Politecnico di Torino

Mathematics in Machine Learning

Data Science and Engineering

Multivariate Time Series Classification: Australian Sign Language

---

Professors:
Gasparini Mauro
Vaccarino Francesco

Students:
Scarciglia Lorenzo
Silvi Andrea

Academic Year 2021/2022

Contents

List of Figures

List of Tables

List of Pseudocodes

Acronyms

$\boldsymbol{\sigma}$ standard deviation. 2, 8–10, 12

5DT Fifth-Dimension Technology. 4

AUSLAN *Aus*tralian sign *Lan*guage. 1, 4, 33

DAG directed acyclic graph. 23

DTW Dynamic Time Warping. 1, 2, 5, 17, 18, 22, 30

ENU East, North, Up. 2, 6, 8, 11

Eros *E*xtended F*ro*benius norm. 2, 5, 15

GAK Global Alignment Kernel. 1, 5, 22, 30

K-NN K-Nearest Neighbors. 1, 2, 5, 17, 18, 30

K-PCA Kernel PCA. 1, 2, 5, 14, 15, 21, 28–30, 33

KEros Kernel Eros. 1, 15, 28–31, 33

KKT Karush-Kuhn-Tucker. 20

ML Machine Learning. 5, 30, 33

MTS Multivariate Time Series. 1, 2, 4–7, 12, 13, 15–18, 22, 25, 28, 30–33

OVO one-versus-one. 2, 23

OVR one-versus-rest. 2, 23

PCA Principal Component Analysis. 1, 2, 5, 13–15, 28–33

PSD positive semidefinite. 14–16

RF Random Forests. 1, 2, 5, 23–25, 28–32

RKHS reproducing kernel Hilbert space. 14, 21

SVD Singular Value Decomposition. 15

SVM Support Vector Machine. 1, 2, 5, 19–23, 28–33

TSF Time Series Forest. 1, 2, 5, 25, 26, 30, 32, 33

1 Introduction

This project aims at solving the problem of classifying a collection of hand signs data belonging to the *Aus*tralian sign *Lan*guage (AUSLAN). This is a dialect of the more common British Sign Language and contains more than 4000 signs, that are used separately or combined in order to describe different words. Signs can be one or two-handed and can contain common movement patterns in three dimensions, so classifying them based on data gathered from examples can be a very interesting task. The dataset used is the *Australian Sign Language signs (High Quality) Data Set* [1], available at the UCI Machine Learning Repository.

Figure 1: *Sign Language Australia* provides different programs for learning AUSLAN.

We first start with a brief description of the dataset and the different approaches we adopt to solve this classification problem. We then perform in Section 2 some data exploration and visualization to better understand the dataset. We continue in Section 3 by describing the theory behind all the pre-processing techniques we adopt. In the following Section 4 we explore the theory of the classification algorithms we use in our approaches, while in Section 5 we present the techniques we follow to evaluate models performances. In Section 6 we report the results of the different approaches we follow and finally in Section 7 we sum up our findings. The code we develop is available at the following GitHub repository: **https://github.com/notlosca/auslan-classification**.

1.1 AUSLAN dataset

The AUSLAN dataset contains 95 different signs, considered as labels, picked out from more than 4000. These signs can either be performed with one single hand (*one-handed*), or with both hands. The latter signs can also be further divided into specular ones (*double-handed*), or non specular ones (*two-handed*). Out of the 95 different signs, 59 of them are one-handed, 27 are double-handed and 9 are two-handed. Data were captured from a native right-handed signer using two 5DT gloves with position trackers, a right and a left one. These high-quality position trackers allowed to capture 11 features for each hand (22 in total), based on the location and orientation of each hand in space and the openness of each finger: *x, y, z, roll, pitch, yaw, thumb, forefinger, middle finger, ring finger, little finger*. The recording session lasted 9 weeks. Each week 3 observations for each sign were recorded. Hence, at the end of the 9^{th} week 27 examples for each sign were recorded. The overall number of recorded signs adds up to 2565.
Each observation is a MTS (Multivariate Time Series) of 22 variables and variable length. The average length of each data is approximately 57 frames.

1.2 Problem description and proposed approaches

The task we attempt to solve is a multiclass classification problem of a temporal dataset composed of 2565 MTS, each represented by a different amount of frames described by 22 predictors. In order to solve it, before following a classic Machine Learning approach, we need to take into account the temporal representation of the data, meaning that each sample is described by points ordered in time. Thus, we design two different approaches, both based on classic Machine Learning methods, which are extended in order to work with MTS:

- the first one, which we call *"PCA based"* approach, relies on classic ML algorithms such as Support Vector Machine [2] and Random Forests [3], which need as input a dataset containing observations described by feature vectors. In order to obtain this kind of representation from a MTS dataset, we test two different solutions. The first one is based on Kernel PCA [4] [5], which uses as kernel one based on a PCA-based similarity measure for MTS called *Eros* (*E*xtended F*ro*benius *s* norm) [6] [7]. Instead, the second one interpolates and concatenates each time series in order to obtain a fixed size feature vector, whose dimension can be then reduced with the usual PCA algorithm.

- The second one, which we call *"Raw MTS"* approach, relies on extensions of famous ML algorithms such as the K-Nearest Neighbors classifier [8], SVM and RF, which make these methods work directly with MTS datasets. K-NN leverages on *Dynamic Time Warping* (DTW) [9], a similarity metric which works with MTS of different lenghts; SVM considers the Global Alignment Kernel (*GAK*) [10], which is closely related to DTW; the RF extension, called *Time Series Forest* [11], extracts different statistics from random intervals of the MTS and then trains a Random Forests classifier over the newly obtained dataset.

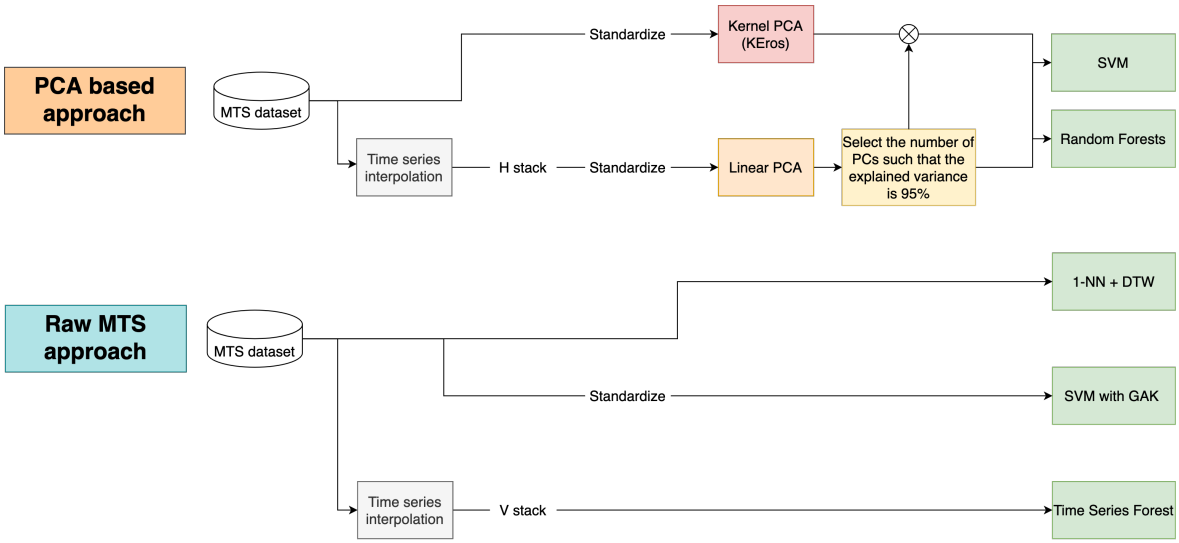


Figure 2: Visual representation of the approaches presented in Section 1.2.

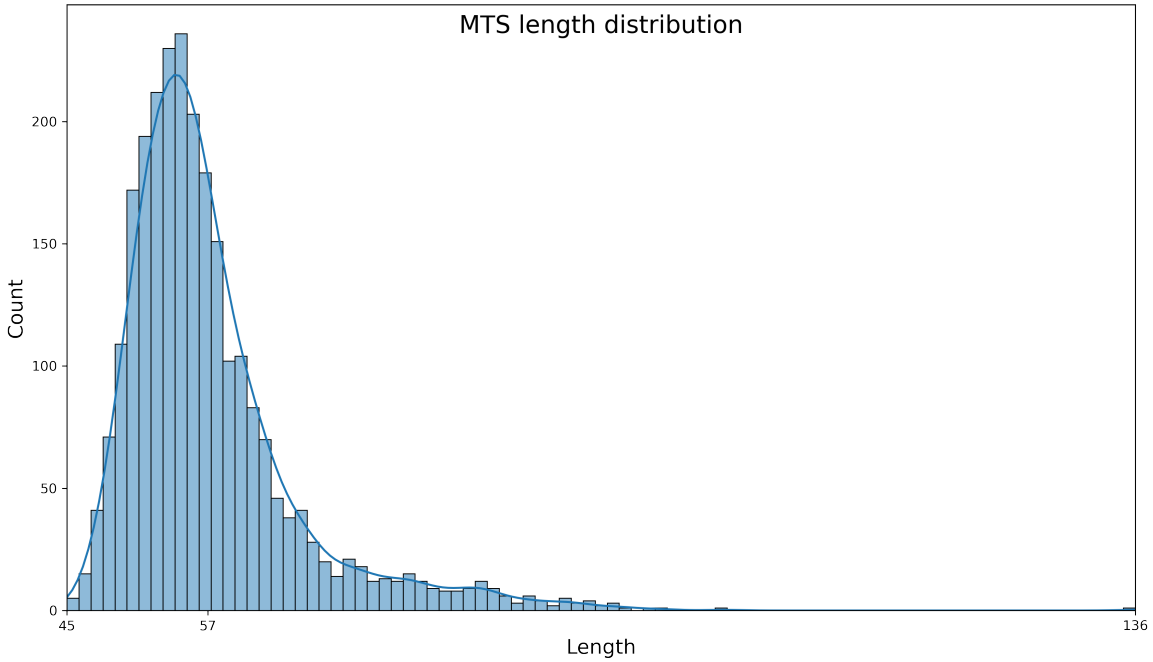


Figure 3: MTS length distribution.

2 Data Exploration

The dataset we are analyzing is well balanced, since for each of the 95 signs there are 27 examples. As for the MTS themselves, their mean length, as in their average number of frames, is approximately 57.29. From Figure 3 we denote that a large majority of the data length is concentrated around the mean, but it also presents a positive skewness. Moreover, in the distribution tail we can denote an outlier. To better understand this finding, in Figure 4 we report the length boxplot for each label: we can observe that most of the signs, barring some longer examples like *building* (labelled as 7), are distributed around the mean value. Regarding the outlier found before, it belongs to the sign *hurry* (labelled as 35), which on average follows the length distribution of the other classes, meaning that the outlier is caused by a longer recording than usual. Since we do not want to truncate data in order not to lose relevant information, and moreover the length of the data recorded depends on how fast the signer performs their actions, we decide to keep the raw data.

The MTS are described by 22 features, 11 for each hand, all containing continuous values. The features are the following:

- `x`, `y`, `z` describe in meters the hand distance from the origin in 3D space. The origin is taken as a point slightly below the chin;
- `roll`, `pitch`, `yaw` describe the hand rotation in 3D space along the (x,y,z) axes, assuming the ENU coordinate system (reported in Figure 5). In this way `roll` describes a counterclockwise rotation around axis y, `pitch` describes a counterclockwise rotation around axis x, and finally `yaw` describes a counterclockwise rotation around axis z. The reference position of the hand is palm flat pointing towards the positive y axis and corresponds to (`roll`, `pitch`, `yaw`) = (0, 0.5, 0.5);
- `thumb`, `forefinger`, `middle finger`, `ring finger`, `little finger` describe the openness of each finger, where 0 means totally flat, while 1 means totally bent.

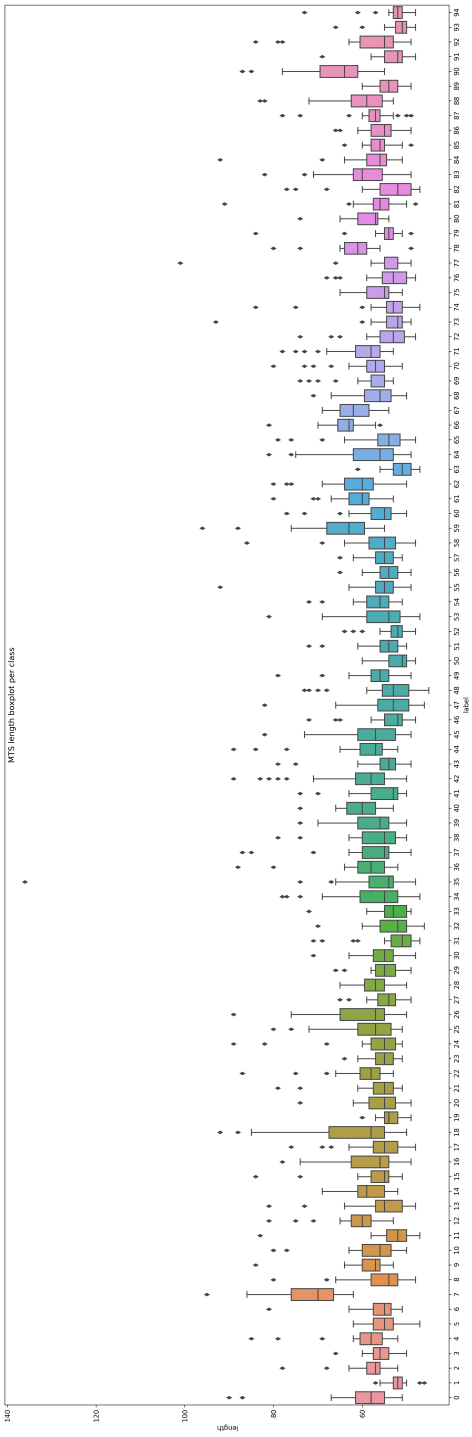


Figure 4: MTS length boxplot for each class.

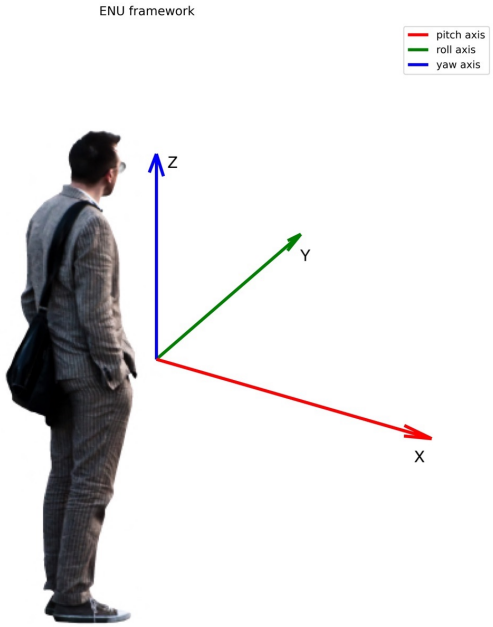


Figure 5: The ENU coordinate system.

Features	**Min**	**Max**	**Mean**	$\boldsymbol{\sigma}$
x left	-0.342	0.284	-0.055	0.043
x right	-0.359	0.506	0.051	0.063
y left	-0.21	0.452	-0.113	0.08
y right	-0.189	0.555	-0.044	0.139
z left	-0.397	0.28	-0.138	0.065
z right	-0.454	0.292	-0.121	0.089
roll left	0.003	0.998	0.598	0.089
roll right	0.001	0.998	0.437	0.09
pitch left	0.119	0.993	0.308	0.133
pitch right	0.037	0.998	0.39	0.167
yaw left	0.064	0.675	0.463	0.052
yaw right	0.001	0.988	0.548	0.088
thumb left	0.0	1.0	0.09	0.23
thumb right	0.0	1.0	0.279	0.363
forefinger left	0.0	1.0	0.053	0.171
forefinger right	0.0	1.0	0.057	0.204
middle finger left	0.0	1.0	0.03	0.145
middle finger right	0.0	1.0	0.226	0.355
ring finger left	0.0	1.0	0.055	0.181
ring finger right	0.0	1.0	0.139	0.306
little finger left	0.0	1.0	0.071	0.193
little finger right	0.0	1.0	0.263	0.358

Table 1: Some statistics for each feature.

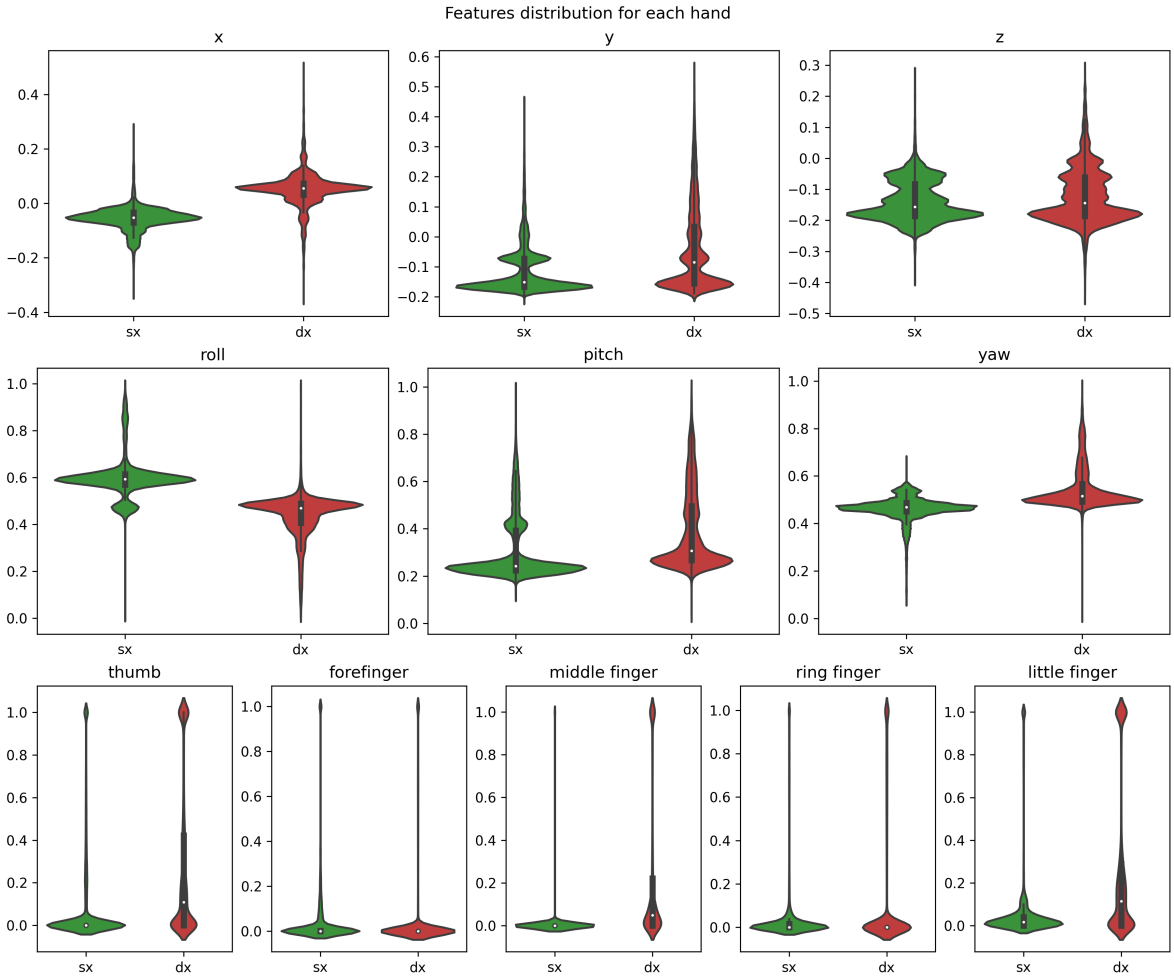


Figure 6: Features distributions for each hand.

In Table 1 we report some statistics for each feature of the whole dataset, dropping the temporal axis from it, thus obtaining a fictitious dataset of dimensions $(146949, 22)$. As we can see, all features except for `x`, `y` and `z` lie between 0 and 1, while the position coordinates are theoretically unbounded.

In Figure 6 we also report the estimated underlying distributions for each feature from the aforementioned dataset, comparing left and right hands. Some interesting observations we found are:

- `x` presents two fairly similar distributions, but flipped around 0, since it describes how far from the origin the hands are on the x axis, clearly shown in Figure 5;
- both `y` and `z` have similar distributions for the left and right hands, except for the longer positive tail of the right hand. Since the signer is right handed and most signs are one-handed, the right hand tends to occupy a larger range of space, as shown also by the standard deviation in Table 1;
- `roll` and `yaw` again are mirrored around 0, due to articular constraints, while `pitch` presents again a longer tail on the right hand, because of the right-handed signer;
- most of the features of the fingers of the right hand present two peaks: the higher one around 0, meaning that most times those fingers are totally flat, and a smaller one around 1, representing

	hello		***God***		***where***	
Features	**Mean**	$\boldsymbol{\sigma}$	**Mean**	$\boldsymbol{\sigma}$	**Mean**	$\boldsymbol{\sigma}$
x left	-0.095	0.011	-0.124	0.061	-0.143	0.038
x right	0.073	0.071	0.078	0.048	0.111	0.021
y left	-0.115	**0.002**	0.020	0.022	0.014	**0.027**
y right	-0.056	**0.172**	0.068	0.138	-0.011	**0.023**
z left	-0.043	0.001	-0.045	0.021	0.007	0.021
z right	-0.019	0.03	-0.037	0.010	-0.033	0.032
roll left	0.489	**0.001**	0.472	0.082	0.521	**0.174**
roll right	0.39	**0.037**	0.389	0.079	0.469	**0.184**
pitch left	0.397	**0.003**	0.535	0.023	0.506	**0.023**
pitch right	0.449	**0.163**	0.513	0.112	0.486	**0.042**
yaw left	0.504	**0.001**	0.495	0.017	0.514	**0.017**
yaw right	0.522	**0.054**	0.552	0.036	0.523	**0.013**
thumb left	0.445	0.021	0.175	**0.364**	0.589	0.212
thumb right	0.282	0.131	0.196	**0.368**	0.769	0.353
forefinger left	0.0	0.0	0.198	**0.390**	0.081	0.051
forefinger right	0.0	0.0	0.052	**0.215**	0.000	0.000
middle finger left	0.0	0.0	0.179	**0.375**	0.000	0.000
middle finger right	0.0	0.0	0.187	**0.387**	0.000	0.000
ring finger left	0.0	0.0	0.204	**0.384**	0.001	0.002
ring finger right	0.0	0.0	0.173	**0.373**	0.000	0.000
little finger left	0.0	0.0	0.216	**0.383**	0.006	0.005
little finger right	0.095	0.022	0.332	**0.312**	0.109	0.043

Table 2: Mean and $\boldsymbol{\sigma}$ of each feature for the signs *hello*, *God*, and *where*.

the fingers as totally bent. The left hand, being used much less, mostly does not present this behaviour. This is also clearly shown by the means and $\boldsymbol{\sigma}$s in Table 1.

We finally plot a couple of frames of the evolution in time of some examples for each type of sign in Figure 7, in particular:

- one example of the sign *hello*, belonging to the one-handed family;
- one example of the sign *God*, belonging to the two-handed family;
- one example of the sign *where*, belonging to the double-handed family.

From this last Figure 7 and Table 2 we can easily notice some differences between signs of different nature: in the *hello* column of Table 2, we can see how the features `y`, `pitch`, `roll` and `yaw` have a standard deviation with a higher order of magnitude for the right hand features with respect to the left ones, since it is a one-handed sign, while a double-handed sign like *where* has a $\boldsymbol{\sigma}$ of similar order of magnitude for these features. It is also interesting to note how a more complex sign like *God* presents a way higher variance of values in the fingers features, both left and right, with respect to the two other less complex signs.

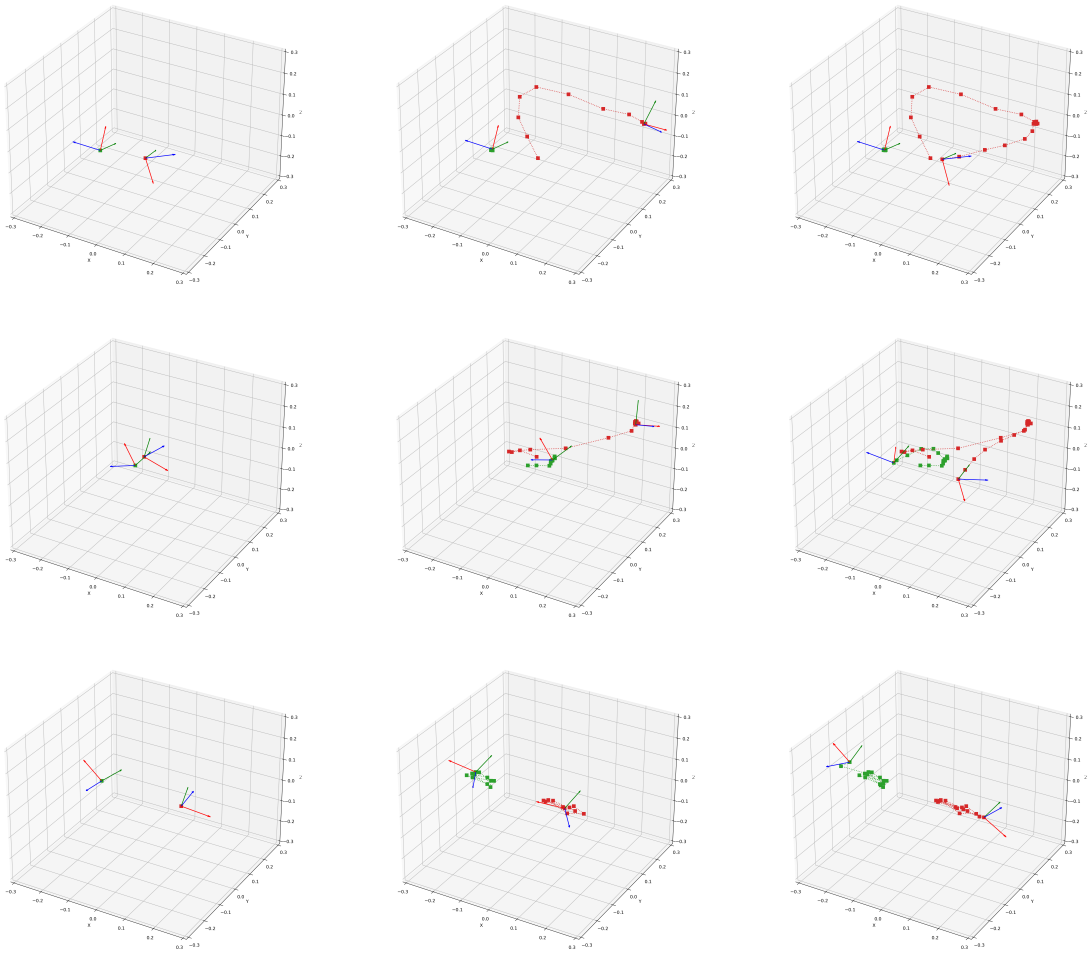


Figure 7: Some frames showing the evolution of some examples of signs belonging to different families. In the first row, we report the **one-handed** sign *hello*, in the second the **two-handed** sing *God*, and finally in the last row the **double-handed** sign *where*. The left hand is colored in green whilst the right one is red. The vectors represent the rotation of the hand concerning the ENU framework presented in Figure 5.

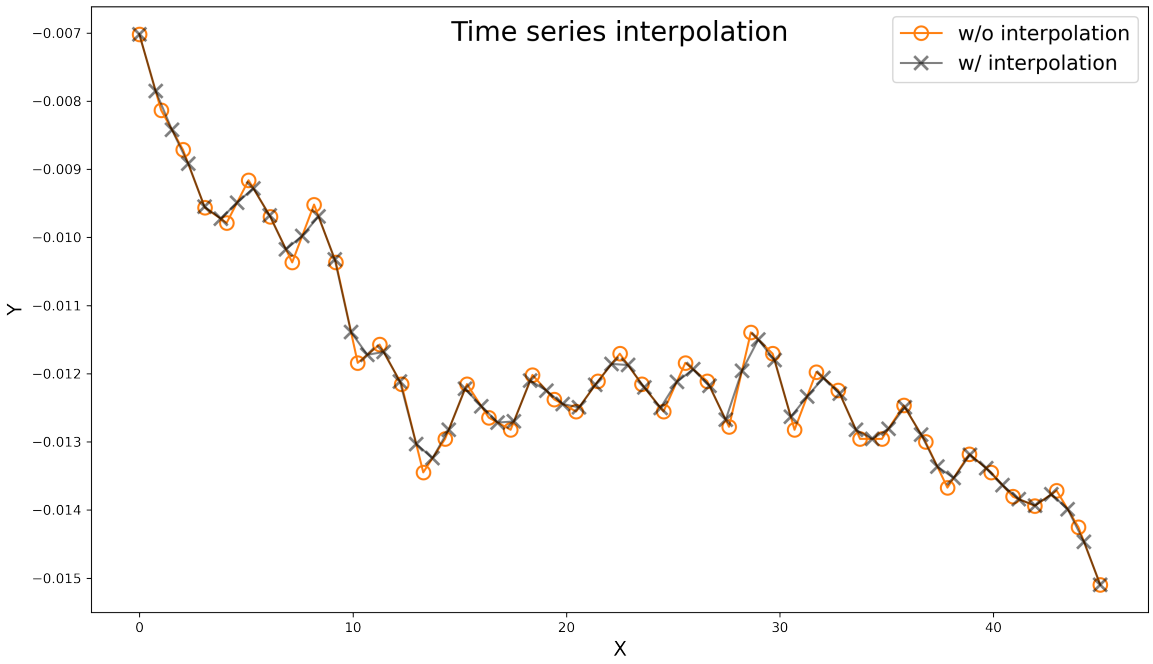


Figure 8: An example of the result of linear interpolation from a time series (in orange) of length 45 to a new one (in black) of length 60.

3 Data Pre-processing

In this section we describe all the different pre-processing techniques adopted in both approaches described in Section 1.2. Note that different approaches rely on different pre-processing techniques, and both pipelines are going to be presented in more details in Section 6.

3.1 Z-score standardization

In order to have features of the same magnitude, Z-score standardization is applied for each column $\mathbf{x}$ by subtracting the mean and dividing by the standard deviation of that column, as:

$$\mathbf{x}_{\text{scaled}} = \frac{\mathbf{x} - \mu}{\sigma} \tag{1}$$

In the case of MTS, we calculate μ and σ of the whole dataset, which are reported in Table 1. Then, for each MTS, the Formula 1 is applied column-wise.

3.2 Time series interpolation

For classification methods that require data to be of the same length, we apply a linear interpolation transformation.

Let a MTS be in the form of $Y = [\mathbf{y}_i]_{i=1}^{n} \in \mathbb{R}^{m,n}$, where $\mathbf{y}_i$ is the column vector of length m of the i-th predictor, and let its indices be in the form $\mathbf{x} = [0, \dots, m-1]^T \in \mathbb{R}^m$. In order to obtain the interpolated MTS $\hat{Y}$ of a fixed length l from Y, we firstly find the intervals length by dividing m by $l-1$, obtaining $l-1$ intervals of length $m/(l-1)$. Then, we obtain the new set of indices

$$\mathbf{x}_{\text{interp}} = [i\, m/(l-1)]^T,\ \forall i \in [0, l-1],\ \mathbf{x}_{\text{interp}} \in \mathbb{R}^l \tag{2}$$

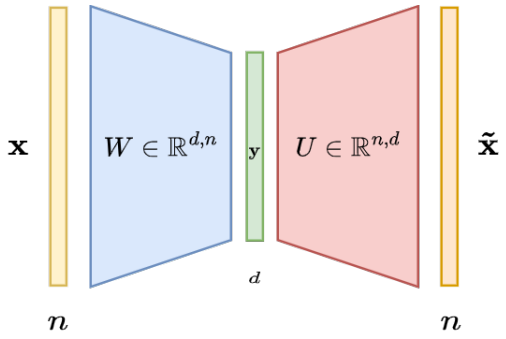


Figure 9: PCA interpretation as an autoencoder.

For each temporal index $\hat{x}_i \in \mathbf{x}_{\text{interp}}$ and for each predictor column $\mathbf{y}_j$, we calculate its respective value $\hat{y}_{i,j}$

$$\frac{\hat{y}_{i,j} - y_{a,j}}{\hat{x}_i - x_a} = \frac{y_{b,j} - y_{a,j}}{x_b - x_a} \implies \hat{y}_{i,j} = y_{a,j} + (\hat{x}_i - x_a)\frac{y_{b,j} - y_{a,j}}{x_b - x_a} \tag{3}$$

where $(x_a, y_{a,j})$ and $(x_b, y_{b,j})$ are two consecutive points of the original time series for which $x_a \leq \hat{x}_i \leq x_b$, $x_a < x_b$. An example on a univariate time series is reported in Figure 8.

We finally obtain a new MTS $\hat{Y} = [\hat{\mathbf{y}}_i]_{i=1}^n \in \mathbb{R}^{l,n}$ where each $\hat{\mathbf{y}}_i$ contains the interpolated values $\hat{y}_{i,j}$ described before.

3.3 Dimensionality reduction - PCA

Principal Component Analysis [4] is a linear dimensionality reduction technique $\mathbf{x} \mapsto W\mathbf{x}$ which maps data $\mathbf{x} \in \mathbb{R}^n$ into a lower dimensionality space of dimension $d < n$ through the use of a matrix $W \in \mathbb{R}^{d,n}$ called the *encoding* matrix.

The aim is to minimize the reconstruction error between the original point $\mathbf{x}$ and the reconstructed one $W\mathbf{x} \mapsto \tilde{\mathbf{x}} = UW\mathbf{x}$, where $U \in \mathbb{R}^{n,d}$ is the *decoding* matrix. PCA can thus be seen as a single layer autoencoder with linear activation functions. This idea is reported in Figure 9. More specifically, given the tabular dataset $X \in \mathbb{R}^{N,n}$ whose i-th row is the i-th observation's feature vector $\mathbf{x}_i^T$, the aim is minimizing the reconstruction error as

$$\underset{W,U}{\operatorname{argmin}} \sum_{i=1}^{N} ||\mathbf{x}_i - UW\mathbf{x}_i||_2^2 \tag{4}$$

It can be proven that the solution to the PCA Problem 4 is to set the columns of U to be $[\mathbf{u}_1, \ldots, \mathbf{u}_d]$ and to set $W = U^T$, where $\mathbf{u}_1, \ldots, \mathbf{u}_d$ are the d leading eigenvectors of the matrix $A = X^T X = \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T$.

It can be observed that the argument of the Problem 4 can be rewritten as, using $W = U^T$,

$$\sum_{i=1}^{N} ||\mathbf{x}_i - UW\mathbf{x}_i||_2^2 = \sum_{i=1}^{N} \left[||\mathbf{x}_i||^2 - \text{trace}(U^T \mathbf{x}_i \mathbf{x}_i^T U)\right] \tag{5}$$

Since the first argument of the sum is fixed, and $\text{trace}(A) + \text{trace}(B) = \text{trace}(A + B)$, an equivalent PCA problem is derived:

$$\underset{U \in \mathbb{R}^{n,d} | U^T U = I}{\operatorname{argmax}} \text{trace}\left(U^T \left(\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T\right) U\right) = \text{trace}\left(U^T (A)\, U\right) \tag{6}$$

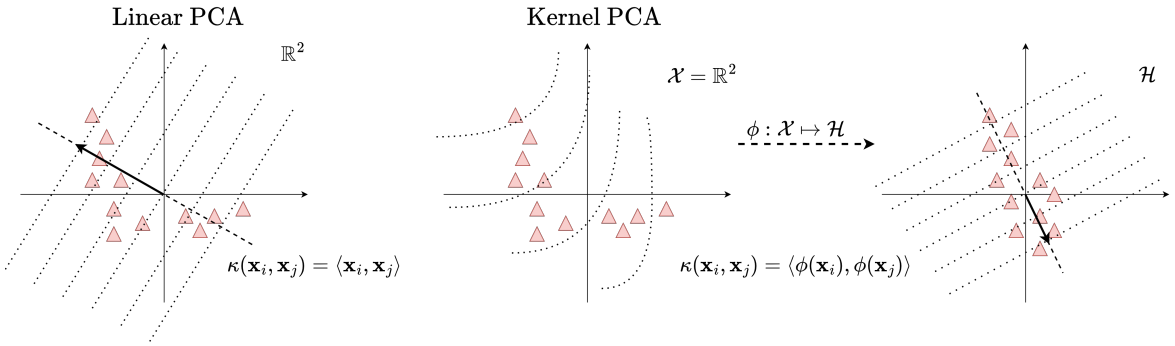


Figure 10: Basic idea of Kernel PCA: by using a non-linear kernel function κ instead of the standard dot product, PCA is implicitly performed in a higher dimensional feature space $\mathcal{H}$ (RKHS), non linearly related to the original feature space.

Since $A = \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T$ is symmetric and positive semidefinite, we can write it using its spectral decomposition $A = VDV^T$, where D is a diagonal matrix containing the non-negative eigenvalues of A, sorted in non-increasing order, and the columns $[\mathbf{v}_1, \ldots, \mathbf{v}_d]$ of V are the corresponding orthogonal eigenvectors. Then, it can be claimed that the solution to Problem 6 is the matrix U whose columns are the d eigenvectors of A corresponding to the d largest eigenvalues. These eigenvectors $[\mathbf{v}_1, \ldots, \mathbf{v}_d]$ correspond to the principal components $[\mathbf{pc}_1, \ldots, \mathbf{pc}_d]$. They identify the directions along which our data has maximal variance in decreasing order. It can be proven that each eigenvalue λ_i corresponds to the variance of the data along the direction found by the corresponding principal component. Hence, each principal component $\mathbf{pc}_i$ explains a fraction $(\lambda_i / \sum_{i=1}^{d} \lambda_i)$ of the variance of the data.

It is common practice to center the data ($\mu_j = \sum_{i=1}^{N} x_{ij} = 0$, for $1 \leq j \leq n$) and then use as the matrix A the sample covariance matrix $\hat{\Sigma}$, defined as

$$\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T \tag{7}$$

3.3.1 Kernel PCA

Kernel PCA [5] extends PCA by performing a non linear form of PCA through the use of *kernel methods*. If we define the matrix $B = XX^T \in \mathbb{R}^{N,N}$, where

$$B_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{8}$$

and we also suppose that $\mathbf{u}$ is an eigenvector of B, that is $\lambda \mathbf{u} = B\mathbf{u}$, we can see that, by multiplying the equality by X^T and using the definition of B, we obtain $X^T B \mathbf{u} = X^T X X^T \mathbf{u} = \lambda X^T \mathbf{u}$. Then, by substituting $X^T X = A$ as defined in the previous section, we end up with $A(X^T \mathbf{u}) = \lambda(X^T \mathbf{u})$. Hence, $X^T \mathbf{u} / ||X^T \mathbf{u}||$ is an eigenvector of A with eigenvalue λ, meaning that we can obtain the PCA solution by calculating the eigenvalues of B instead of A.
By looking at the definition of B in Equation 8 we can see that we only need to know how to calculate the inner product between observations in order to obtain the PCA solution. This enables us to *kernelize* PCA by defining a mapping from the initial domain to a reproducing kernel Hilbert space [12] as $\phi : \mathbb{R}^n \mapsto \mathcal{H}$ and its relative kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. We then obtain the Kernel matrix $K \in \mathbb{R}^{N,N}$ where $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Hence, we obtain the principal components by solving the eigenvalue problem $\lambda \mathbf{u} = K\mathbf{u}$ with K instead of B. A visual representation of the main idea behind K-PCA is reported in Figure 10.

Note that, even if the original data are zero-mean centered, they are not guaranteed to be centered in feature space. Since data needs to be centered in order to perform an effective PCA, we center the kernel matrix as

$$\bar{K} = K - \mathbf{1}_N K - K\mathbf{1}_N + \mathbf{1}_N K \mathbf{1}_N \tag{9}$$

where $\mathbf{1}_N$ is an $N \times N$ matrix filled of $1/N$ as values.

3.3.2 Kernel PCA for MTS: KEros

Following [7], we decide to use as kernel function a similarity measure that works between MTS of different lengths based on PCA called Eros [6]. Let $A \in \mathbb{R}^{m_A,n}$ and $B \in \mathbb{R}^{m_B,n}$ be two MTS, and let $V_A^T = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$ and $V_B^T = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ be the two right eigenvector matrices obtained from applying SVD to the sample covariance matrices of A and B respectively. Then, the Eros similarity measure between A and B is defined as

$$Eros(A, B, \mathbf{w}) = \sum_{i=1}^{n} w_i |\langle \mathbf{a}_i, \mathbf{b}_i \rangle| = \sum_{i=1}^{n} w_i |\cos\theta_i| \tag{10}$$

where $\mathbf{w}$ is a normalized weight vector based on the eigenvalues of the MTS dataset. All the i-th eigenvalues obtained from all the MTS are then aggregated in order to obtain w_i. Each element w_i is the average of the i-th eigenvalues of each MTS in the dataset. The pseudocode for computing the vector $\mathbf{w}$ is reported in Algorithms 1 and 2.

Algorithm 1 MTS preprocessing for Eros.

Require: the dataset of N MTS examples in the form $X_i \in \mathbb{R}^{m_i,n}$.
  for $i = 1$ to N **do**
    $X_i \leftarrow$ the i-th MTS in the dataset;
    $\hat{\Sigma}_i \leftarrow$ the covariance matrix of X_i;
    $[U_i, D_i, V_i^T] \leftarrow \mathbf{SVD}(\hat{\Sigma}_i)$;
    $\boldsymbol{\lambda}_i \leftarrow$ the normalized eigenvalues in D_i;
    store V_i^T as the i-th right eigenvector matrix of X_i;
  end for
  return $[\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N]$, $[V_1^T, \ldots, V_N^T]$

Algorithm 2 Computing the weight vector $\mathbf{w}$ for Eros.

Require: the eigenvalues $[\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N]$ obtained in Algorithm 1, where $\boldsymbol{\lambda}_i \in \mathbb{R}^n$.
  for $i = 1$ to n **do**
    $w_i \leftarrow \frac{1}{N} \sum_{j=1}^{N} \lambda_{i,j}$;
  end for
  for $i = 1$ to n **do**
    $w_i \leftarrow w_i / \sum_{j=1}^{n} w_j$;
  end for
  return $\mathbf{w} = [w_i, \ldots, w_n]^T$

The Kernel matrix is constructed as $K^{Eros}(i, j) = Eros(X_i, X_j, \mathbf{w})$, for all the MTS X_i and X_j in the dataset. Note that Eros is not a proper distance metric since it does not satisfy the triangle inequality, and it cannot be represented in the form of a dot product, but, as long as the matrix K^{Eros} defined above is positive semidefinite and symmetric, it can be utilized as a Gram Matrix for Kernel PCA [13]. Note from Formula 10 that $Eros(A, B, \mathbf{w}) = Eros(B, A, \mathbf{w})$, hence K^{Eros} is symmetric. To make K^{Eros} also positive semidefinite, if it is not already PSD after calculating it (i.e. all of its eigenvalues are not ≥ 0), we adopt the first naïve approach from [14], which is $\bar{K}^{Eros} \leftarrow K^{Eros} + \delta I$,

with δ sufficiently larger in absolute value than the most negative eigenvalue of K^{Eros}. We finally center the K^{Eros} matrix in feature space following Formula 9. The pseudocode to compute K^{Eros} is reported in Algorithm 3. This matrix can then be used as the Gram matrix to obtain the principal components as described in Section 3.3.1.

Algorithm 3 Computing the kernel matrix $\bar{K}^{Eros}$.

Require: $\mathbf{w}$, the dataset of N MTS examples.
  for $i = 1$ to N **do**
    for $j = 1$ to N **do**
      $K^{Eros}(i, j) \leftarrow Eros(X_i, X_j, \mathbf{w})$;
      $K^{Eros}(j, i) \leftarrow K^{Eros}(i, j)$;
    end for
  end for
  if K^{Eros} is not PSD **then**
    $K^{Eros} \leftarrow K^{Eros} + \delta I$
  end if
  $\bar{K}^{Eros} \leftarrow K^{Eros} - \mathbf{1}_N K^{Eros} - K^{Eros}\mathbf{1}_N + \mathbf{1}_N K^{Eros}\mathbf{1}_N$
  return $\bar{K}^{Eros}$

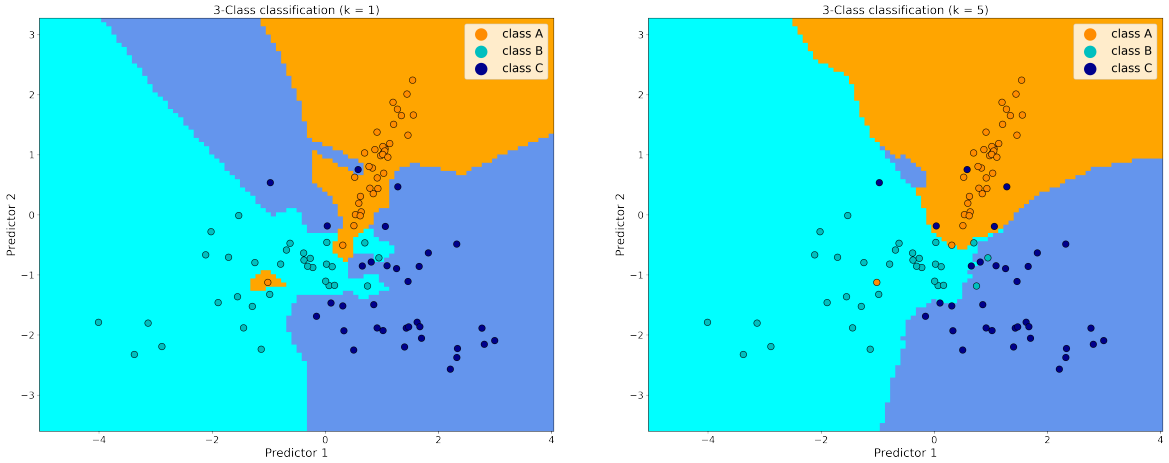


Figure 11: An example of how the class boundaries of the K-NN classification results vary with respect to the number of neighbors.

4 Algorithms

This section aims at presenting the theory of the classification algorithms used in the experiments reported in Section 6. Since the problem is multi-label classification, all the algorithms adopted are *supervised*, meaning that a classifier is trained with a set of labelled data aiming at minimizing the difference between the predictions over the data and their actual labels.

4.1 K-Nearest Neighbors

K-Nearest Neighbors is the only non-parametric classification method that we adopt. The main idea behind it is to memorize the training set and then to predict the label of any new instance of data by assigning the *most common* label of its k *closest* neighbors. This is based on the assumption that points close in the features domain are likely to have the same label. A distance metric is used to calculate distances between data points. Standard K-NN adopts the Euclidean distance $||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\sum_{k=1}^{n}(x_{i,k} - x_{j,k})^2}$ as metric, but also custom metrics can be used.
When $k = 1$, the algorithm considers only the closest neighbor when labelling new data. If $k > 1$, then a majority vote approach is used, based on the labels of the k closest points. An example of the different behaviour concerning the value of k is reported in Figure 11. The vote of each neighbor can also be weighted by the distance between the new data and the neighbor.
Although K-NN is a fairly simple algorithm, some advantages are its incrementality and its computational complexity of training of $\mathcal{O}(1)$, since we only need to store data. On the other hand, the computational complexity of testing when using the Euclidean distance as metric is $\mathcal{O}(Nn + Nk)$, where N is the number of training examples, n the number of predictors and k the number of neighbors, chosen as hyperparameter. This is the simplest *brute force* K-NN implementation, which can be further optimized by using algorithms which rely on tree structures.

4.1.1 Similarity metric for MTS: Dynamic Time Warping

When dealing with a MTS dataset, the Euclidean distance is not a suitable distance metric, since:

- it is only defined on vectors of the same length, while time series often have different lengths;
- it compares values at each time point independently, whereas time series values are correlated.

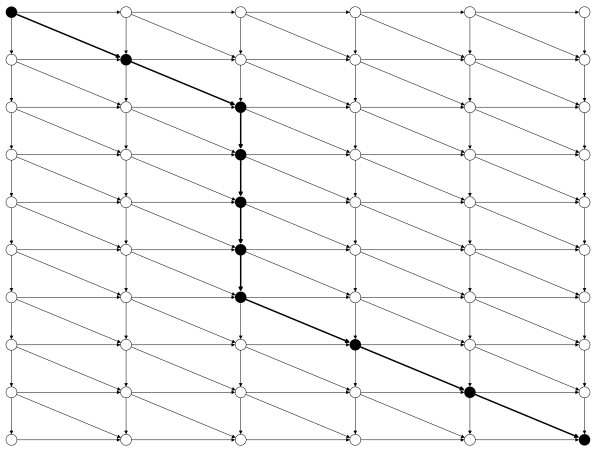


Figure 12: Graphical representation of the DTW scores. The graph has as nodes the entries of the cost matrix C computed between two MTS of lengths, respectively, $m_i = 10$ and, $m_j = 6$. The shortest path between the source entry $(1, 1)$, located at the top left, and the target entry $(10, 6)$, located at the bottom right, is the warping path associated with the minimum score, i.e. the DTW score.

Dynamic Time Warping [9] is a metric suited for time series that solves the two issues reported above. It is a widely used metric that has been introduced for speech recognition, but is used in many fields that require sequence alignment. It aims at calculating an *optimal matching* between two sequences, given some restrictions.
Considering two MTS $X_i = [\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,m_i}]^T \in \mathbb{R}^{m_i,n}$ and $X_j = [\mathbf{x}_{j,1}, \ldots, \mathbf{x}_{j,m_j}]^T \in \mathbb{R}^{m_j,n}$, firstly a cost matrix $C \in \mathbb{R}^{m_i,m_j}$ is computed as $C_{u,v} = f(\mathbf{x}_{i,u}, \mathbf{x}_{j,v})$, with $1 \leq u \leq m_i$, $1 \leq v \leq m_j$, and f as the squared Euclidean distance.
We then define a warping path as the sequence $p = (p_1, \ldots, p_S)$ satisfying the following constraints:

- $\forall s \in \{1, \ldots, S\}$, $p_s = (u_s, v_s) \in \{1, \ldots, m_i\} \times \{1, \ldots, m_j\}$;
- $p_1 = (1, 1)$ (starting point) and $p_S = (m_i, m_j)$ (finish point);
- $\forall s \in \{1, \ldots, S-1\}$, $p_{s+1} - p_s \in \{(0, 1), (1, 0), (1, 1)\}$

The cost associated with a warping path p is defined as the sum of the costs of the states visited by the path as

$$C_p(X_i, X_j) = \sum_{s=1}^{S} C_{u_s,v_s} \tag{11}$$

Finally, the Dynamic Time Warping score is defined as the minimum cost among all the warping paths $p \in \mathcal{P}$:

$$\text{DTW}(X_i, X_j) = \min_{p \in \mathcal{P}} C_p(X_i, X_j) \tag{12}$$

A graphical representation of the warping path associated with the minimum cost and the relative DTW is presented in Figure 12.

One of the main limitations of DTW is its computational complexity of $\mathcal{O}(m_i m_j n)$, where m_i and m_j are the lengths of two time series and n the number of predictors. This makes K-NN's computational complexity of testing increase to $\mathcal{O}(N\, m_i m_j n + Nk)$, with N being the number of training examples. Moreover, faster implementations of K-NN based on tree-structures cannot be exploited since DTW does not satisfy the triangular inequality.

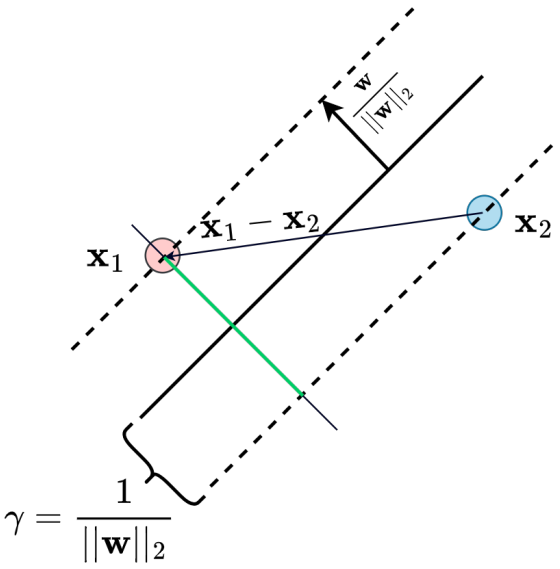


Figure 13: SVM margin derivation. The green line is the projection of the vector $\mathbf{x}_1 - \mathbf{x}_2$ on the normal vector $\mathbf{w}/||\mathbf{w}||_2$.

4.2 SVM

Support Vector Machine is a well established learning algorithm that aims at diving points of different classes by finding the best separating hyperplane. In the binary classification setting, data $\mathbf{x}_i \in \mathbb{R}^n$ are labelled as $y \in \{-1, 1\}$, and the problem becomes finding the best oriented separating hyperplane such that data points on one side are all labelled as -1 and on the other are labelled as 1. SVM finds as best hyperplane the one that maximizes the distance between the hyperplane and the closest data instances belonging to different classes. These points are called *support vectors* and have the most influence on the position of the separating hyperplane.

Let $X = [(\mathbf{x}_i, y_i)]_{i=1}^N$ be a dataset such that $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \{-1, 1\}$, we say that X is linearly separable by a separating hyperplane $\pi : (\mathbf{w}, b)$ if

$$\exists\, \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R} : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0 \;\; \forall i \tag{13}$$

where $\langle \mathbf{w}, \mathbf{x}_i \rangle = \mathbf{w} \cdot \mathbf{x}_i$ is the dot product.
We define as *canonical hyperplanes* the hyperplanes passing through $\langle \mathbf{w}, \mathbf{x}_i \rangle + b = 1$ and $\langle \mathbf{w}, \mathbf{x}_i \rangle + b = -1$, on which the closest points of different classes lie. The region between the canonical hyperplanes is called the *margin band*.
More specifically, let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two support vectors lying on the two canonical hyperplanes, thus $\langle \mathbf{w}, \mathbf{x}_1 \rangle + b = 1$ and $\langle \mathbf{w}, \mathbf{x}_2 \rangle + b = -1$. We easily deduce that $\langle \mathbf{w}, (\mathbf{x}_1 - \mathbf{x}_2) \rangle = 2$. We define the normal vector to the separating hyperplane π as $\mathbf{w}/||\mathbf{w}||_2$. By projecting the difference between $\mathbf{x}_1$ and $\mathbf{x}_2$ onto the normal vector $\mathbf{w}/||\mathbf{w}||_2$, we obtain the distance between the two canonical hyperplanes as $(\mathbf{x}_1 - \mathbf{x}_2) \cdot \mathbf{w}/||\mathbf{w}||_2 = 2/||\mathbf{w}||_2$. We can finally define the margin $\gamma = 1/||\mathbf{w}||_2$ as half of the distance between the two canonical hyperplanes.
The described margin derivation is reported in Figure 13.

The objective is then to maximimize the margin γ as

$$\max_{\mathbf{w}} \frac{1}{||\mathbf{w}||_2} \rightarrow \min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||_2^2 \;\text{ s.t. }\; y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \;\; \forall i \tag{14}$$

Since this is a constrained optimization problem, it can be reduced as minimizing the *Lagrangian function*

$$L(\mathbf{w}, b; \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||_2^2 - \sum_{i=1}^{N} \alpha_i (y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1) \tag{15}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T$ is the vector of Lagrange coefficients. This is called the *primal* formulation of the SVM classifier, which has as Karush-Kuhn-Tucker conditions

$$\begin{aligned} \frac{\partial L}{\partial b} &= -\sum_{i=1}^{N} \alpha_i y_i = 0 \\ \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0 \\ y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) &\geq 1 \ \ \forall i, \\ \alpha_i (y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1) &= 0 \ \ \forall i, \\ \alpha_i &\geq 0 \ \ \forall i \end{aligned} \tag{16}$$

If we substitute $\mathbf{w}$ in Equation 15, we obtain the *dual* function

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{17}$$

The dual problem is then defined as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & J(\boldsymbol{\alpha}) \\ \text{s.t.} \quad & \alpha_i \geq 0, \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{18}$$

In the solution, from the KKT conditions we can see that either

- $\alpha_i > 0$ and the constraint is tight ($y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$) which means that $\mathbf{x}_i$ lies on one of the canonical hyperplanes, hence it is a support vector;
- $\alpha_i = 0$ and x_i is not a support vector, since the constraint is not tight ($y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 1$).

Thanks to the Fritz John Optimality Condition, which states that the solution to the problem of maximizing the margin is

$$\mathbf{w}^* = \sum_{i=I} \alpha_i \mathbf{x}_i \tag{19}$$

where $I = \{i : |\langle \mathbf{w}, \mathbf{x_i} \rangle| = 1\}$ is the set of indices of the support vectors, we can easily see that only the support vectors are relevant for the problem of maximizing the margin. This SVM formulation is called *hard margin* SVM, since points cannot lie within the margin band.

4.2.1 Soft margin SVM

Outliers and noisy data can badly influence the position of the separating hyperplane. These effects can be greatly reduced by introducing a *soft margin*, which means relaxing the constraint of the

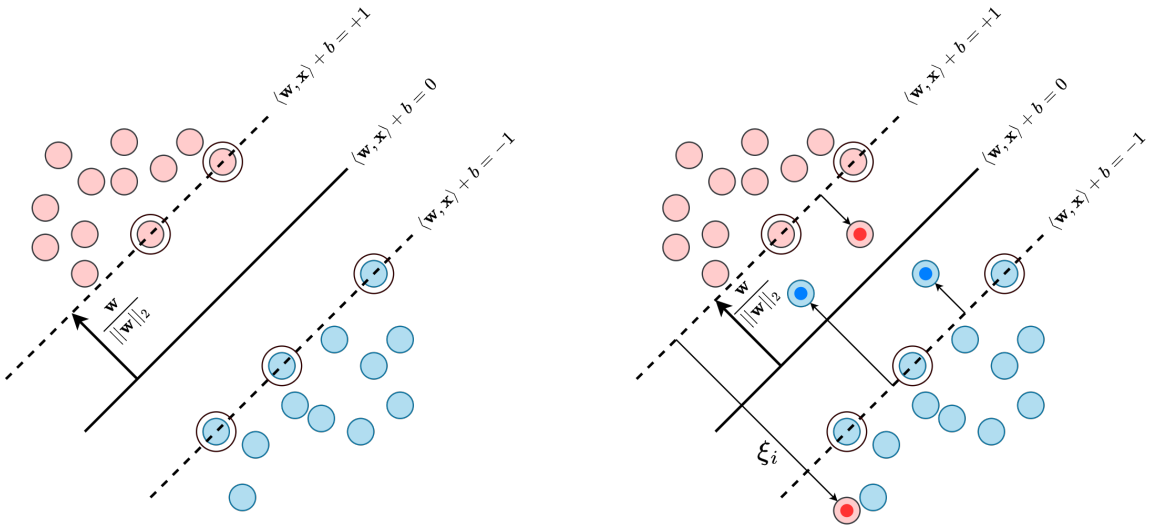


Figure 14: On the left: *hard margin* SVM. We do not allow data points to enter the margin.
On the left: *soft margin* SVM. We relax the constraint concerning points that could not enter the margin. In this setting, if a data point surpasses the margin by ξ_i, it will increase the objective function we aim to minimize by $C\xi_i$.

minimization Problem 14 by introducing some slack variables $\xi_i \geq 0$, obtaining the new problem

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad \forall i, \\ & \xi_i \geq 0 \qquad\qquad\qquad\quad \forall i \end{aligned} \tag{20}$$

We are now allowing points to enter the margin band of a quantity ξ_i, but at a cost $C\,\xi_i$. Note that we can reconduct Problem 20 to the *hard margin* setting if $C \to \infty$, which implies that $\xi_i = 0\ \forall i$. Instead, if $C \to 0$, we are including all points in the margin band.

4.2.2 Kernel SVM

Even if we allow noisy points to penetrate the margin, some datasets may not be linearly separable in their original space due to intrinsic geometry. Since data that is not linearly separable in the input space can always be linearly separable in a higher dimensionality feature space, we can map the original data into a new feature space through the feature map $\phi : \mathcal{X} \mapsto \mathcal{H}$. In this feature space we can fit the best linear separating hyperplane, which is actually a non-linear separator in the input space. From the dual formulation of SVM in Equation 17, we can see that it involves the inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Since the inner product can be considered as a linear kernel, we can just substitute it with a kernel

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \tag{21}$$

where $\mathcal{H}$ is a reproducing kernel Hilbert space, as described for Kernel PCA in Section 3.3.1.

Hence, we do not actually need to compute the mapping $\phi(\mathbf{x})\ \forall \mathbf{x}$, but we just need to compute the kernel matrix K where $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, which is much more efficient.

In Figure 15 a visual representation of the idea behind kernel SVM is reported.

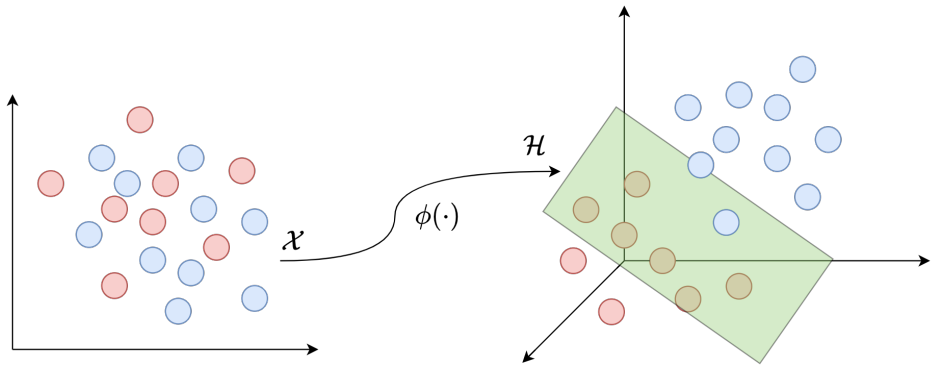


Figure 15: Visual representation of kernel SVM.

Finally, the dual formulation then becomes

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\ &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{22}$$

Some of the most used kernels are:

- the linear kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$,
- the polynomial kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\beta \langle \mathbf{x}_i, \mathbf{x}_j \rangle + r)^d$,
- the sigmoid kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \langle \mathbf{x}_i, \mathbf{x}_j \rangle + r)$,
- the Gaussian or RBF kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\beta ||\mathbf{x}_i - \mathbf{x}_j||^2)$,

where d, β and r are chosen hyperparameters. Actually, to be a kernel, the only necessary condition is for the Gram matrix G where $G_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ to be a symmetric and positive semi-definite matrix.

4.2.3 Kernel SVM for MTS: the Global Alignment Kernel

Global Alignment Kernel is introduced in [10] as part of a family of kernel functions that handles time series data of different lengths. It leverages on the DTW ideas shown in Section 4.1.1, in order to map a sequence onto another, i.e. align the two sequences. This kernel matrix K is defined as

$$K_{i,j} = \sum_{p \in \mathcal{P}} e^{-\beta \, C_p(X_i, X_j)} \tag{23}$$

where $X_i \in \mathbb{R}^{m_i, n}$ and $X_j \in \mathbb{R}^{m_j, n}$ are two MTS, $\mathcal{P}$ is the set of all possible warping paths p, as defined in Section 4.1.1, $C_p(X_i, X_j)$ is the cost associated to a warping path p as defined in Equation 11, and β is a smoothing factor, set as a hyperparameter. The similarity described by GAK considers the costs of the set of all possible paths, providing a richer statistic than the minimum cost of that set, which is how the DTW score, as defined in 12, is computed. In this sense, GAK considers two sequences similar not only if they have one single alignment with high score, but a wide set of efficient alignments.

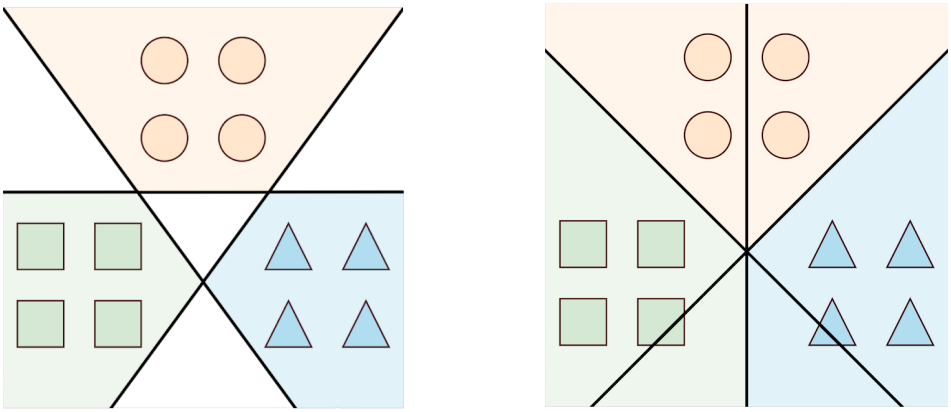


Figure 16: The results of SVM classification adopting the one-versus-rest scheme on the left and the one-versus-one scheme on the right. Note on the left that the white empty spaces signify that the classifiers for those regions do not come to an agreement.

4.2.4 Multiclass SVM

In order to adapt SVM to multiclass classification problems, two principal schemes are typically used:

- *one-versus-rest*, or *OVR*, where C separate SVMs, one for each class, are trained, leading to C different separating hyperplanes. The c-th classifier considers the label c as positive, while the other $C-1$ labels are considered as negative, making the setting binary again. For predicting new data, the label c picked is the one for which the distance from the c-th separating hyperplane is the greatest, while also being classified as positive by the c-th classifier.

- *one-versus-one*, or *OVO*, where $\binom{C}{2}$ SVMs are trained, one for each pair of classes, by considering each time only the points belonging to those classes. A node in a directed acyclic graph (DAG) is then associated to each of the binary classification problems. For predicting new data, $C-1$ consecutive binary decisions, starting from the root node, have to be taken in order to classify it.

A visual representation of the results of the different schemes is reported in Figure 16. When the number of classes C is large and the dataset is balanced, the OVR approach can perform poorly: if we consider a dataset of C classes, each containing Z examples, then each classifier is trained with $Z(C-1)$ negative examples and only Z positive examples, making it a very unbalanced binary classification problem. This problem worsens the larger the number of classes C gets.

4.3 Decision trees and Random Forests

Since Random Forests classifier is an *ensemble* of decision trees, we first need to introduce the concept of the latter.

Decision trees are functions which aim to divide the feature space into regions. The feature space is recursively partitioned starting from the whole, i.e. the root node. It is subdivided only into regions of which boundaries are orthogonal to the axes. A visual representation is provided in Figure 17. To split a node, a set of predictors of the input data is considered. The predictor used to split is the one which maximizes a *gain* metric. This approach is known as the greedy top-down approach. Gain metrics are used to characterise the quality of a splitting node t by measuring its impurity. A high value means that the split is not good, because the split does not divide data of different classes homogeneously (all classes are well balanced). Instead, a low value means the split is good, since data

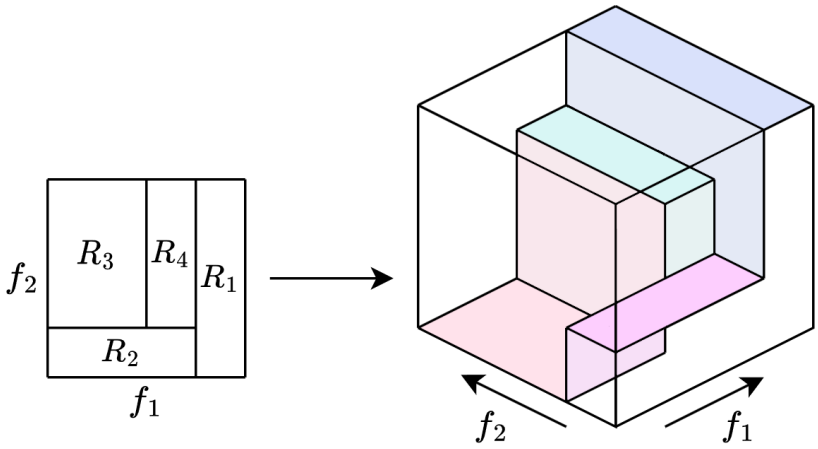


Figure 17: A visual representation of how the feature space (here represented by features $f1$ and $f2$) gets recursively linearly divided into d-dimensional boxes by the decision tree algorithm.

of different classes are divided homogeneously (all records belong to the same class). Two often used gain metrics to measure the impurity of a node t are the *Gini index*

$$\text{GINI}(t) = 1 - \sum_{c=1}^{C} [\hat{p}(l = c|t)]^2 \tag{24}$$

and the *entropy*

$$\text{Entropy}(t) = -\sum_{c=1}^{C} \hat{p}(l = c|t) \log_2 \hat{p}(l = c|t) \tag{25}$$

where $\hat{p}(l = c|t)$ is the relative frequency of the c-th class at node t. Both measures return the highest value when records are equally distributed among all classes (high impurity), and the lowest when all records belong to one class (low impurity). When classifying new unseen data, the decision tree is followed starting from the root node, until a leaf node is reached. Here, a majority vote is performed between the training examples in that leaf node.
Though decision trees are very simple, quick to compute and operate like a white-box, they are very weak learners that easily overfit, especially when their depth is too large.

Random Forests is a method that reduces overfitting by creating an ensemble of decorrelated decision trees through *bagging*.
The main idea behind bagging, or bootstrap aggregation, is to combine B predictions of B different learners, each trained on a different dataset, built by sampling with replacement N examples from the original dataset of size N. In the classification setting, the final label is obtained through majority voting between the B predictions. A visual representation of this process is reported in Figure 18.
Bagging dramatically improves learners with a high expected variance like decision trees, since by combining predictions from different learners the variance is reduced.
To obtain decorrelated trees, Random Forests performs a random selection of p predictors whenever a split has to be performed. The split is then allowed to use only one of those p predictors. Commonly, we choose $p = \sqrt{n}$, where n is the cardinality of the full set of predictors.

To summarize, Random Forests grows B different trees, each of them trained on a different bootstrapped dataset, performing each split on a different subset of the original predictors.
The computational complexity of training the Random Forests algorithm, assuming that the resulting trees are balanced, is $\mathcal{O}(BpN\log(N))$, where B is the number of trees in the forest, p is the number of predictors considered at each split, N is the number of examples in the dataset, and $log(N)$ is

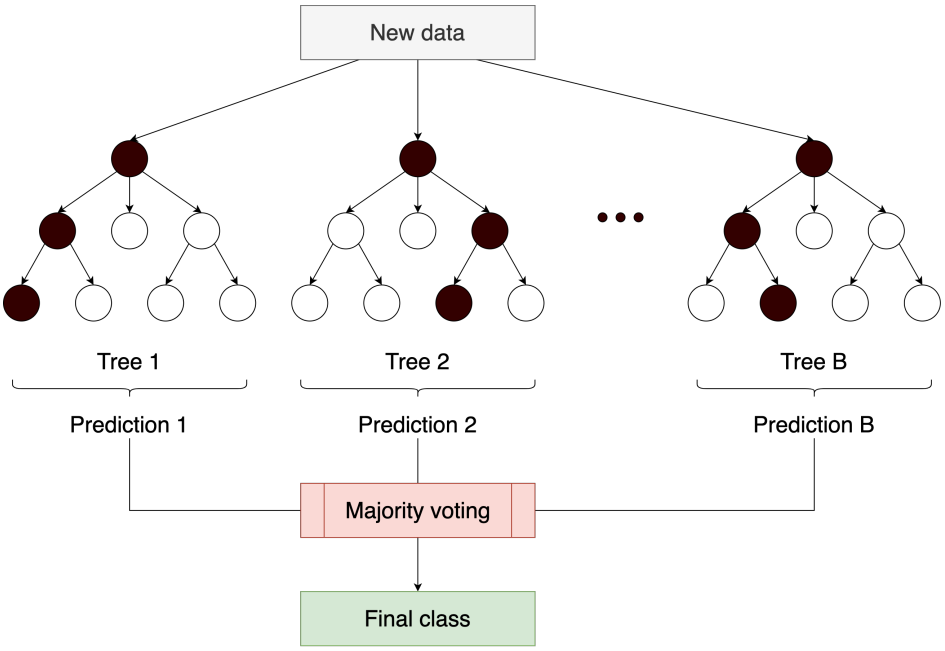
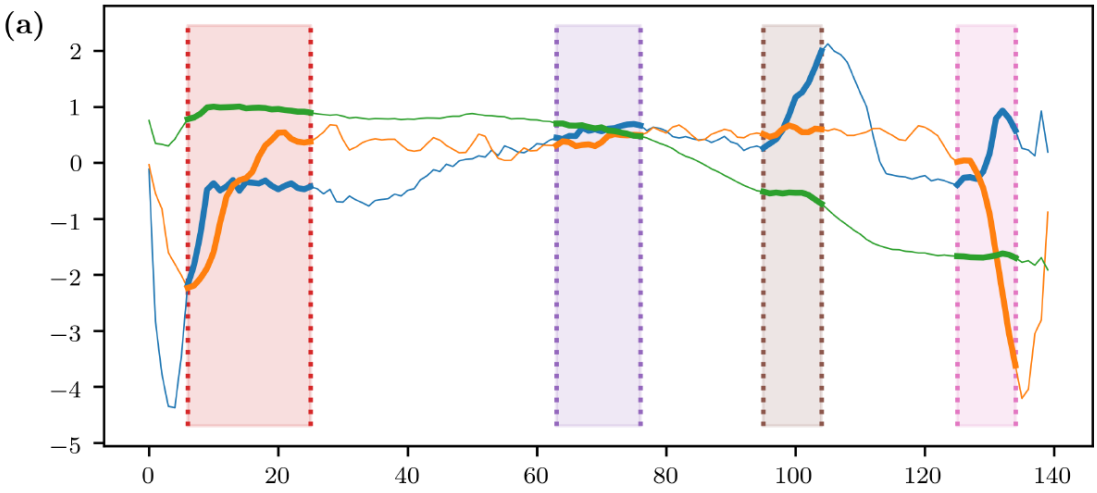


Figure 18: A visual representation of how a new observation is classified by a Random Forests classifier.

the depth of a balanced tree. Instead, for inference the complexity is reduced to $\mathcal{O}(\log(N))$. We can reduce both complexities to $\mathcal{O}(BpN\log(\alpha))$ and $\log(\alpha)$ by setting a `max depth` hyperparameter equal to $\alpha < N$, which limits the maximum depth of each tree. This can also be proven helpful for not making the trees overfit on the training data. Other hyperparameters that can be varied are the number of trees B, the minimum number of samples required at each leaf node and the minimum number of samples required to split a node.

4.3.1 Random Forests for MTS: Time Series Forest

In order to utilize a MTS dataset as input of tree-based methods, we introduce the Time Series Forest algorithm [11]. Firstly, given a a dataset of N MTS of dimension $m \times n$, $\sqrt{nm}$ random intervals of different lengths are generated, with minimum length r as a hyperparameter, and applied to each time series, obtaining from each of them $\sqrt{nm}$ subsequences of different lengths. Then, mean, standard deviation and slope are calculated for each interval, obtaining for each MTS a vectorial representation of dimensions $3\sqrt{nm}$. Finally, a decision tree is grown from this new dataset. This process is repeated $B =$ `n estimators` times, each time sampling different random intervals, in order to obtain an ensemble of learners. Then, just like Random Forests, predictions are combined through majority vote. A visual representation of this algorithm is reported in Figure 19.
Note that here the bootstrapping method is applied by sampling with replacement for each tree the random intervals, rather than the samples like in Random Forests.
The computational complexity of training TSF is $\mathcal{O}(B\sqrt{nm}N\log(N))$ where B is the number of estimators, $\sqrt{nm}$ is the number of random intervals and N is the number of MTS in the dataset. The computational complexity of inference instead is $\mathcal{O}(\log(N))$.



(b)

	Interval 1			Interval 2			Interval 3			Interval 4		
	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope
Time series 1	-0.61	0.504	0.052	0.58	0.086	0.02	1.004	0.572	0.197	0.189	0.504	0.156
Time series 2	-0.467	0.977	0.158	0.403	0.084	0.017	0.568	0.061	0.011	-1.204	1.305	-0.431
Time series 3	0.944	0.061	0.002	0.604	0.075	-0.018	-0.57	0.065	-0.017	-1.671	0.024	0.003

Figure 19: The main concept behind Time Series Forest. In (a), random intervals are generated and the random subsequences of each time series are extracted. In (b), three features are calculated for each subsequence: mean, standard deviation and slope. A decision tree is then fitted onto this new dataset. This process is repeated B times. Taken from [15].

Figure 20: 5-fold cross-validation.

5 Model Evaluation

In this section we introduce the practices we adopt in order to find the best hyperparameters for the models trained and to evaluate their predictions. Note that in order to perform our analysis, we divide the original dataset into a *training* dataset and a *test* dataset. The training dataset consists in $77.\bar{7}\%$ of the original dataset, while the remaining $22.\bar{2}\%$ is the test dataset. Since the original dataset is well-balanced, i.e. composed by 27 examples for each of the 95 classes, to ensure that the relative class frequencies are preserved, we perform a random *stratified* split, ending up with 21 and 6 examples for each class, respectively, in the training and test datasets.

5.1 K-fold cross-validation

Recall that the test error is the average error that results from using a learning model to predict the response on a new observation, while the training error is calculated by applying the same learning model to the observations used in its training. Then, the training error often heavily underestimates the test error. In order to more accurately estimate the latter, we divide the training, composed by N^{train} examples, into K equal-sized parts of dimensions $Q = N_{train}/K$, where $F_1, \ldots, F_K$ denotes the indices of the observations in the k-th part. For each k-th part, we fit the model to the other $K-1$ parts combined, and then obtain the predictions for the left out k-th part. We do this K times and then average the errors, to obtain a more accurate test error estimate. This process is called K-fold cross validation, and its error is computed as

$$CV_K = \frac{1}{K} \sum_{k=1}^{K} Err_k \tag{26}$$

where $Err_k = \frac{1}{Q} \sum_{i \in F_k} I(y_i \neq \hat{y}_i)$ and I is the classification error which counts the number of misclassified examples. K-fold cross-validation is often used for model selection (or hyperparameter tuning), and once the best combination of hyperparameters is chosen, i.e. the one that minimizes the

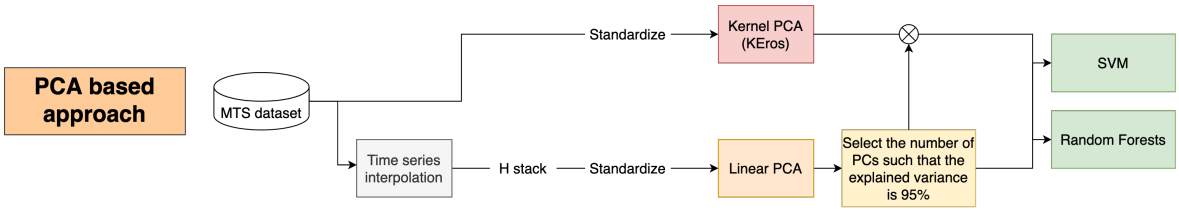


Figure 21: *PCA based* approach pipeline.

average of the validation errors, then the algorithm is retrained using this combination on the whole original training set. A visual representation of 5-fold cross-validation is reported in Figure 20.

5.2 Evaluation metric

In order to evaluate our models performances we adopt the *accuracy* metric. It counts the number of correctly classified observations by the model over the total number of observations, as

$$\text{Accuracy} = \frac{\#\text{ of correctly classified examples}}{\#\text{ total number of examples}} \tag{27}$$

6 Experiments

In this section we report the pipelines and results of our experiments, which can be divided in the two macro approaches introduced in Section 1.2 and that we now describe more thoroughly, along with their implementations. Note that, for every classification algorithm, we perform 5-fold cross validation in order to find the best combination of hyperparameters.

6.1 PCA based approach

We call the first approach we follow *"PCA based"* since we adopt as data preprocessing methods both PCA and K-PCA, to then compare their results. Figure 21 presents its pipeline.

We start by interpolating the data in order to obtain a fixed size representation of each MTS, in order to apply linear PCA. As target length we decide to interpolate each MTS to 60 points, which is the closest multiple of 10 to the mean length of the dataset (57.29), obtaining N MTS of size $(60, 22)$. To obtain one feature vector for each data, we stack their rows, obtaining a feature vector of size $60 \cdot 22 = 1320$. On this new dataset of size $(N, 1320)$, we apply Z-score standardization, in order to have columns values in a similar range, so that PCA, as a variance-maximization problem, can perform well. By performing linear PCA, we decide to keep the first 54 principal components, since they retain 95% of the variance of the original data, thus obtaining a dimensionality reduction from $(N, 1320) \to (N, 54)$. The resulting scree plot is reported in Figure 22. To perform linear PCA, we use `scikit-learn` [18] PCA implementation.

Another way to perform dimensionality reduction through PCA without the need to interpolate the time series is to use a kernel that calculates distances between MTS of different lengths, like KEros, which we introduced in Section 3.3.2. We implement the code concerning KEros-PCA ourselves from scratch, following [7]. Before applying K-PCA, we apply Z-score standardization to each MTS. In order to compare the two different methods, we decide to retain the same amount of principal components.

We choose as classifiers Random Forests and Support Vector Machine (see Sections 4.3, 4.2), considering respectively as hyperparameters, for RF

- `impurity metric`: *Gini* and *entropy*,
- `# of estimators`: $\{10, 25, 50, 100, 200\}$,

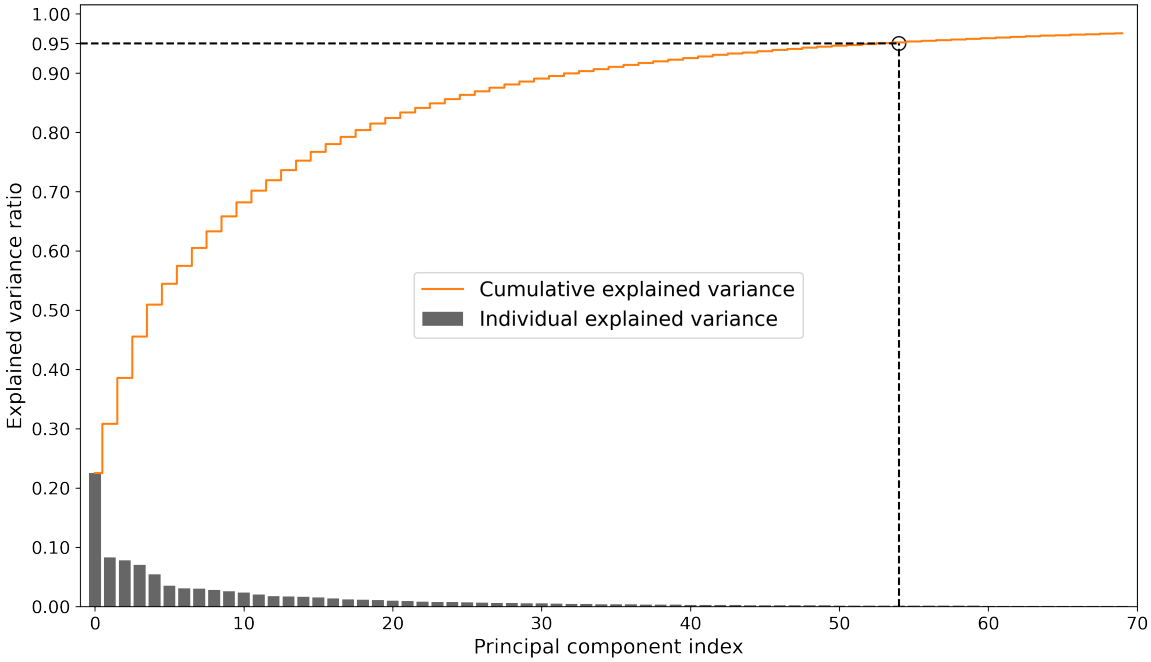


Figure 22: In black we plot the ratio of variance explained by each principal component, while in orange we plot the cumulative sum of the variance explained.

- `max depth`: $\{10, 20, 30, 50, 100\}$,
- `minimum samples per split`: $\{2, 6, 10\}$,
- `minimum samples per leaf`: $\{1, 3, 4\}$

and for SVM

- `kernel function`: *linear*, *polynomial*, *RBF*,
- `C`: $\{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 5 \cdot 10^{-2}, 10^{-2}, 1\}$,
- β: $\{10^{-3}, 10^{-2}, 1, 2, 5, 10\}$, only considered by the polynomial and RBF kernels,
- `degree` or d: $\{3, 6, 10, 15, 20, 25, 30\}$, only considered by the polynomial kernel,
- `r`: $\{0, 0.1, 0.5, 1, 5, 10\}$, only considered by the polynomial kernel.

The best combinations of hyperparameter for each dimensionality reduction method and each classification algorithm are reported in Table 3.

After finding the best hyperparameters combinations, we retrain the classifiers on the whole training set using the first 54 principal components. In Figure 25 we report the accuracy results on the test set. We colour in green the results concerning K-PCA with KEros while in red we report the ones concerning linear PCA. At first glance we can see how K-PCA with KEros outperforms linear PCA when we use an SVM classifier, while it performs poorly when using an RF classifier. Linear PCA instead consistently performs well using both classifiers. From Table 3 we can notice that linear PCA finds as best hyperparameter for SVM the linear kernel, suggesting that the data obtained after dimensionality reduction are linearly separable. On the other hand, also from Table 3 we can see that K-PCA with KEros finds as best hyperparameter for SVM the polynomial kernel with a very high

Algorithm	Hyperparameters best combination
Linear PCA: SVM	Linear kernel, C = 1
Linear PCA: RF	Gini index, max. depth = 30, min. samples per leaf = 1, min. samples per split = 2, 200 estimators
K-PCA: KEros SVM	Polynomial kernel, C = 0.005, $\beta = 5$, degree = 30, r = 1
K-PCA: KEros RF	Entropy, max. depth = 30, min. samples per leaf = 1, min. samples per split = 10, 200 estimators
KNN with DTW	Number of neighbors = 1, uniform weighting scheme
SVM with GAK	C = 0.0001, $\beta = 1$
Time Series Forest	200 estimators, min. interval length = 4

Table 3: Summary table reporting the best combination for each trained algorithm.

degree (`d` = 30), suggesting that the new data are not linearly separable. This might be why RF finds it harder to subdivide the features space for KEros with respect to linear PCA, leading to poorer performances. We use `scikit-learn` implementations of Random Forests and SVM classification algorithms.

In order to investigate how much the number of principal components influences the accuracy scores for both linear PCA and KEros K-PCA, we also plot the accuracy on the test set, varying each time the number of principal components, obtaining Figure 23. As we can see, KEros with SVM outperforms by a lot linear PCA with SVM when keeping a low number of principal components (≤ 20), while this decreases as the number of principal components grows.

6.2 Raw MTS approach

The second approach we follow aims at using classic Machine Learning classification algorithms with extensions that can deal with data in the MTS format. Figure 24 reports the pipeline we follow.
While Time Series Forest requires that time series are of the same length, K-NN with DTW and SVM with GAK are able to deal with data of different lengths when using these metrics, which are capable of comparing directly data in the time series format.

Following [15], we decide to start with K-NN, which is usually considered as a good baseline for time series classification problems. We use `tslearn` [16] implementation of K-NN, with DTW as the metric. As hyperparameters, we consider

- `k` : $\{1, 3, 5\}$ as the number of neighbors,
- `weighting scheme`: uniform and weighted.

The best performing combination can be seen in Table 3. In Figure 25 we report its test accuracy results, which reaches a solid 0.781.
The other algorithm which does not require any data transformation, albeit a standardization for better performances, is SVM with the GAK kernel. We use the SVM implementation from `tslearn`, using `gak` as kernel. As hyperparameters we consider

- $C : \{10^{-4}, 10^{-2}, 1\}$,
- $\beta : \{10^{-1}, 1, 10\}$

The best resulting hyperparameters can be found in Table 3, while the accuracy results of the trained model with said hyperparameters can be found in Figure 25. SVM with GAK outperforms the baseline defined above, but its results are far lower compared to SVM applied after PCA and K-PCA.

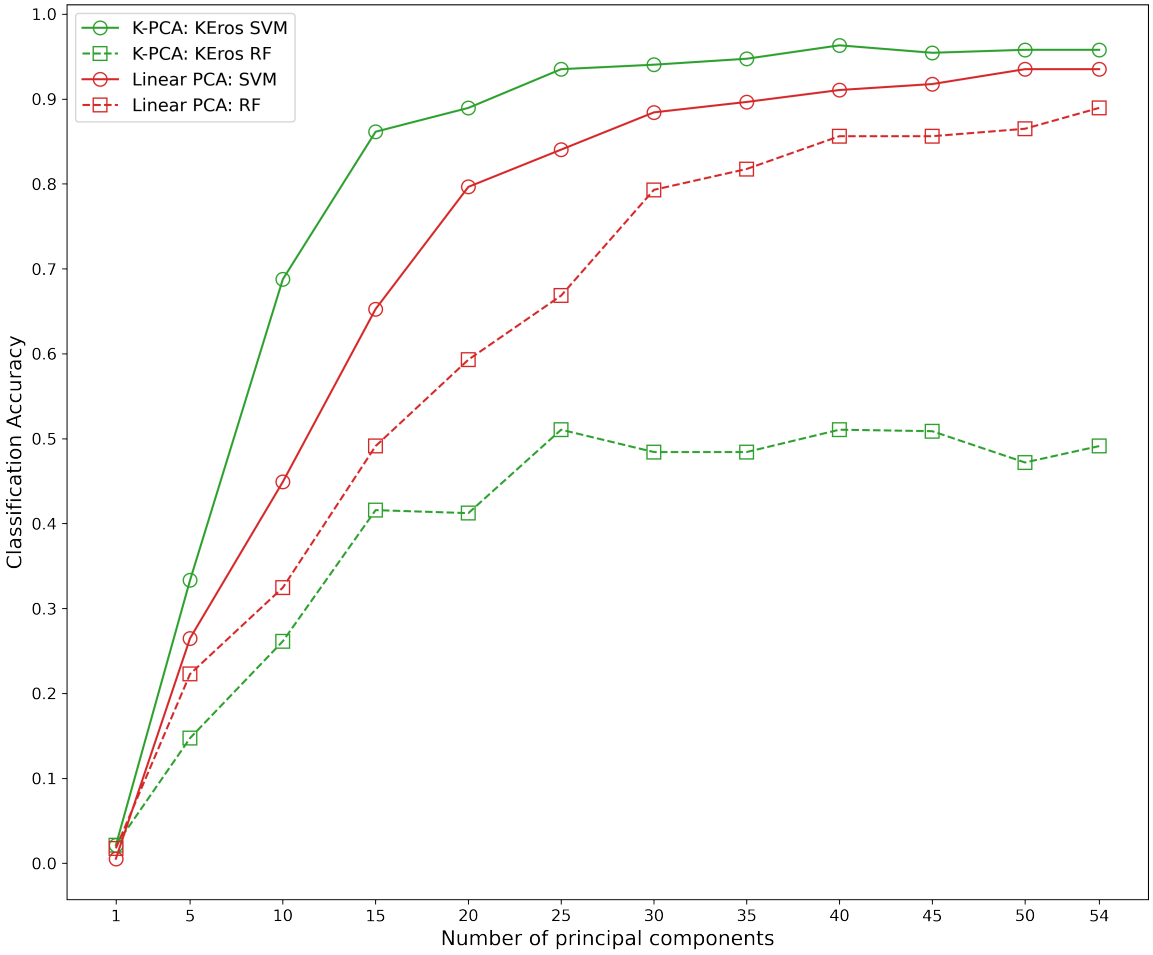


Figure 23: PCA accuracy results comparison by number of principal components kept. The linear PCA results are reported in red, while the results from KEros are reported in green. The results obtained with SVM are circles, while the ones obtained with RF are squares.

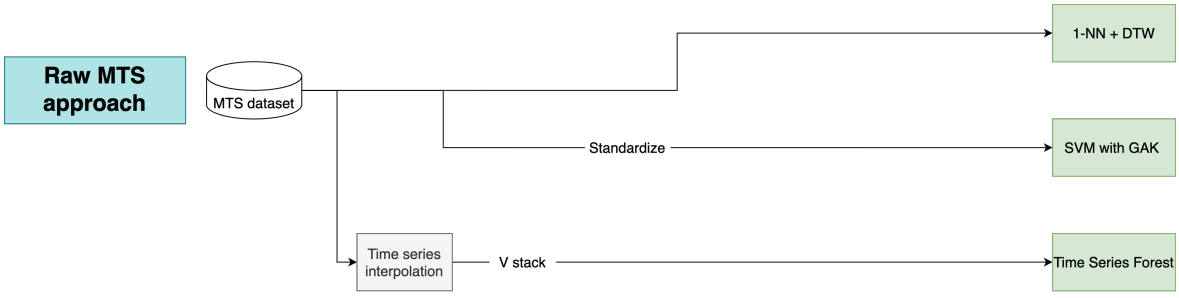


Figure 24: *Raw MTS* approach pipeline.

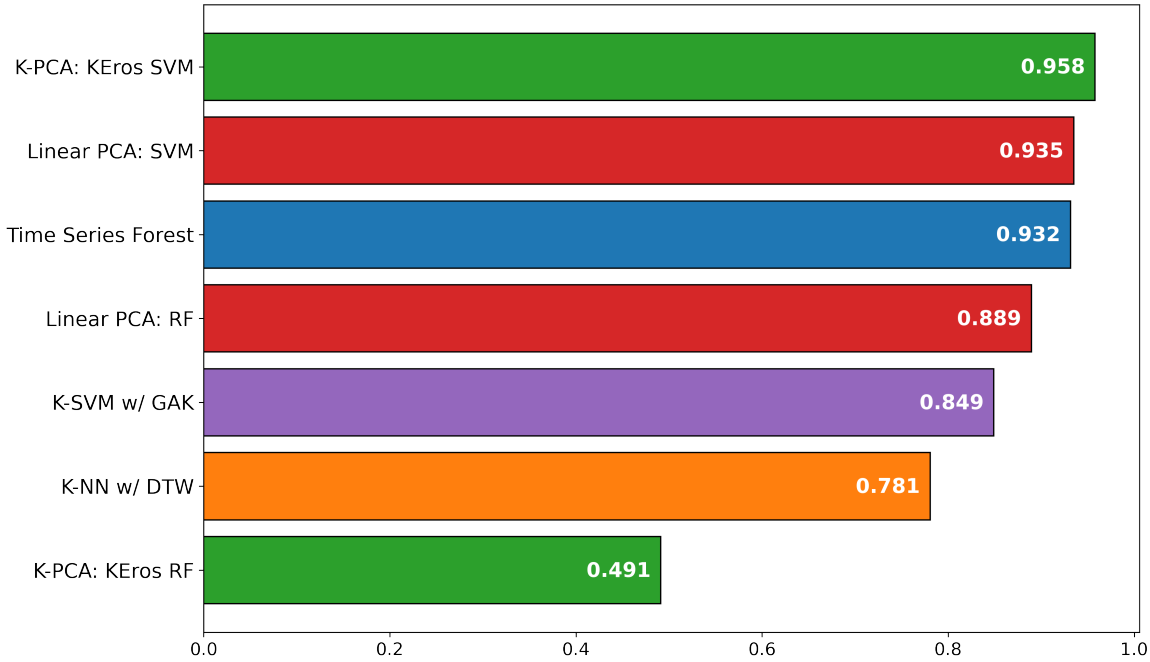


Figure 25: Final accuracy scores overall comparison.

Finally, the last algorithm we consider is Time Series Forest. Since this algorithm requires time series to be of the same length, we firstly interpolate them in the same way described for linear PCA. Then, in order to use the Time Series Forest implementation from `sktime` [17], we stack each time series columns into a single row, obtaining a feature vector of length $60 \cdot 22 = 1320$ for each observation. We consider as hyperparameters

- `# of estimators` : $\{10, 25, 50, 100, 200\}$,
- `min interval length` or r : $\{3, 4, 5, 6, 10\}$

The combination that performs better is reported in Table 3. The results on the test set are finally reported in Figure 25. Between all the algorithms considered in the *"Raw MTS"* approach, this last one by far performs best. It is also worth noting that it outperforms the Random Forests counterpart used in the *"PCA based"* approach, almost reaching the same accuracy as the SVM methods used in the first approach.

7 Conclusion

In this work, available at *https://github.com/notlosca/auslan-classification*, we explored how the usual multiclass classification setting is extended when data are not in the *standard* format (a single labelled feature vector for each observation), but instead in the MTS one. We investigated how dimensionality reduction techniques such as PCA and K-PCA need to be extended in order to tackle time series data, and also how classic ML classification algorithms can deal, in different ways, with this type of data. We can note from the results how the different dimensionality reduction approaches we followed lead to very high performances, suggesting that they are as powerful in the time series setting as they are in the standard one. It is also interesting to note how well KEros performs, even though it is not a proper kernel. We were also pleasantly surprised by the Time Series Forest algorithm, which reaches similar accuracy results as the methods mentioned above, but does not need dimensionality reduction on its input data. Instead, it requires data to be of the same length, making it harder to be adopted in a real-life scenario. This is why we consider KEros as the possible candidate method for a real-life scenario, albeit no new data are introduced after training the classifier, since both PCA and SVM are not incremental. A possible application could be translating in real time AUSLAN native speakers and other models could also be trained, using the same data framework, on other sign languages.

We only explored some extensions of classic ML methods for MTS classification on the surface. The results obtained might be further improved by performing a deeper hyperparameter search, trying different variations of algorithms, metrics and kernels we used, or also by applying different deep learning methods which are also proven to work well with MTS, like Recurrent Neural Networks, LSTMs and Transformers [19], [20], [21].

8 References

[1] Kadous, M. W., "Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series", PhD Thesis (draft), School of Computer Science and Engineering, University of New South Wales, 2002.

[2] Cortes, C., Vapnik, V.: Support-vector network. Mach. Learn. 20, 273–297 (1995)

[3] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[4] Karl Pearson F.R.S. (1901) LIII. On lines and planes of closest fit to systems of points in space, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2:11, 559-572, DOI: 10.1080/14786440109462720

[5] B. Schölkopf, A. Smola and K. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," in Neural Computation, vol. 10, no. 5, pp. 1299-1319, 1 July 1998, doi: 10.1162/089976698300017467.

[6] Yang, K. and Shahabi, C., 2004, November. A PCA-based similarity measure for multivariate time series. In Proceedings of the 2nd ACM international workshop on Multimedia databases (pp. 65-74).

[7] Yang, K. and Shahabi, C., 2005, November. A pca-based kernel for kernel pca on multivariate time series. In IEEE Intern. Conf. on Data Mining.

[8] Fix, E. and Hodges, J.L., 1989. Discriminatory analysis. Nonparametric discrimination: Consistency properties. International Statistical Review/Revue Internationale de Statistique, 57(3), pp.238-247.

[9] Sakoe, H. and Chiba, S., 1978. Dynamic programming algorithm optimization for spoken word recognition. IEEE transactions on acoustics, speech, and signal processing, 26(1), pp.43-49.

[10] Cuturi M (2011) Fast global alignment kernels. In: Pro- ceedings of the 28th International Conference on In- ternational Conference on Machine Learning, Omni- press, pp 929–936

[11] Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Information Sciences 239:142–153

[12] Aronszajn, N., 1950. Theory of reproducing kernels. Transactions of the American mathematical society, 68(3), pp.337-404.

[13] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. J. Mach. Learn. Res., 29-30, 2004.

[14] X. Nguyen, M. I. Jordan, and B. Sinopoli. A kernel-based learning approach to ad hoc sensor network localization. ACM Trans. Sen. Netw., 1(1):134–152, 2005.

[15] Johann Faouzi. Time Series Classification: A review of Algorithms and Implementations. Ketan Kotecha . Machine Learning (Emerging Trends and Applications), Proud Pen, In press, 978-1-8381524- 1-3. ffhal-03558165, p. 11.

[16] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar and E. Woods. "Tslearn, A Machine Learning Toolkit for Time Series Data", Journal of Machine Learning Research, vol. 21, num. 118, pp. 1-6, 2020.

[17] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, Franz Király (2019): "sktime: A Unified Interface for Machine Learning with Time Series"

[18] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[19] Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L. and Muller, P.A., 2019. Deep learning for time series classification: a review. Data mining and knowledge discovery, 33(4), pp.917-963.

[20] Pham, T.D. Time–frequency time–space LSTM for robust classification of physiological signals. Sci Rep 11, 6936 (2021). https://doi.org/10.1038/s41598-021-86432-7

[21] Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J. and Sun, L., 2022. Transformers in time series: A survey. arXiv preprint arXiv:2202.07125.