. If our task requires to study these underlying properties, we must logically plot all our data points.

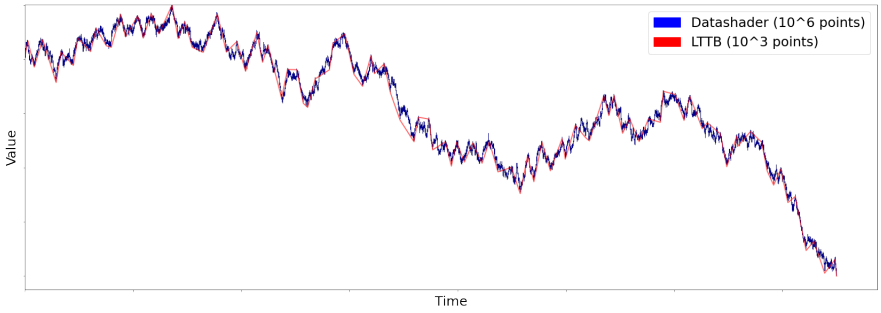


Figure 2.6: Datashader and LTTB demonstrated on one milion points.

The *Datashader* renderer [55] is a complete graphical pipeline from the original data to final graph. It breaks plotting into multiple computations on intermediate representations. That allows multiple fast and efficient aggregations, such as value counts and averaging. Then Datashader renders the results to the final image's pixels as color saturations. The resulting representation is a raster image that accurately represents the aggregated information. Accuracy is limited only by the resolution of the final image and, due to its effectiveness, it can process millions of data points in a reasonable time. As the raster image loses detail as we zoom in, this method gives us an excellent overview of the whole time series (or dataset), but if we want to explore details, we need to repeat the processing pipeline all over again.

2.3 Chapter Summary

In this chapter, we showed several dimensionality reduction techniques used for visualization. Generally, it is advisable to start with PCA to capture the global data structure and then combine it with either t-SNE, UMAP, or densMAP to examine the details. In our case, we will primarily use UMAP and densMAP as they have several advantages over t-SNE.

For time series plotting, we can use the LTTB downsampling algorithm if we want a smaller representation for visualization or use the Datashader renderer to render the original data accurately.

3 Case Study – Analysis of Power Consumption Dataset

In recent years, we observe widespread IoT technologies in everyday life due to the decreasing price of smart sensors and high-quality network availability. Electricity distribution and consumption are no exception, and smart electric meters are now standard components in modern households. It eventually leads to a large amount of sensor data structured as time series, and there is a need for efficient analysis tools to process this information.

As a part of our research in Gauss Algorithmic [1], we got an opportunity to analyze such a dataset. Our assignment was to analyze, find clusters, and detect anomalies in a large dataset of power consumption curves, emphasizing their long-term behavior. We decided to prepare an end-to-end analytic pipeline for this dataset, from the initial data processing to the final outcomes, that can be further used by other data scientists in their studies.

This chapter will go through all steps in our analytic pipeline on the power consumption dataset using techniques mentioned in the previous chapters and will propose a novel approach for time series transformation.

3.1 Dataset Description

First we will describe in detail the dataset that was used within this thesis. Our dataset consists of a large set of power consumption curves for customers located in Slovakia. Because of the strong emphasis on customers' privacy, all data are completely anonymous. We have only randomly generated numbers from our contracting authority representing customer labels and power consumption curves for every consumer. Thus, we lack information about the customer's location or type (apartment, household, company). We have over 20 000 time series from the February 1st 2017 to August 1st 2018. The sampling rate of our time series is one value in 15 minutes.

1. `https://www.gaussalgo.com/`

This initial step in our analysis aims to determine the quality of input data in terms of missing values, extremes, and dataset size. We found a high rate of incorrect or missing data. There are entirely missing days throughout the period and even several months in 2018 (Fig. 3.1). On average, we have approximately 52 000 data points, which corresponds to almost 547 days for each customer. In the end, we removed a portion of power consumption curves because they had only a few to none uncorrupted values, leaving us with over 12 000 time series for further analysis.

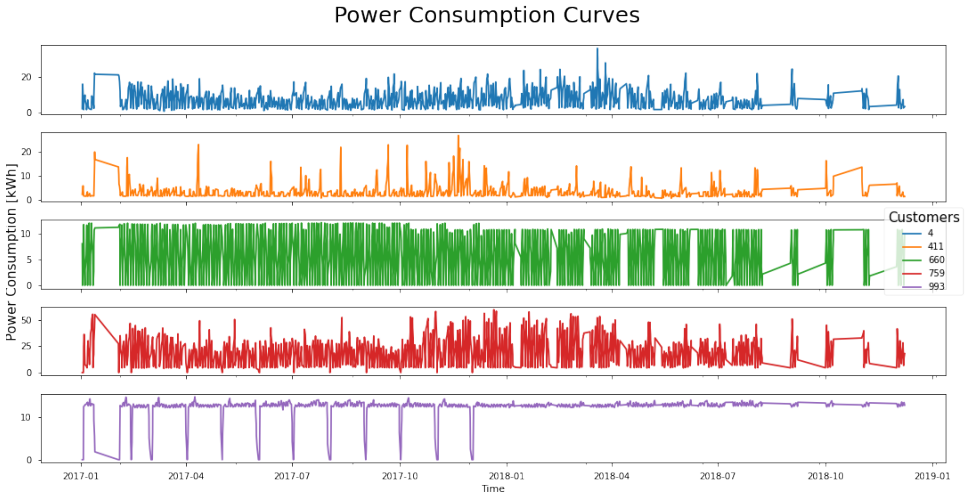


Figure 3.1: Example of power consumption curves with missing values and significantly different behavior.

3.2 Requirements

With our dataset's detailed overlook, we will divide our work into two objectives to accomplish our assignment. The first objective is to process the dataset into a form that could be easily analyzed (*Data Engineering*), and the second goal is to perform the analysis – clustering and anomaly detection with an emphasis on their long-term behavior (*Data Analysis*).

1. **Data Engineering:** A combination of rapidly changing customer power consumption, sensor errors, changes based on external factors, such as time of the year, and lack of information about a customer makes this task remarkably challenging. Another critical detail is the size of our dataset and the length of individual time series. To explore and analyze this data, we have to prepare the following steps: propose preprocessing and feature extraction resistant to missing data, extract long-term behavior, and maintain reasonable time and space complexity for all steps.

2. **Data Analysis:** After the data preparation, we want to analyze our dataset and find outliers. As we are working with a large dataset of time series, we want to prepare meaningful views and visualizations, which will guide a user towards clusters and anomalous data.

3.3 Preprocessing

We are working with a large number of data coming from smart electric meters. It is not unusual to encounter missing or unrealistic values due to sensor or network error. Therefore, it is essential to handle these types of defects in the preprocessing stage so they do not influence the results of the subsequent data analysis.

In our dataset, we distinguish between two unrealistic value cases: values with negative power consumption and sudden dramatic change in power consumption only for a single data point. It is not uncommon to observe a combination of both in one power consumption curve. We assume that this type of error is due to sensor or network failure. To deal with values below zero, we replace them with float *nan* (not a number), representing the missing value. To remove value jumps, we compute our power curve's second derivation and replace values with a disproportionately large absolute value of the second derivation by *nan* values.

We are not replacing the missing values with real numbers in our preprocessing step. Instead, we use the feature extraction technique that is resistant to them. We will address this problem in more detail in the next section.

3.4 Feature Extraction

Consumers' power consumption tends to change rapidly due to sudden extensive usage of electric devices, but it should be more stable in the long-term behavior. Thus we want to extract features that will reflect the long-term behavior and will not be significantly affected by swift changes. These features will provide a better overview of user behavior and make it possible to find users with similar power consumption. To overcome fluctuations of power in short periods, we want to study long-term periodic and non-periodic behavior. We do this by extracting seasonalities (periodic behavior) and trends (non-periodic behavior) in customer power consumption.

Due to the geological location and climate in Slovakia, power consumption is generally changing during the year. The main factors are changes in temperature and sunlight hours through seasons. To capture these changes, we extract weekly seasonality separately for every meteorological season. Another characteristic comes from daily seasonality, where we distinguish between free days and workdays. We also want to utilize trends and yearly seasonalities of our data. As holidays have a substantial impact on power consumption, we want to capture their effect in the model.

For extracting these seasonalities and trends from our data, we are using the Prophet by Facebook [56]. It uses a decomposable time series model with three main components: trend, seasonality, and holidays [57]. It is formally defined as:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{3.1}$$

Where $g(t)$ is the trend function, $s(t)$ is the seasonality function, $h(t)$ corresponds to the effects of holidays, and ϵ_t is noise keeping the normal distribution.

Prophet provides an interface to detect changepoints in trends automatically. However, because our data is relatively short, only eighteen months of data, we do not expect any significant trend changes in our data. Thus we are using a linear trend with zero change points. The Prophet uses the Fourier series representation to model multiple periodic effects (seasonalities) with different lengths. We use yearly seasonality, free day and workday daily seasonality, and weekly seasonality for every meteorological season. To include the effects of

holidays, it uses a matrix of regressors corresponding to every holiday. This effect is also applied to days before and after the holiday. Other essential features are the ability to deal with missing values, computational speed, and scalability.

In this work, we manually define seasonalities based on the expected behavior of different customers based on the consultation with our contracting authority. First, we expect a different behavior on free days (holidays, weekends) and workdays. Then, because of the weather changes during the year, there can be a change in weekly behavior, so we are using a weekly seasonality for every season. Finally, we want to see the trend and the yearly seasonality. We can see the example of this decomposition in Fig. 3.2. Using this feature extraction allows us to find customers with similar properties in one seasonality but significantly different behavior in others – for example, two customers with almost identical seasonalities but significantly higher power consumption could be a sign of energy theft.

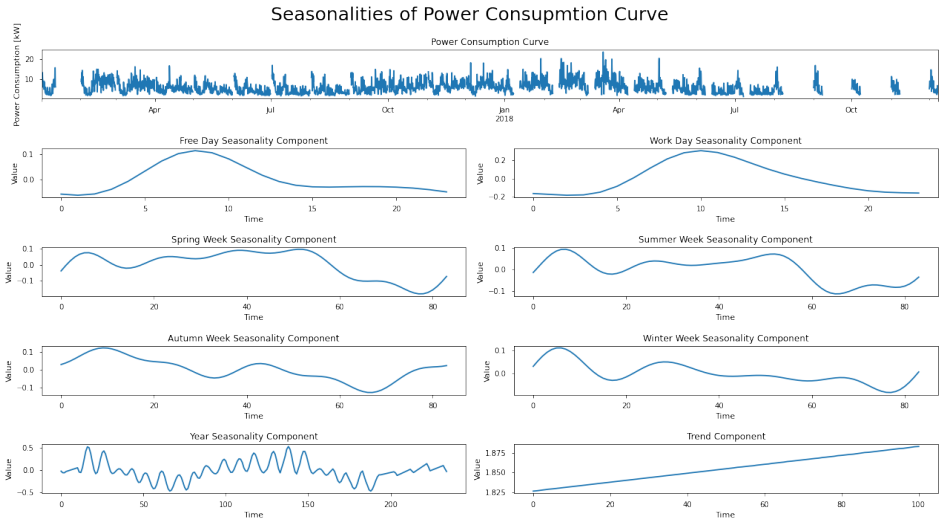


Figure 3.2: Example of seasonalities extracted from a single power consumption curve (top-most curve) from our dataset using the Prophet.

In summary, we are describing our original time series with eight new time series: free day and workday seasonalities, spring, summer, winter, and autumn weekly seasonalities, yearly seasonality, and trend. In the next section, we will provide techniques for combining

information from time series with multiple components with different lengths (*multi-component* or *multi-model* time series) into a single representation.

3.5 Interactive Feature Space Building

Once we use seasonalities and trends, our dataset changes from univariate time series into a multi-component time series consisting of eight time series with different lengths and temporal resolution. Because of the lack of knowledge about the customer, we are working with unsupervised learning on a large dataset of multi-component time series. In our solution, we will first propose a technique for transforming this type of data into a spatial representation and then propose an optimized version of the technique for work with large datasets.

3.5.1 Multi-Component Feature DTW Transformation

We propose a novel technique called *Multi-Component Feature DTW transformations* (*MCFDTW*) to transform the dataset of multi-component time series into a spatial representation. As Kate [7] has shown, it is possible to combine multiple Feature DTW transformations (FDTW). Similarly to his approach, the MCFDTW is a union of FDTW transformations for every component of our multi-component time series.

Having a dataset $S = T_0, T_1, \ldots, T_n$ of multi-component time series $T_n = (a_{n0}, a_{n1}, \ldots, a_{nj})$, where a_{nj} are time series of an arbitrary length and $A_j = (a_{0j}, a_{1j}, \ldots, a_{nj})$ are tuples of time series from one component, we define the MCFDTW transformation as:

$$MCFDTW(S) = (FDTW(A_0)|FDTW(A_1)|\ldots|FDTW(A_j)) \quad (3.2)$$

where | denotes a concatenation of feature matrices from FDTW transformations.

To increase the efficiency of MCFDTW, instead of the original FDTW transformation, we can use the Prototyped Feature DTW (PFDTW) as it is not computing whole distance matrices.

3.5.2 Prototype Selection

The prototypes are crucial for the success of PFDTW transformation, but their correct selection is a challenging task. Iwana et al. [32] proposed two ways for the prototype selection from the full distance matrix for the dataset. The first one is using statistics to remove statistically insignificant feature vectors from the full distance matrix. This technique is usable in supervised and unsupervised learning, but it is impractical on large datasets because it requires the full distance matrix as a starting point. The second approach uses the AdaBoost algorithm to select the best features for classification, making it available only for supervised learning.

Because in our task, we are working with a large dataset and applying an unsupervised learning approach, we are proposing several bottom-up methods for the prototype selection:

1. Random selection
2. Manual selection
3. Selection by density
4. Selection by Predicted Correlation

Random Selection

The first and most straightforward option is to use randomly selected prototypes. This approach is very efficient, and if our sample is large enough, it covers a sufficient portion of the full feature space. Based on our tests on the UCR datasets [58], a random prototype selection seems to be a very efficient approach that still produces comparable results in terms of classification accuracy (Appendix A.1). Because of these advantages, we recommend using this method of prototype selection as a starting point.

Manual Selection

The second approach is selecting specific time series as prototypes by hand. It is usable if the user has deep knowledge about the data

and could pinpoint the significant time series. As the resulting feature vectors are the distances to these specific time series, it is easily interpretable for the user.

Even though using only manually selected prototypes has the advantage of additional explainability, the user could miss some important time series. To minimize this problem, we recommend combing manual selection with a random selection. If randomly selected prototypes improved the final results, we could suspect either that we selected improper prototypes or that our selection is too small.

In our pipeline, we are using the combination of dimensionality reduction methods and visualization for the manual selection of new prototypes (see 3.5.3 Interactive Feature Space Building).

Selection by Density

Another approach comes from an observation that we need more points to describe a dense region than the sparse one, similar to the Kohonen map [59]. This strategy starts with a random sample and then uses data visualization to select new prototypes from the dense regions and remove prototypes that are far away.

Because after the MCFDTW transformation, the transformed feature matrix has the number of dimensions equal to the number of selected prototypes, we can encounter the Curse of Dimensionality [37]. This is problematic for density-based techniques. Because there is no conventional solution to this issue, instead of using a fully automatic solution, we recommend to use a combination of dimensionality reduction and visualization techniques (see 3.5.3 Interactive Feature Space Building). Based on the work with our dataset, this approach yields stable results.

Selection by Predicted Correlation

As two strongly correlated feature vectors do not give much information from a statistical point of view, the last strategy we propose comes from an assumption that we could predict the correlation of potential new prototypes. To do so, we are using the correlations between currently selected prototypes and their feature vectors.

We start with the random prototype selection and transform the dataset into a feature matrix. Then we use Spearman's rank correlation coefficient and compute the correlation matrix between the feature vectors produced from our prototypes. Afterwards, we use the feature matrix and correlation coefficient of other prototypes to train a regressor to predict the correlation on other time series for each prototype. Finally, we select the prototypes with the lowest average predicted correlation.

Unfortunately, this method did not prove significantly better during our tests than the randomized selection of similar size. The prediction of the correlation of new prototypes was reasonably accurate, but it seems that the correlation was not the most reliable predictor of prototype quality. We think that there is an opportunity for future research direction.

3.5.3 Interactive Feature Space Building

Considering the techniques used for prototype selection, we split our feature space building process into an interactive iterative pipeline. The pipeline consists of six steps:

1. Select a subsample of the dataset.
2. Build a Prototyped Feature DTW space.
3. Vizualize the dataset.
4. Choose new or remove prototypes using the visualization.
5. Repeat steps 2-4 until we are satisfied with the result or do not evidence any change.
6. Apply MCFDTW to the whole dataset.

Because our dataset consists of a large number of multi-component time series, we will randomly select a fraction of our dataset in the first step. We are using approximately one-third of our dataset, which allows us to work in real-time. As we expect large groups of similar power curves, the selection of this portion should consist of all major consumer groups, thus minimize the negative effect of not using the whole dataset.

In the second step, we are building the new feature space by using MDFDTW. In the first iteration, we are using the random selection of prototypes. After the first iteration, the feature space building is fully automatic based on the prototypes selected or removed in next step. As we are using the iterative approach, we do not recalculate already computed feature vectors, which increases the efficiency.

In the third step, we are selecting new prototypes using data visualizations. Firstly, we use PCA and UMAP/densMAP to visualize our dataset's global and local structures. As displayed in Fig. 3.3, we can study the local and global structure of the dataset, including prototypes' positions. One point in Fig. 3.3 represents a single customer – one multi-component time series. Using interactive zoom and selection, we can select time series to inspect them directly, study their position and surrounding in the UMAP and PCA transformations, and closely examine their seasonalities and trends (Fig. 3.4).

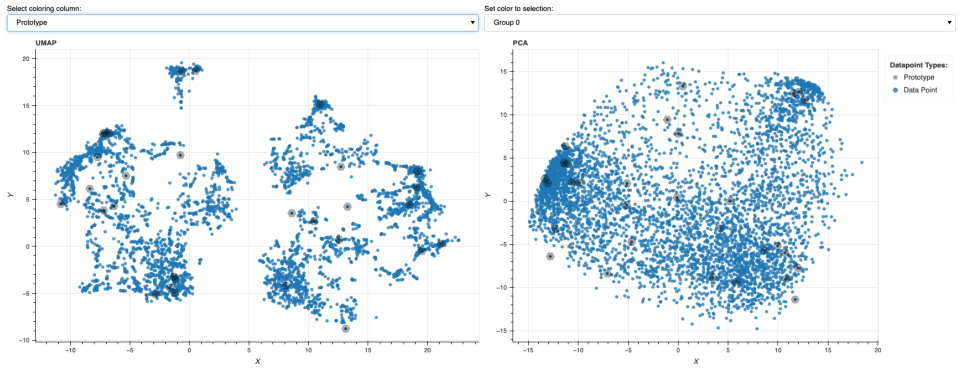


Figure 3.3: Visualization of the subsample of Power Consumption dataset using PCA and UMAP. Grey hexagons highlight prototypes.

To help with the inspection, we are providing multiple coloring options. Firstly, it is possible to select, group, and color time series manually. As this coloring is persistent throughout iterations, we can use it to study the change of positions of our time series in terms of their surroundings. Another option is to use coloring by automatic clustering or anomaly detection score. We will discuss this in the next section.

Once we are satisfied with the selected prototypes, we repeat steps 2-4 until we do not see evidence of any significant change in our visu-

Figure 3.4: Example of inspection of seasonalities and trends of selected customers using the interactive selection.

alization. Because UMAP/densMAP are stochastic, there are always some changes, but they should not be global (mixing of manually selected groups). Afterwards, we apply the MCFDTW transformation to the whole dataset.

3.6 Clustering and Anomaly Detection

As our dataset is transformed into a feature matrix, we can practically use any clustering algorithm for spatial data. In our application, the user can choose from all methods available in *Scikit-learn* or *HDBSCAN*. To overcome the curse of dimensionality from MCFDTW transformation and to reduce the computational time, we are using two clustering pipelines:

1. We are using *PCA* to reduce the number of dimensions into 50-100 dimensions based on the explained variance ratio for *distance-based* clustering methods. Even though these clustering

methods are not as affected by high dimensionality as *density-based* methods, they usually perform better on lower dimensions.

2. For *density-based* methods *OPTICS* and *HDBSCAN*, we use PCA transformation into 50 dimensions followed by UMAP/densMAP embedding into 30 dimensions. As these embeddings preserve the local structure in the data, they help to overcome the sparsity of high-dimensional space.

After the clustering, we can visualize the results in the original visualization of our dataset or its subsample (Fig. 3.5). We can use this view in prototype selection, for example, using prototypes closest to the K-means centers or discovering dense regions using HDBSCAN.

Figure 3.5: Using clustering from the K-means method as coloring in the visualization of a subsample of our dataset.

For detecting the anomalies in our dataset, we are using *Isolation Forest* and *LODA*. It is possible to use them on a whole dataset or a selected subsample. As both these methods are very efficient, we use them directly on the MCFDTW feature vector without any dimensionality reduction. Another option for anomaly detection comes from clustering with OPTICS or HDBSCAN, as both methods select certain points as noise.

Similarly to the clustering, we can plot the results of anomaly detection in our visualizations to either study their location within the dataset or to use them for identifying new potential prototypes (Fig. 3.6).

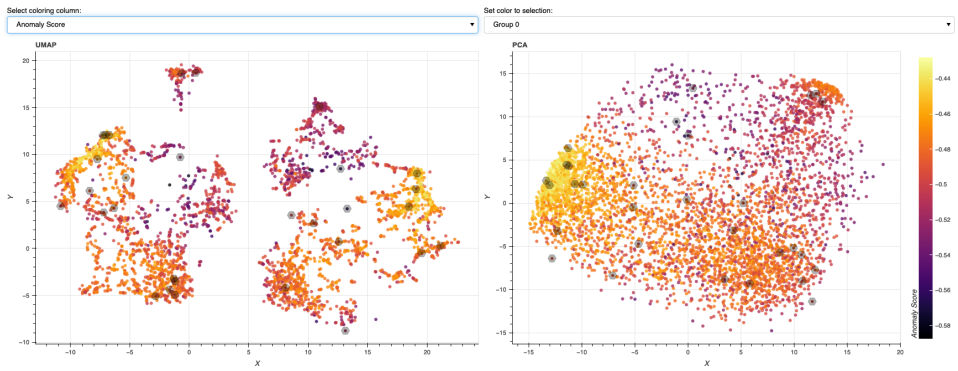


Figure 3.6: Coloring the data points by their anomaly score from the Isolation Forest method. The lower the value, the more anomalous is the data point.

3.7 Implementation

Our solution is mainly usable by data scientists, so we tried to use the tools that are typical and common in this field. We are using the Python programming language, and our solution is distributed as a single open-source package.

We implemented the Feature DTW and related algorithms using the well-known API defined by the *Scikit-learn* project [60, 61], which is the most used tool for machine learning in Python. As *Scikit-learn* does not natively support work with time series, we stick to the time series interface defined by the *Tslearn* library [62]. We are using *dtaidistance* [63], *fastdtw* [64], *dtw-python* [65], and *Tslearn* [62] python packages for computing the DTW distance and its various approximations.

Based on our knowledge of data science work, the most significant part is in the *IPython* [66] and *Jupyter notebooks* [67], and so our implementation is fully integrated into these technologies. For the visualization and rendering, we use the combination of *Matplotlib* package [68] for smaller non-interactive visualizations, *Bokeh* server [69] for interactive visualizations, and *Datashader* [55] for processing and rendering the large time series.

Even though there are some implementations of the LODA anomaly detection method, none of them is complete, and all of them lack the feature to explain the cause of the anomaly. We prepared an open-

source *anlearn* anomaly detection python package [2] with our implementation of LODA, that is, to the best of our knowledge, the only one containing sparse projections and the ability to explain cause of the anomaly.

As we want our work to be fully replicable, we are using the *Nix* package manager [70] in combination with pinned requirements for all Python packages and their dependencies. This allows us to have a fully replicable work environment with the exact version of Python and all the used packages.

We were using only open-source packages in our work, and all of our source codes, including all notebooks from experiments, are available in our Github repository [3].

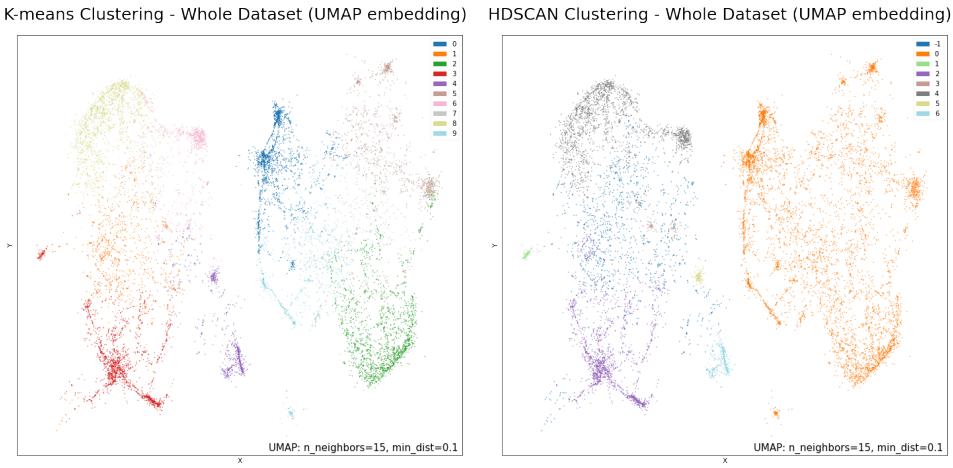
3.8 Results

In our analysis, we prepared a complete processing pipeline for the large dataset of power consumption curves, starting from feature extraction and ending with data visualization, clustering, and anomaly detection.

When manually comparing time series close together in the UMAP visualization, we can see similarities in seasonalities, trends, and statistical properties. Because of this fact, we can assume that the Multi-Component Feature DTW transformation maintained properties necessary for further analysis of multi-component time series.

Upon a visual exploration, we can see two obvious larger well-separated clusters in our data (Fig. 3.7). In K-means clustering, we can see that both visual clusters are divided into convex similar-sized clusters (Fig. 3.7a). In HDBSCAN clustering, we can see six clusters and a relatively large portion of noise data points (Fig. 3.7b). Interestingly, the two large clusters from the visualization are divided very unequally. The first consists only of a single cluster, while the second one of multiple smaller clusters with noise between them.

2. `https://github.com/gaussalgo/anlearn`
3. `https://github.com/H00N24/visual-analysis-of-big-time-series-datasets`



(a) (b)

Figure 3.7: (a) displays the K-means clustering on power consumption dataset. (b) shows the HDBSCAN clustering. Cluster labels are positive numbers, in HDBSCAN clustering -1 corresponds to noise.

In anomaly detection, we used Isolation Forest and LODA methods. The results are slightly different, but we can see overlap in the group of most anomalous points. As the anomalous points have visually different power consumption curves, seasonalities, trends, and statistical properties than the rest of the points, we can consider these users to have an anomalous behavior (Fig. 3.8).

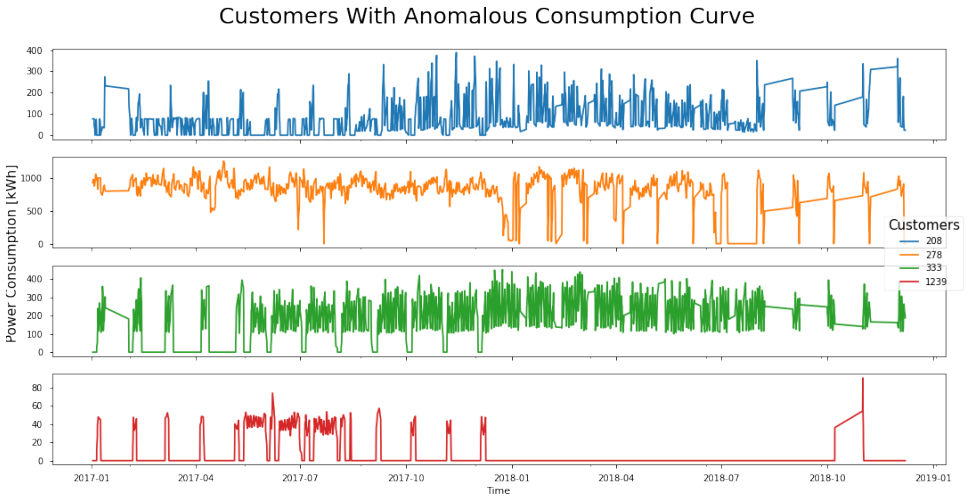


Figure 3.8: Example of anomalous consumers detected by Isolation Forest and LODA methods. Color denotes a single customer.

4 Conclusion and Future Work

The aim of this thesis was to perform an exhaustive literature search and present the overview of the existing techniques used for visual analysis of large datasets of long time series. We applied these methods in the analysis of power consumption curves and proposed a novel approach for transforming datasets of time series into feature space representation.

In the first part, we discussed distance measures used for time series, including their advantages and disadvantages, the Feature DTW transformation, and methods utilized for clustering and anomaly detection on large datasets.

The second part focused on methods used for visual exploration of time series and datasets. We listed several methods for dimensionality reduction that are specifically useful for finding visually meaningful low-dimensional representations of datasets and algorithms for downsampling time series that maintain their visual properties.

The third part, and the main contribution of this thesis, is a proposed novel method Multi-Component Feature DTW (MCFDTW) transformation, and the strategies for prototype selection. MCFDTW efficiently transforms large datasets of multi-component time series into a meaningful smaller feature space representation used in analysis pipelines. To demonstrate the usefulness of our approach, we analyzed the power consumption dataset resulting in multiple views of the dataset revealing the local and global structures within. Then we found clusters in our data with similar seasonalities and trends and also anomalous data points.

From the perspective of future work, we see an opportunity for possible research in converting the manual selection of prototypes to the automatic one for Prototyped Feature DTW and MCFDTW transformations. We tried multiple strategies based on the automatic selection by predicting the correlation of new prototypes or using the data density. None of our strategies provided significantly better results than the same amount of randomly selected prototypes, yet we still believe there is a strategy that will yield superior results.

Bibliography

1. VAN WIJK, J. J.; VAN SELOW, E. R. Cluster and calendar based visualization of time series data. In: *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*. 1999, pp. 4–9. Available from DOI: 10.1109/INFVIS.1999.801851.
2. WANG, Xiaoyue; DING, Hui; TRAJCEVSKI, Goce; SCHEUERMANN, Peter; KEOGH, Eamonn J. Experimental Comparison of Representation Methods and Distance Measures for Time Series Data. *CoRR*. 2010, vol. abs/1012.2789. Available from arXiv: 1012.2789.
3. ESLING, Philippe; AGON, Carlos. Time-Series Data Mining. *ACM Comput. Surv.* 2012, vol. 45, no. 1. ISSN 0360-0300. Available from DOI: 10.1145/2379776.2379788.
4. GÓRECKI, Tomasz; PIASECKI, Paweł. A Comprehensive Comparison of Distance Measures for Time Series Classification. In: STELAND, Ansgar; RAFAJŁOWICZ, Ewaryst; OKHRIN, Ostap (eds.). *Stochastic Models, Statistics and Their Applications*. Cham: Springer International Publishing, 2019, pp. 409–428. ISBN 978-3-030-28665-1.
5. SPIEGEL, Stephan; JAIN, Brijnesh; ALBAYRAK, Sahin. Fast Time Series Classification under Lucky Time Warping Distance. In: 2014. Available from DOI: 10.1145/2554850.2554885.
6. XI, Xiaopeng; KEOGH, Eamonn; SHELTON, Christian; WEI, Li; RATANAMAHATANA, Chotirat Ann. Fast Time Series Classification Using Numerosity Reduction. In: *Proceedings of the 23rd International Conference on Machine Learning*. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 1033–1040. ICML '06. ISBN 1595933832. Available from DOI: 10.1145/1143844.1143974.
7. KATE, Rohit. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*. 2015, vol. 30. Available from DOI: 10.1007/s10618-015-0418-x.

8. YI, B.; FALOUTSOS, C. Fast Time Sequence Indexing for Arbitrary Lp Norms. In: *VLDB*. 2000.

9. KEOGH, Eamonn; KASETTY, Shruti. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*. 2003, vol. 7, no. 4, pp. 349–371. ISSN 1573-756X. Available from DOI: 10.1023/A:1024988512476.

10. GIUSTI, R.; BATISTA, G. E. A. P. A. An Empirical Comparison of Dissimilarity Measures for Time Series Classification. In: *2013 Brazilian Conference on Intelligent Systems*. 2013, pp. 82–88. Available from DOI: 10.1109/BRACIS.2013.22.

11. BERNDT, Donald J.; CLIFFORD, James. Using Dynamic Time Warping to Find Patterns in Time Series. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. Seattle, WA: AAAI Press, 1994, pp. 359–370. AAAIWS'94.

12. SAKOE, H.; CHIBA, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1978, vol. 26, no. 1, pp. 43–49. Available from DOI: 10.1109/TASSP.1978.1163055.

13. KEOGH, Eamonn; RATANAMAHATANA, Chotirat. Exact indexing of dynamic time warping. *Knowledge and Information Systems*. 2005, vol. 7, pp. 358–386. Available from DOI: 10.1007/s10115-004-0154-9.

14. ITAKURA, F. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1975, vol. 23, no. 1, pp. 67–72. Available from DOI: 10.1109/TASSP.1975.1162641.

15. ZHANG, Zheng; TAVENARD, R.; BAILLY, Adeline; TANG, X.; TANG, P.; CORPETTI, T. Dynamic Time Warping under limited warping path length. *Inf. Sci.* 2017, vol. 393, pp. 91–107.

16. SALVADOR, Stan; CHAN, Philip. Toward Accurate Dynamic Time Warping in Linear Time and Space. In: 2004, vol. 11, pp. 70–80.

17. CHEN, Y.; NASCIMENTO, M. A.; OOI, B. C.; TUNG, A. K. H. SpADe: On Shape-based Pattern Detection in Streaming Time Series. In: *2007 IEEE 23rd International Conference on Data Engineering*. 2007, pp. 786–795. Available from DOI: 10.1109/ICDE.2007.367924.

18. FRENTZOS, E.; GRATSIAS, K.; THEODORIDIS, Y. Index-based Most Similar Trajectory Search. In: *2007 IEEE 23rd International Conference on Data Engineering*. 2007, pp. 816–825. Available from DOI: 10.1109/ICDE.2007.367927.

19. LATECKI, L. J.; WANG, Q.; S. Koknar-Tezel; MEGALOOIKONOMOU, V. Optimal Subsequence Bijection. In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 2007, pp. 565–570. Available from DOI: 10.1109/ICDM.2007.47.

20. VLACHOS, Michalis; KOLLIOS, George; GUNOPULOS, Dimitrios. Discovering similar multidimensional trajectories. In: 2002, pp. 673–684. ISBN 0-7695-1531-2. Available from DOI: 10.1109/ICDE.2002.994784.

21. BANERJEE, Arindam; GHOSH, Joydeep. Clickstream Clustering using Weighted Longest Common Subsequences. 2001.

22. MORSE, Michael; PATEL, Jignesh. An efficient and accurate method for evaluating time series similarity. In: 2007, pp. 569–580. Available from DOI: 10.1145/1247480.1247544.

23. CHEN, Lei; ÖZSU, M. Tamer; ORIA, Vincent. Robust and Fast Similarity Search for Moving Object Trajectories. In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. Baltimore, Maryland: Association for Computing Machinery, 2005, pp. 491–502. SIGMOD '05. ISBN 1595930604. Available from DOI: 10.1145/1066157.1066213.

24. CHEN, Lei; NG, Raymond. On the Marriage of Lp-Norms and Edit Distance. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. Toronto, Canada: VLDB Endowment, 2004, pp. 792–803. VLDB '04. ISBN 0120884690.

25. GOLAY, X.; KOLLIAS, S.; STOLL, G.; MEIER, D.; VALAVANIS, A.; BOESIGER, P. A new correlation-based fuzzy logic clustering algorithm for fMRI. *Magn Reson Med*. 1998, vol. 40, no. 2, pp. 249–260.

26. SHATKAY, H.; ZDONIK, S. B. Approximate queries and representations for large data sequences. In: *Proceedings of the Twelfth International Conference on Data Engineering*. 1996, pp. 536–545. Available from DOI: 10.1109/ICDE.1996.492204.

27. XIONG, Yimin; YEUNG, Dit-Yan. Time series clustering with ARMA mixtures. *Pattern Recognition*. 2004, vol. 37, pp. 1675–1689. Available from DOI: 10.1016/j.patcog.2003.12.018.

28. KEOGH, Eamonn; LONARDI, Stefano; RATANAMAHATANA, Chotirat. Towards parameter-free data mining. In: 2004, pp. 206–215. Available from DOI: 10.1145/1014052.1014077.

29. SANTOS, C. C.; BERNARDES, J.; VITANYI, P. M. B.; ANTUNES, L. Clustering Fetal Heart Rate Tracings by Compression. In: *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*. 2006, pp. 685–690. Available from DOI: 10.1109/CBMS.2006.68.

30. ESPOSTI, M.; FARINELLI, Chiara; MENCONI, Giulia. Sequence distance via parsing complexity: Heartbeat signals. *Chaos, Solitons & Fractals*. 2009, vol. 39, pp. 991–999. Available from DOI: 10.1016/j.chaos.2007.03.005.

31. AGHABOZORGI, Sr; SHIRKHORSHIDI, Ali Seyed; WAH, Teh. Time-series clustering - A decade review. *Information Systems*. 2015, vol. 53. Available from DOI: 10.1016/j.is.2015.04.007.

32. IWANA, B. K.; FRINKEN, V.; RIESEN, K.; UCHIDA, S. Efficient temporal pattern recognition by means of dissimilarity space embedding with discriminative prototypes. *Pattern Recognition*. 2017, vol. 64, pp. 268–276. ISSN 0031-3203. Available from DOI: https://doi.org/10.1016/j.patcog.2016.11.013.

33. JIN, Xin; HAN, Jiawei. K-Means Clustering. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Boston, MA: Springer US, 2010, pp. 563–564. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_425.

34. ROKACH, Lior; MAIMON, Oded. Clustering Methods. In: *Data Mining and Knowledge Discovery Handbook*. Ed. by MAIMON, Oded; ROKACH, Lior. Boston, MA: Springer US, 2005, pp. 321–352. ISBN 978-0-387-25465-4. Available from DOI: 10.1007/0-387-25465-X_15.

35. FREY, Brendan J; DUECK, Delbert. Clustering by passing messages between data points. *science*. 2007, vol. 315, no. 5814, pp. 972–976.

36. NG, Andrew; JORDAN, Michael; WEISS, Yair. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*. 2001, vol. 14, pp. 849–856.

37. RICHARD, B. Dynamic programming. *Science*. 1966, vol. 153, no. 3731, pp. 34–37.

38. BAXTER, Rohan A. Mixture Model. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Boston, MA: Springer US, 2010, pp. 680–682. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_547.

39. JIN, Xin; HAN, Jiawei. Mean Shift. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Boston, MA: Springer US, 2010, pp. 652–653. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_527.

40. ESTER, Martin; KRIEGEL, Hans-Peter; SANDER, Jörg; XU, Xiaowei, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Kdd*. 1996, vol. 96, pp. 226–231. No. 34.

41. ANKERST, Mihael; BREUNIG, Markus M; KRIEGEL, Hans-Peter; SANDER, Jörg. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record*. 1999, vol. 28, no. 2, pp. 49–60.

42. CAMPELLO, Ricardo J. G. B.; MOULAVI, Davoud; SANDER, Joerg. Density-Based Clustering Based on Hierarchical Density Estimates. In: PEI, Jian; TSENG, Vincent S.; CAO, Longbing; MOTODA, Hiroshi; XU, Guandong (eds.). *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172. ISBN 978-3-642-37456-2.

43. MCINNES, Leland; HEALY, John; ASTELS, Steve. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*. 2017, vol. 2, no. 11, p. 205.

44. LIU, Fei Tony; TING, Kai Ming; ZHOU, Zhi-Hua. Isolation forest. In: *2008 eighth ieee international conference on data mining*. 2008, pp. 413–422.

45. DUNTEMAN, George H. *Principal components analysis*. Sage, 1989. No. 69.

46. MAATEN, Laurens van der; HINTON, Geoffrey. Visualizing Data using t-SNE. *Journal of Machine Learning Research*. 2008, vol. 9, no. 86, pp. 2579–2605. Available also from: `http://jmlr.org/papers/v9/vandermaaten08a.html`.

47. BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013, vol. 35, no. 8, pp. 1798–1828. Available from DOI: `10.1109/TPAMI.2013.50`.

48. WATTENBERG, Martin; VIÉGAS, Fernanda; JOHNSON, Ian. How to Use t-SNE Effectively. *Distill*. 2016. Available from DOI: `10.23915/distill.00002`.

49. MAATEN, Laurens van der. *Barnes-Hut-SNE*. 2013. Available from arXiv: `1301.3342 [cs.LG]`.

50. MCINNES, L.; HEALY, J.; MELVILLE, J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*. 2018. Available from arXiv: `1802.03426 [stat.ML]`.

51. XIAO, Han; RASUL, Kashif; VOLLGRAF, Roland. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017-08-28. Available from arXiv: `cs.LG/1708.07747 [cs.LG]`.

52. NARAYAN, Ashwin; BERGER, Bonnie; CHO, Hyunghoon. Density-Preserving Data Visualization Unveils Dynamic Patterns of Single-Cell Transcriptomic Variability. *bioRxiv*. 2020. Available from DOI: `10.1101/2020.05.12.077776`.

53. KEOGH, Eamonn; CHAKRABARTI, Kaushik; PAZZANI, Michael; MEHROTRA, Sharad. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems*. 2001, vol. 3, no. 3, pp. 263–286. ISSN 0219-1377. Available from DOI: 10.1007/PL00011669.

54. STEINARSSON, Sveinn. *Downsampling time series for visual representation*. 2013. PhD thesis.

55. CONTINUUM ANALYTICS, Inc.; CONTRIBUTORS. *Datashader: Accurately render even the largest data* [online]. 2015 [visited on 2020-12-17]. Available from: https://datashader.org/index.html.

56. TAYLOR, Sean; LETHAM, Benjamin. Forecasting at Scale. *The American Statistician*. 2017, vol. 72. Available from DOI: 10.1080/00031305.2017.1380080.

57. HARVEY, Andrew C; PETERS, Simon. Estimation procedures for structural time series models. *Journal of Forecasting*. 1990, vol. 9, no. 2, pp. 89–108.

58. DAU, H. A.; KEOGH, E.; KAMGAR, K.; YEH, Chin-Chia M.; ZHU, Y.; GHARGHABI, S.; RATANAMAHATANA, C. A.; YANPING; HU, B.; BEGUM, N.; BAGNALL, A.; MUEEN, A.; BATISTA, G.; HEXAGON-ML. *The UCR Time Series Classification Archive*. 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

59. KOHONEN, Teuvo. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*. 1982, vol. 43, no. 1, pp. 59–69. ISSN 1432-0770. Available from DOI: 10.1007/BF00337288.

60. BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J.; LAYTON, R.; VANDERPLAS, J.; JOLY, A.; HOLT, B.; VAROQUAUX, G. API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

61. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

62. TAVENARD, Romain; FAOUZI, Johann; VANDEWIELE, Gilles; DIVO, Felix; ANDROZ, Guillaume; HOLTZ, Chester; PAYNE, Marie; YURCHAK, Roman; RUSSWURM, Marc; KOLAR, Kushal; WOODS, Eli. Tslearn, A Machine Learning Toolkit for Time Series Data. *Journal of Machine Learning Research*. 2020, vol. 21, no. 118, pp. 1–6. Available also from: `http://jmlr.org/papers/v21/20-091.html`.

63. MEERT, Wannes; HENDRICKX, Kilian; CRAENENDONCK, Toon Van. *wannesm/dtaidistance v2.0.0*. Zenodo, 2020. Version v2.0.0. Available from DOI: `10.5281/zenodo.3981067`.

64. SLAYPNI. *fastdtw*. 2019. Available also from: `https://github.com/slaypni/fastdtw`.

65. GIORGINO, Toni et al. Computing and visualizing dynamic time warping alignments in R: the dtw package. *Journal of statistical Software*. 2009, vol. 31, no. 7, pp. 1–24.

66. PÉREZ, Fernando; GRANGER, Brian E. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*. 2007, vol. 9, no. 3, pp. 21–29. ISSN 1521-9615. Available from DOI: `10.1109/MCSE.2007.53`.

67. KLUYVER, Thomas; RAGAN-KELLEY, Benjamin; PÉREZ, Fernando; GRANGER, Brian; BUSSONNIER, Matthias; FREDERIC, Jonathan; KELLEY, Kyle; HAMRICK, Jessica; GROUT, Jason; CORLAY, Sylvain; IVANOV, Paul; AVILA, Damián; ABDALLA, Safia; WILLING, Carol. Jupyter Notebooks – a publishing format for reproducible computational workflows. In: LOIZIDES, F.; SCHMIDT, B. (eds.). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. 2016, pp. 87–90.

68. HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, vol. 9, no. 3, pp. 90–95. Available from DOI: `10.1109/MCSE.2007.55`.

69. BOKEH DEVELOPMENT TEAM. *Bokeh: Python library for interactive visualization*. 2018. Available also from: `https://bokeh.pydata.org/en/latest/`.

70. DOLSTRA, Eelco; LÖH, Andres. NixOS: A purely functional Linux distribution. In: *Proceedings of the 13th ACM SIGPLAN international conference on Functional programming*. 2008, pp. 367–378.

A Appendix

A.1 Comparison of Distance Measures and Prototype Selection Strategies on UCR datasets

As a part of this thesis, we tested multiple combinations of different distance measures and Feature DTW transformations, including various strategies for prototype selection on UCR datasets [58]. In our implementation, we are using five-split cross-validation and comparing the average accuracy of each algorithm. For complete results, please see our repository [1]. Based on our results (Fig. A.1, Fig. A.2), we make these conclusions:

- The overall best distance measure is a combination of constrained DTW computed on the original time series and their derivations.
- Prototyped Feature DTW outperforms Prototyped Feature DTW.
- Random selection of prototypes can be a useful starting strategy.

Notation

Distance measures:

- *dtw* - Dynamic Time Warping [11]
- *fdtw* - Fast Dynamic Time Warping [16]
- *sakoe_chiba* - DTW with Sakoe-Chiba constraint [12]
- *itakura* - DTW with Itakura constraint [14]
- *dd_distance_measure_α* - a combination of distance measure using the original time series and their first derivatives $((1-\alpha) * dist(x,y) + \alpha * dist(x',y'))$ [7]

1. https://github.com/H00N24/visual-analysis-of-big-time-series-datasets

Prototype selection + Classification methods:

- *Random_X* - Randomly selecting *X*% of training data as prototypes (*Random*_10, *Random*_30, *Random*_50) + Linear SVC
- *LassoSVC* - Selecting prototypes with the highest importance from the Linear Support Vector Machine classifier with l1 penalization + Linear SVC
- *Tree* - Selecting prototypes with the highest importance from the Extra Tree Classifier + Linear SVC
- *SVC* - Using all prototypes + Linear SVC
- 1*NN* - Nearest Neighbor classifier using one closest neighbor

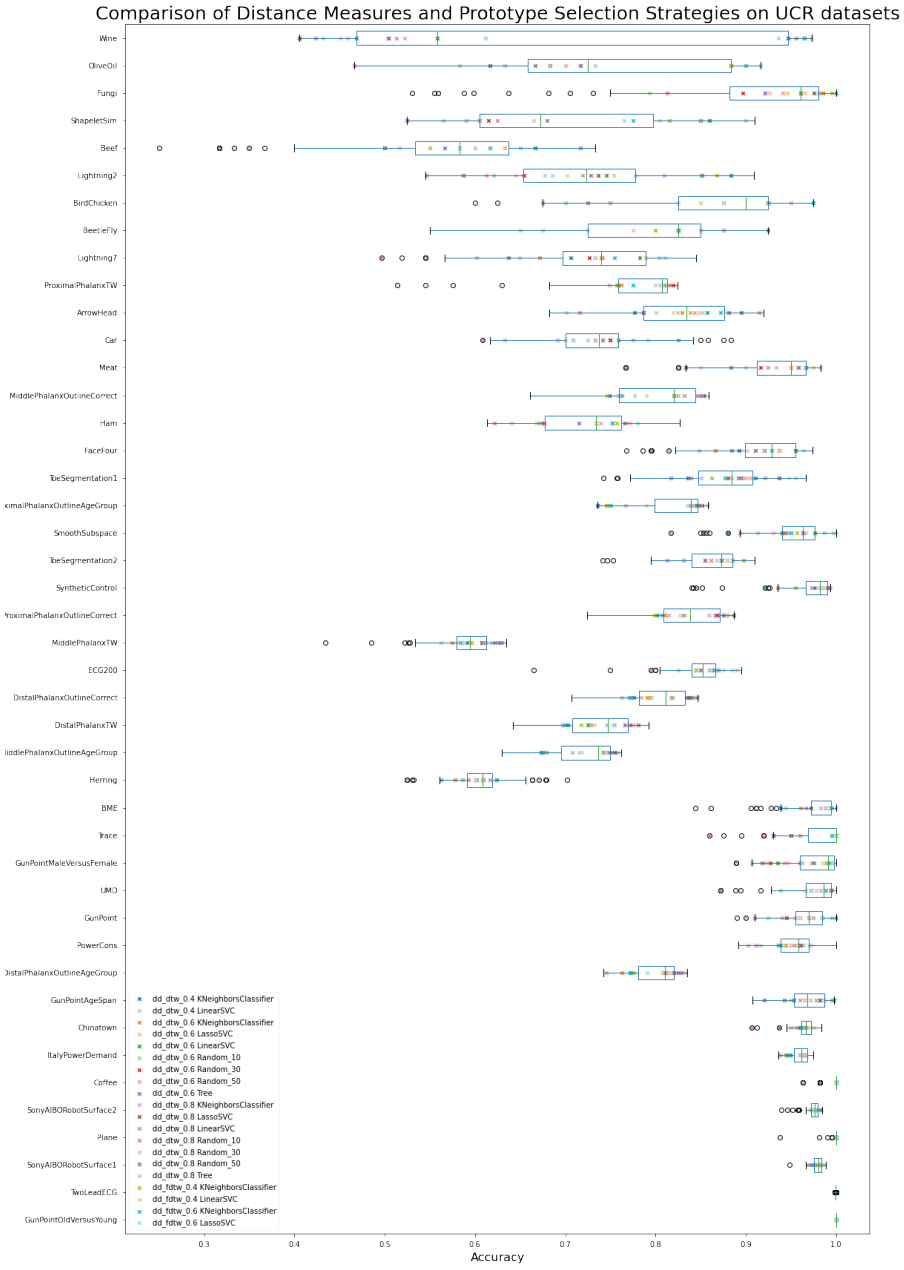


Figure A.1: Average accuracy of distance measures, prototype selection strategies, and classification algorithms on UCR datasets.

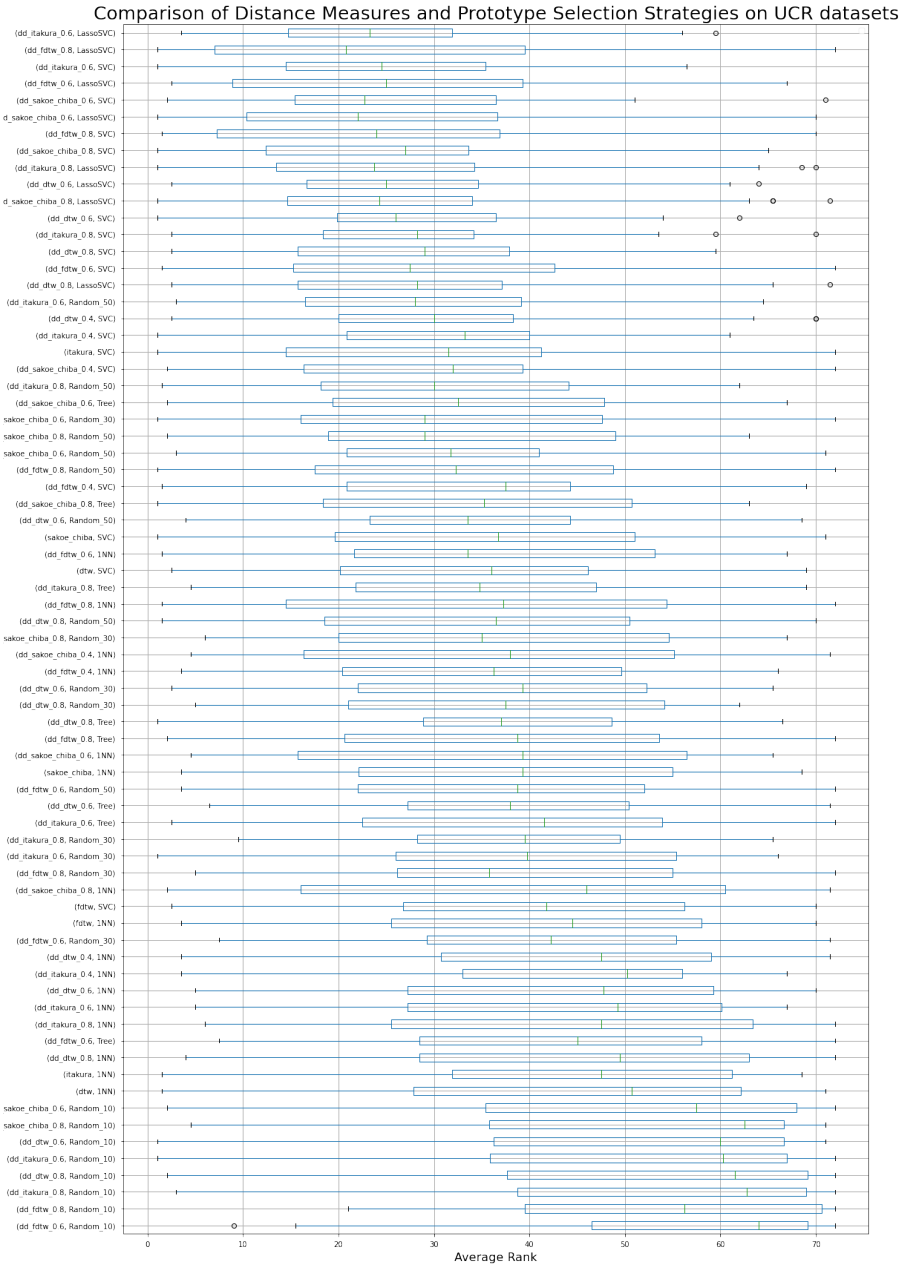


Figure A.2: Average rank of combinations of distance measures and prototype selection strategies on UCR datasets.