Australian Sign Language Dataset

A Multivariate Time Series Classification Problem

Scarciglia Lorenzo and Silvi Andrea

Politecnico di Torino

Mathematics in Machine Learning
September, 2022

Outline

Table of Contents

Introduction (1) - Task and Dataset

- Task: Multivariate Time Series Classification
  - Each observation is a labelled MTS $X_i \in \mathbb{R}^{m_i,n}$, where m_i is the length of a specific MTS, and n is the number of predictors.
- Dataset: AUSLAN
  - *Au*stralian *S*ign *Lan*guage Dataset.
  - This dataset is composed of 2565 labelled MTS of different lengths and 22 predictors each.
  - 95 signs (labels) are represented in this dataset. Each sign has 27 examples.
    - Signs also belong to 3 different families: (i) 59 are *one-handed*, (ii) 9 are *two-handed*, (iii) and 27 are *double-handed*.
  - All the signs were recorded by the same right-handed signer wearing motion capture gloves.

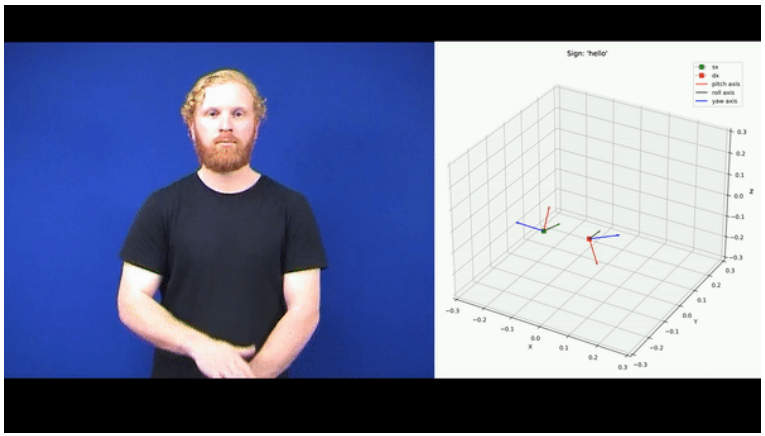
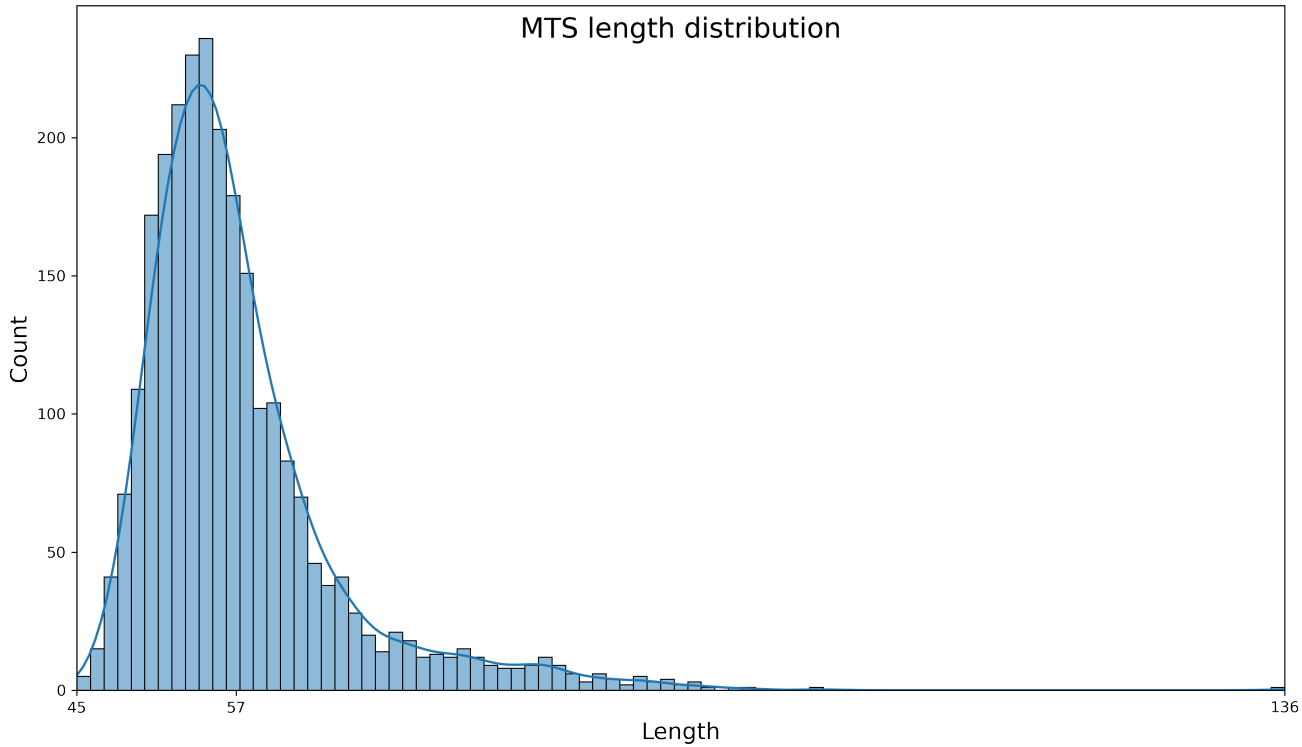


Table of Contents

Data Exploration (1) - Length Distribution

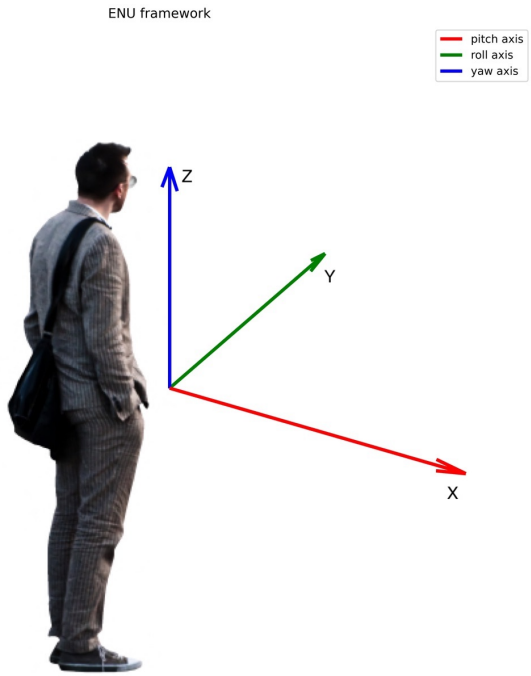
- MTS can be of different lengths. This happens because signers can be slower or faster at performing a sign.
- The average length of the dataset is 57.29 frames.



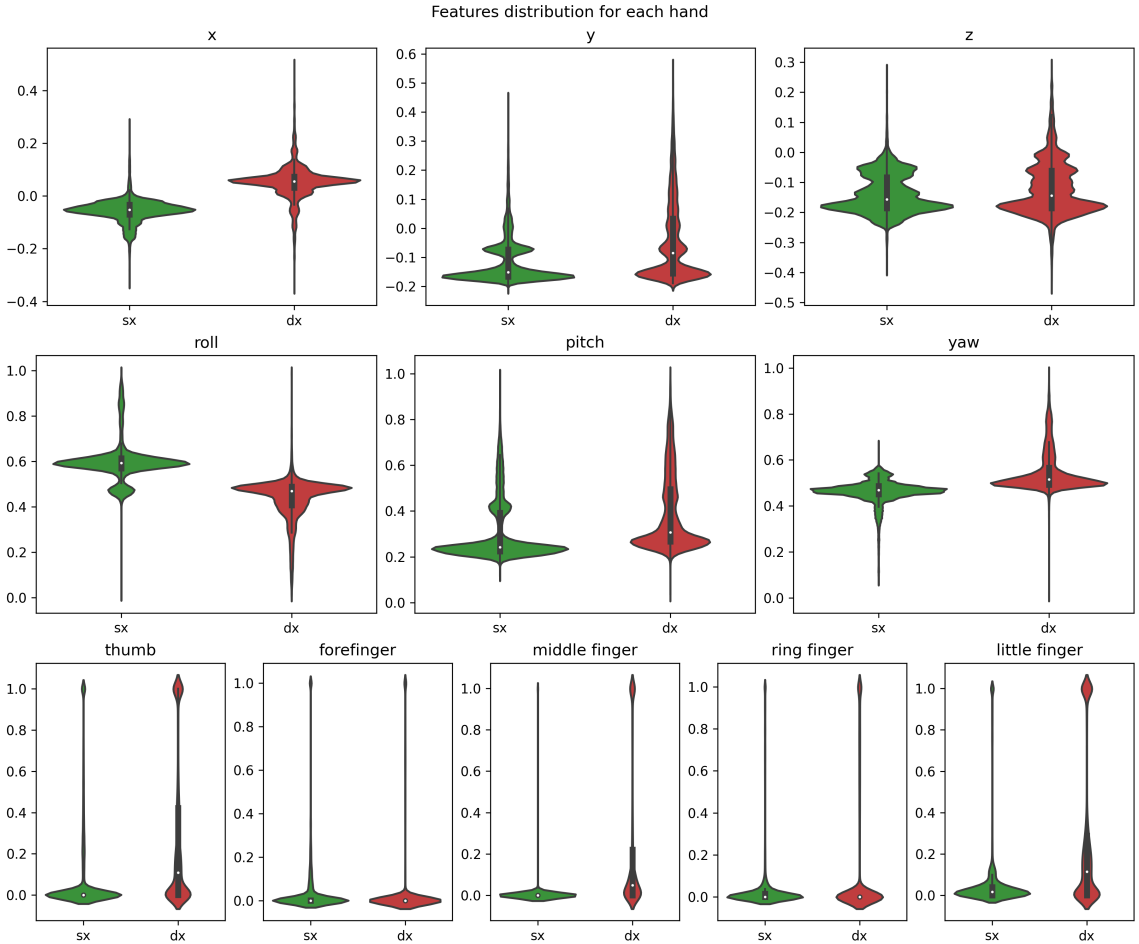
Data Exploration (2) - Features Description

There are 22 predictors, 11 for each hand.

- x, y, z describe in meters the hand distance from the origin in 3D space.
- *thumb*, *forefinger*, *middle finger*, *ring finger*, *little finger* describe the openness of each finger, where 0 means totally flat, while 1 means totally bent.
- *roll*, *pitch*, *yaw* describe the hand rotation in 3D space along the (x,y,z) axes, assuming the ENU coordinate system.



Data Exploration (3) - Features Distributions



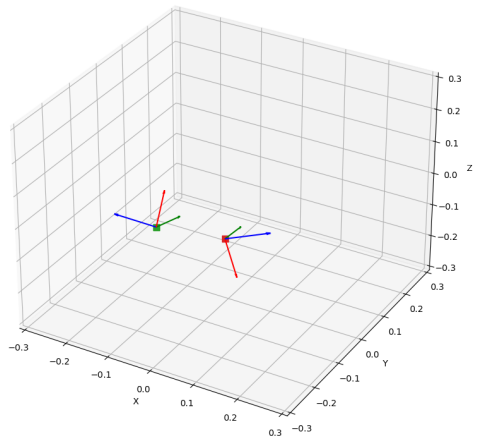
Features	Min	Max	Mean	σ
x left	-0.342	0.284	-0.055	0.043
x right	-0.359	0.506	0.051	0.063
y left	-0.21	0.452	-0.113	0.08
y right	-0.189	0.555	-0.044	0.139
z left	-0.397	0.28	-0.138	0.065
z right	-0.454	0.292	-0.121	0.089
roll left	0.003	0.998	0.598	0.089
roll right	0.001	0.998	0.437	0.09
pitch left	0.119	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

Data Exploration (4) - One-handed signs

A one-handed sign example: *Hello*

- Recall that the signer is right-handed
- *y*, *z*, *roll*, *pitch*, *yaw* mostly characterize the signs belonging to the one-handed family.

Features	Mean	σ
x left	-0.095	0.011
x right	0.073	0.071
y left	-0.115	**0.002**
y right	-0.056	**0.172**
z left	-0.043	0.001
z right	-0.019	0.03
roll left	0.489	**0.001**
roll right	0.39	**0.037**
pitch left	0.397	**0.003**
pitch right	0.449	**0.163**
yaw left	0.504	**0.001**
yaw right	0.522	**0.054**
thumb left	0.445	0.021
thumb right	0.282	0.131
forefinger left	0.0	0.0
forefinger right	0.0	0.0
middle finger left	0.0	0.0
middle finger right	0.0	0.0
ring finger left	0.0	0.0
ring finger right	0.0	0.0
little finger left	0.0	0.0
little finger right	0.095	0.022

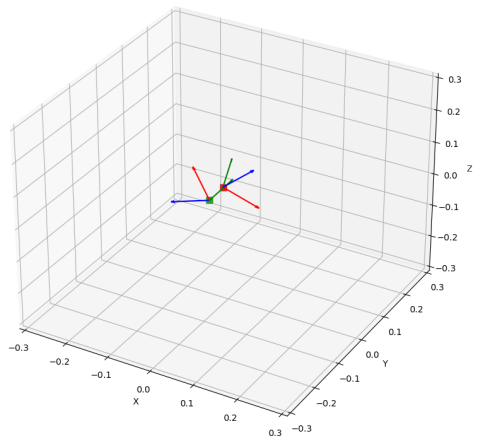


Data Exploration (4) - Two-handed signs

A two-handed sign example: *God*

- Since these signs are performed by both hands, the left hand features have a higher variance.
- These are the most complex signs (asymmetrical), thereby fingers features have the highest variance between the three families.

Features	Mean	σ
x left	-0.124	0.061
x right	0.078	0.048
y left	0.020	0.022
y right	0.068	0.138
z left	-0.045	0.021
z right	-0.037	0.010
roll left	0.472	0.082
roll right	0.389	0.079
pitch left	0.535	0.023
pitch right	0.513	0.112
yaw left	0.495	0.017
yaw right	0.552	0.036
thumb left	0.175	**0.364**
thumb right	0.196	**0.368**
forefinger left	0.198	**0.390**
forefinger right	0.052	**0.215**
middle finger left	0.179	**0.375**
middle finger right	0.187	**0.387**
ring finger left	0.204	**0.384**
ring finger right	0.173	**0.373**
little finger left	0.216	**0.383**
little finger right	0.332	**0.312**



Data Exploration (4) - Double-handed signs

A double-handed sign example: *Where*

- Since these signs are symmetric, the features variance for both hands are comparable.
- *x*, *roll* mean values are mirrored.

Features	Mean	σ
x left	-0.143	0.038
x right	0.111	0.021
y left	0.014	**0.027**
y right	-0.011	**0.023**
z left	0.007	0.021
z right	-0.033	0.032
roll left	0.521	**0.174**
roll right	0.469	**0.184**
pitch left	0.506	**0.023**
pitch right	0.486	**0.042**
yaw left	0.514	**0.017**
yaw right	0.523	**0.013**
thumb left	0.589	0.212
thumb right	0.769	0.353
forefinger left	0.081	0.051
forefinger right	0.000	0.000
middle finger left	0.000	0.000
middle finger right	0.000	0.000
ring finger left	0.001	0.002
ring finger right	0.000	0.000
little finger left	0.006	0.005
little finger right	0.109	0.043

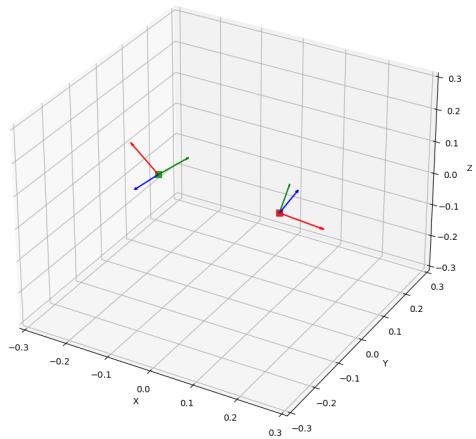


Table of Contents

Model Evaluation

- We randomly divide the dataset in a *stratified* manner into *training* $(0.\bar{7})$ and *test* set $(0.\bar{2})$.
- To obtain a more accurate test error estimate, we apply 5-fold stratified cross validation on the training set, in order to find the best combinations of hyperparameters for our models.
- Since we have a well-balanced dataset, we use the accuracy score to evaluate our models.

$$\text{Accuracy} = \frac{\#\text{ data correctly classified}}{\#\text{ data}}$$

Figure: 5-Fold Cross-Validation

Table of Contents

Theory (0a) - Z-score Standardization

- Good idea to have features of the same magnitude before applying dimensionality reduction techniques or classifiers like SVM.
- Z-score standardization is applied for each column $\mathbf{x}_i$ of the dataset by subtracting the mean μ_i and dividing by the standard deviation σ_i of that column:

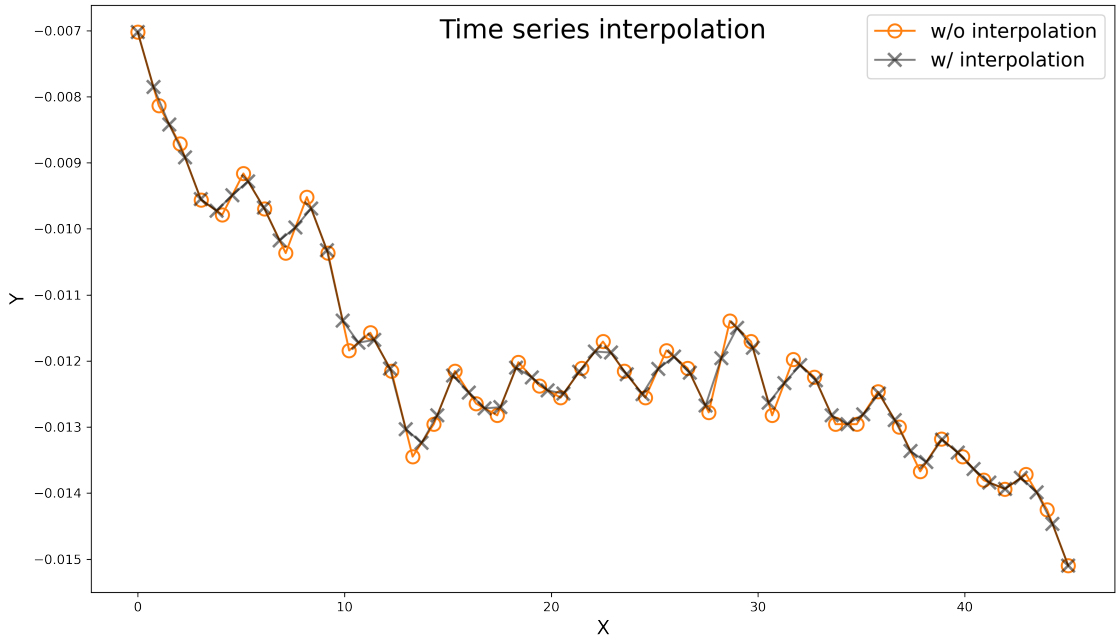
$$\mathbf{x}_{i,\text{scaled}} = \frac{\mathbf{x} - \mu_i}{\sigma_i}$$

- The vectors $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_n]$ and $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_n]$ are computed on the training set and used for standardizing the test set too.
- When data are in the MTS format, we compute $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the whole stacked training set.
  - Then Z-Score is applied to each MTS column-wise.

Theory (0b) - Linear Interpolation

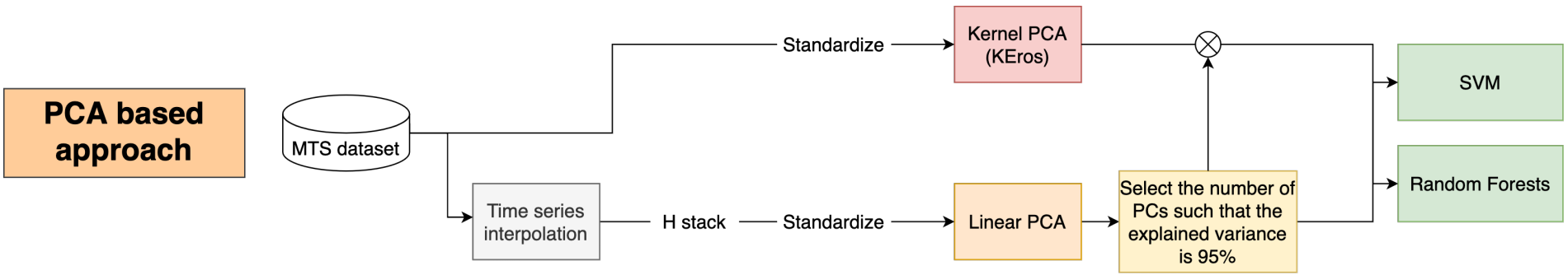
- To obtain MTS of the same length, we perform linear interpolation.
- For a generic MTS $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_n], \mathbf{y}_i \in \mathbb{R}^m$ with temporal indices $\mathbf{x} = [x_1, \ldots, x_m]$, we interpolate each predictor values separately.
- We obtain new indices $\mathbf{x}_{\text{interp}} = [\hat{x}_1, \ldots, \hat{x}_{60}]$ and new values $[\hat{y}_{1,j}, \ldots, \hat{y}_{60,j}]^T$ for the j-th predictor.

$$\frac{\hat{y}_{i,j} - y_{a,j}}{\hat{x}_i - x_a} = \frac{y_{b,j} - y_{a,j}}{x_b - x_a}, \forall i \text{ s.t. } \hat{x}_i \in [x_a, x_b]$$



Experiments (1) - *PCA based* approach

- Linear PCA
  - Time Series linear interpolation + stack
- Kernel PCA
  - with KEros kernel

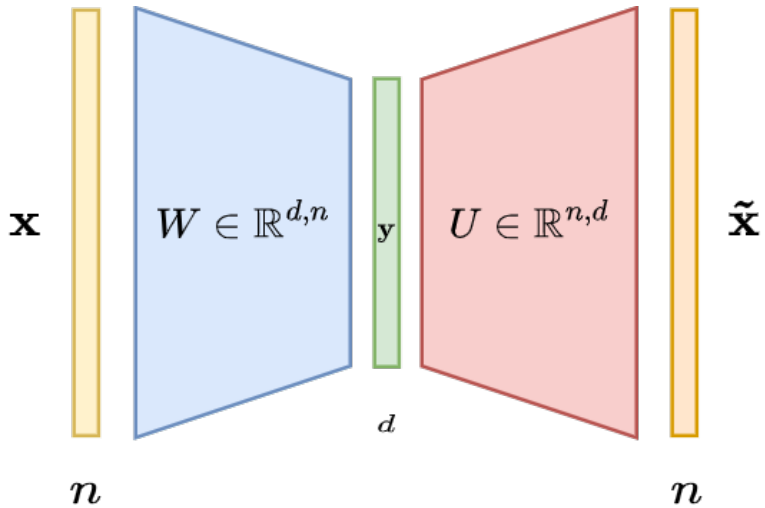


Theory (1a) - PCA

- PCA is a linear dimensionality reduction technique which projects data in the subspace spanned by the principal components.

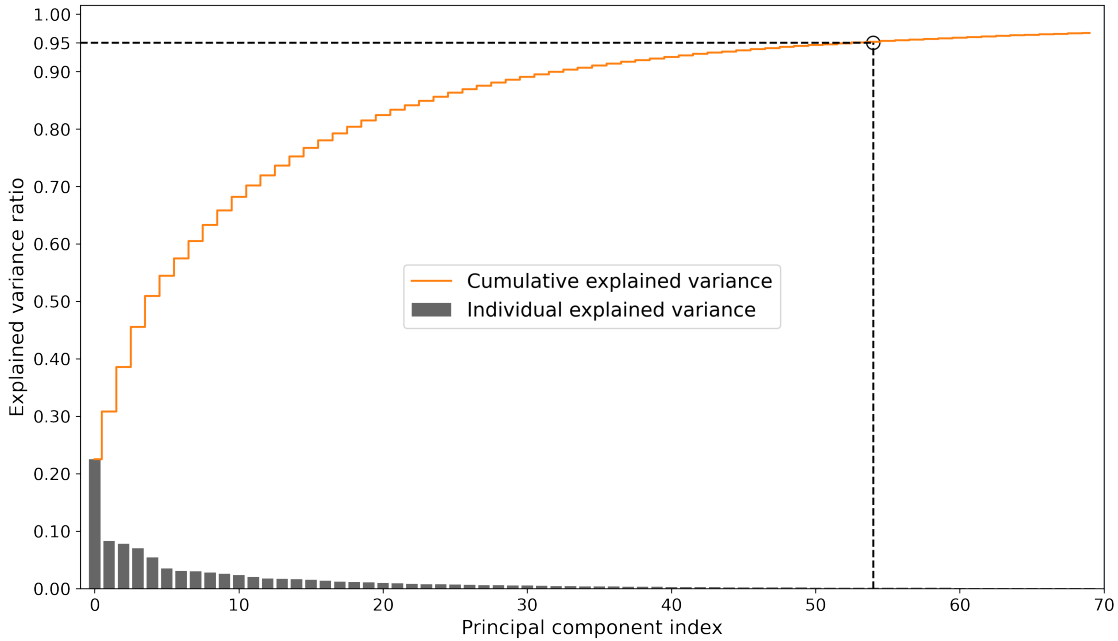
$$\mathbf{x} \mapsto \mathbf{y} = W\mathbf{x},\ \mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^d,\ d < n$$

- The aim is to minimize the reconstruction error $\underset{W,U}{\text{argmin}} \sum_{i=1}^{m} ||\mathbf{x}_i - UW\mathbf{x}_i||_2^2$
- The solution 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 $A = X^T X = \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T$.
- Once the principal components of the training set are found, we project the test set onto the space spanned by those principal components.



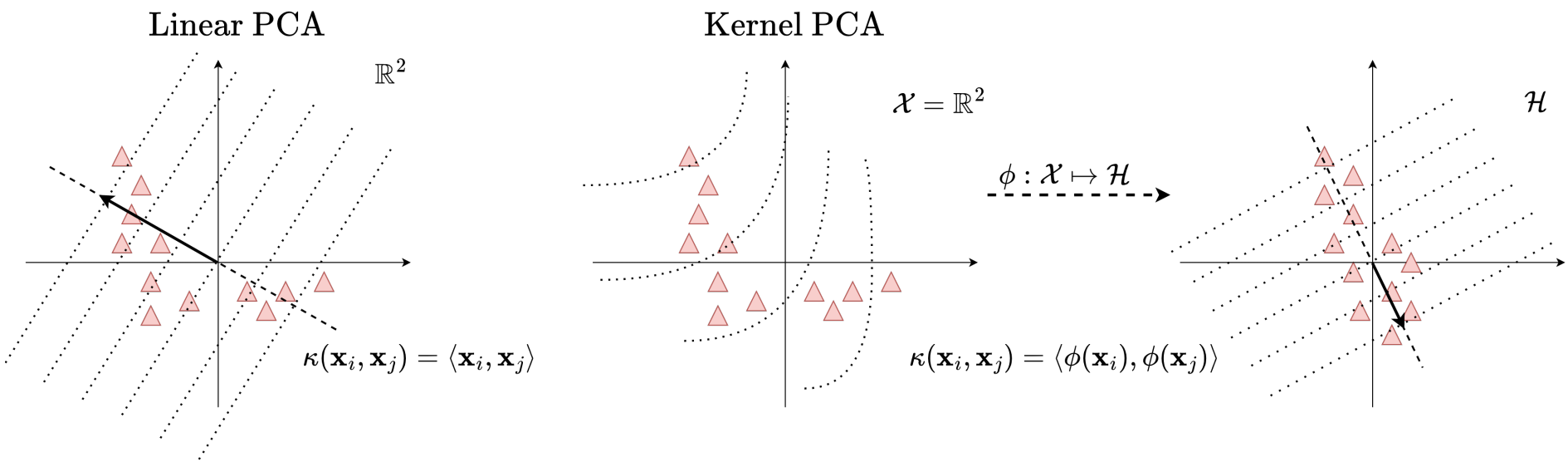
Theory and Experiments (1b) - PCA Variance Explained

- Each principal component $\mathbf{pc}_i$ explains a fraction of the variance of the original data.
  - The associated eigenvalue λ_i corresponds to the variance of the original data along the direction of $\mathbf{pc}_i$.
- We can plot the fraction of variance explained for each $\mathbf{pc}_i$ and the cumulative variance explained.
- Since we want to retain 95% of the variance of the original data, we keep the first 54 principal components.



Theory (1c) - Kernel PCA

- Kernel PCA extends PCA by performing a non linear form of PCA through the use of *kernel methods*.
- By substituting matrix $A = X^T X$ with $B = XX^T$ where $B_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ it can be proven that the PCA solution can be obtained by solving the eigenvalue problem of B instead of A.
- This enables us to *kernelize* PCA by substituting the inner product with a kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$, where $\mathcal{H}$ is a RKHS with relative kernel $\kappa(\cdot, \cdot)$.



Theory (1d) - KEros (1)

- We use as kernel function a similarity measure that works between MTS of different lengths called *Eros*.
- For each MTS X_i, the algorithm computes the right eigenvectors V_i^T and relative eigenvalues of its covariance matrix through SVD.

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;
    $[U_i, D_i, V_i^T] \leftarrow$ **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 $[\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N]$, $[V_1^T, \ldots, V_N^T]$

Theory (1d) - KEros (2)

- A weight vector $\mathbf{w}$ is computed by aggregating the eigenvalues of all the MTS in the dataset.

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 $\mathbf{w} = [w_i, \ldots, w_n]^T$

Theory (1d) - KEros (3)

- Given two MTS A and B and their right eigenvectors $[\mathbf{a}_1, \ldots, \mathbf{a}_n]$ and $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$, the Eros distance is then computed as

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

- 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.

Algorithm 3 Computing the kernel matrix 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
  return K^{Eros}

Theory (1d) - KEros (4)

- It can be proven that, as long as K^{Eros} is a symmetric positive semidefinite matrix, it can be used as Kernel matrix even though Eros is not a proper distance metric and cannot be represented as a dot product.
- The Kernel matrix must be centered in feature space prior to applying PCA.

Algorithm 4 Ensure that K^{Eros} is PSD and center in feature space.

Require: K^{Eros} from Algorithm 3.

if K^{Eros} is not PSD **then**

$K^{Eros} \leftarrow K^{Eros} + \delta I$

end if

$\bar{K}^{Eros} \leftarrow K^{Eros} - \mathbf{1}_N K^{Eros} - K^{Eros}\mathbf{1}_N + \mathbf{1}_N K^{Eros}\mathbf{1}_N$

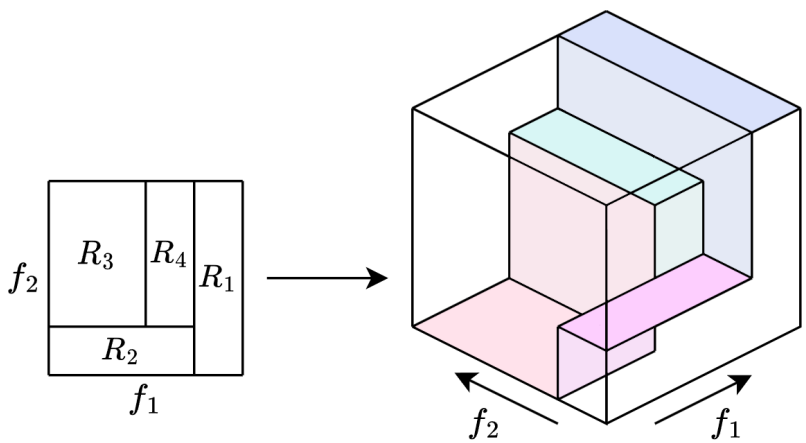
return $\bar{K}^{Eros}$

Theory (1e) - Decision Trees

- Decision trees are functions which aim to divide the feature space into regions.
- *Top-down approach*: a tree recursively partitions the features space into subregions with axes orthogonal to the boundaries.
- *Greedy*: each split is done in a way that minimizes an impurity metric.
- Two often used metrics to measure the impurity of a node t are

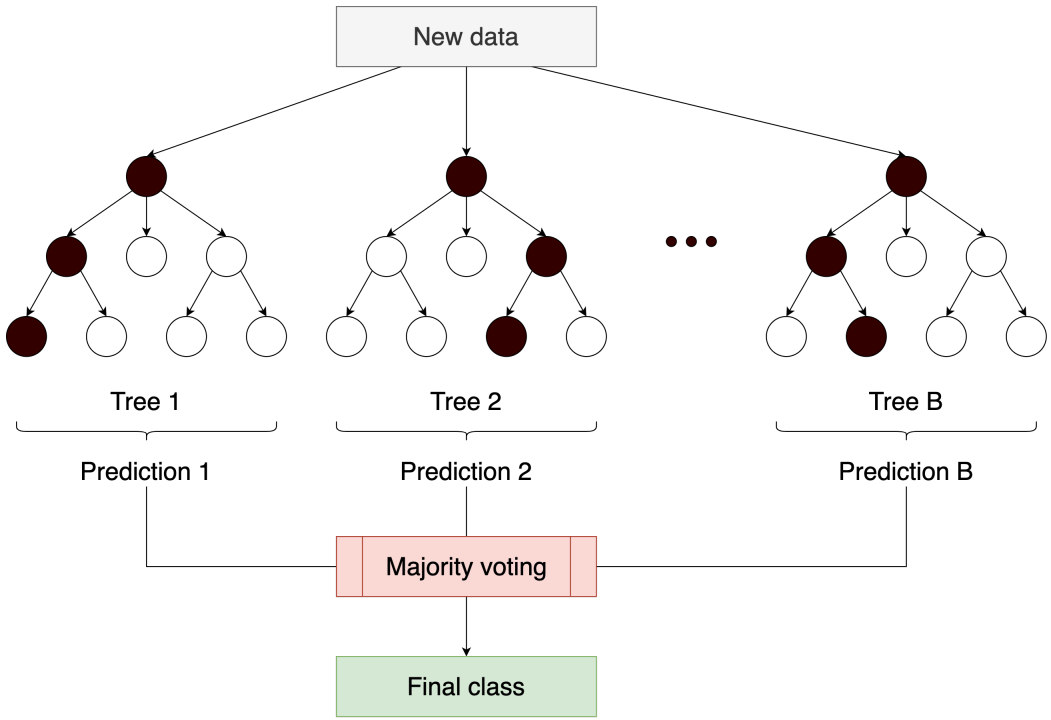
$$\text{GINI}(t) = 1 - \sum_{c=1}^{C} [\hat{p}(l = c|t)]^2$$

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



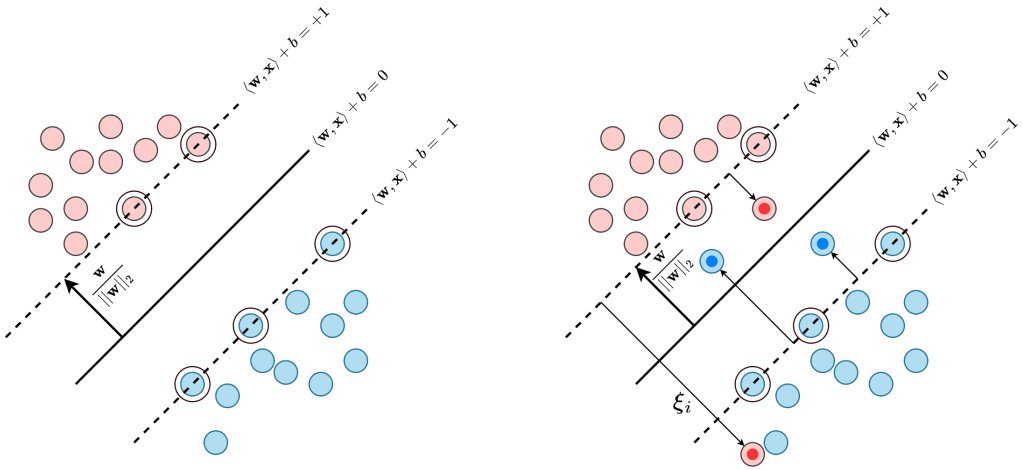
Theory (1f) - Random Forests

- They are are an ensemble of decision trees.
  - *Bagging*: Each tree is trained on a different bootstrapped dataset of the same size of the original dataset to reduce overfitting.
  - *Feature bagging*: each split is performed on a different subset of the original predictors to obtain decorrelated trees.
- The final prediction for unseen data is taken via majority voting between all the trees.



Theory (1g) - SVM

- Support Vector Machine is a binary classifier that aims at dividing points belonging to different classes by finding the best oriented separating hyperplane $\pi : (\mathbf{w}, b)$.
- *Hard-margin SVM*: The objective is to maximize the margin $\gamma = \frac{1}{||\mathbf{w}||_2}$:
  - $\max_{\mathbf{w}} \frac{1}{||\mathbf{w}||_2} \rightarrow \min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||_2^2$ s.t. $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \ \ \forall i$
  - In this setting, we do not allow data points to enter the margin band.
- *Soft-margin SVM*: By introducing *slack variables* $\xi_i \geq 0$ we can relax the constraint of the hard-margin problem.
  - This makes SVM more robust to noisy data and outliers.
  - A point $\mathbf{x}_i$ is now allowed to enter the margin band at the cost of $C\xi_i$, where C is a fixed cost.

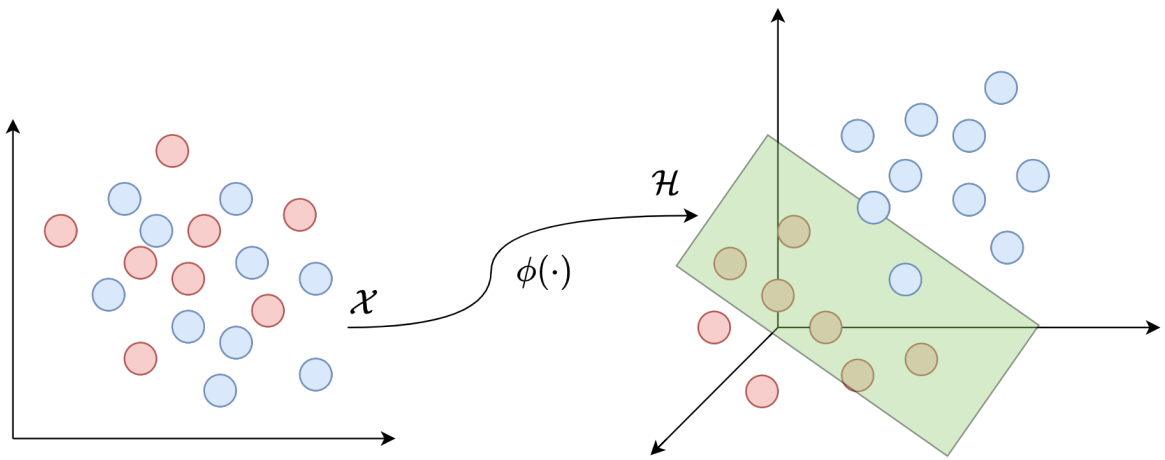


Theory (1h) - Kernel SVM

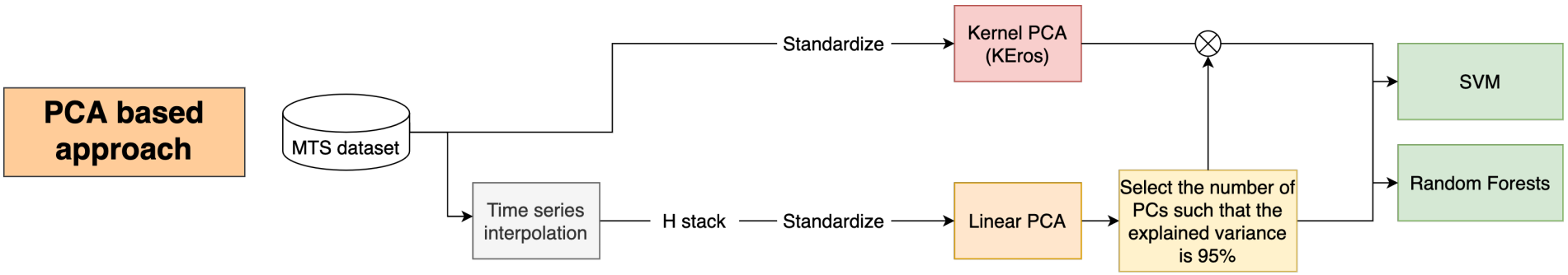
- Soft-margin SVM may not be enough when the intrinsic geometry of the data makes them not linearly separable.
- The hard margin problem is a constrained optimization problem, which can be solved using the Lagrangian function with relative multipliers α_i, obtaining the dual formulation:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \text{ s.t. } \alpha_i \geq 0\,, \sum_{i=1}^{N} \alpha_i y_i = 0$$

- We can substitute the inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with any kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$
  - We just need to compute the Kernel matrix K where $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$.



Experiments (1i) - *PCA based* pipeline

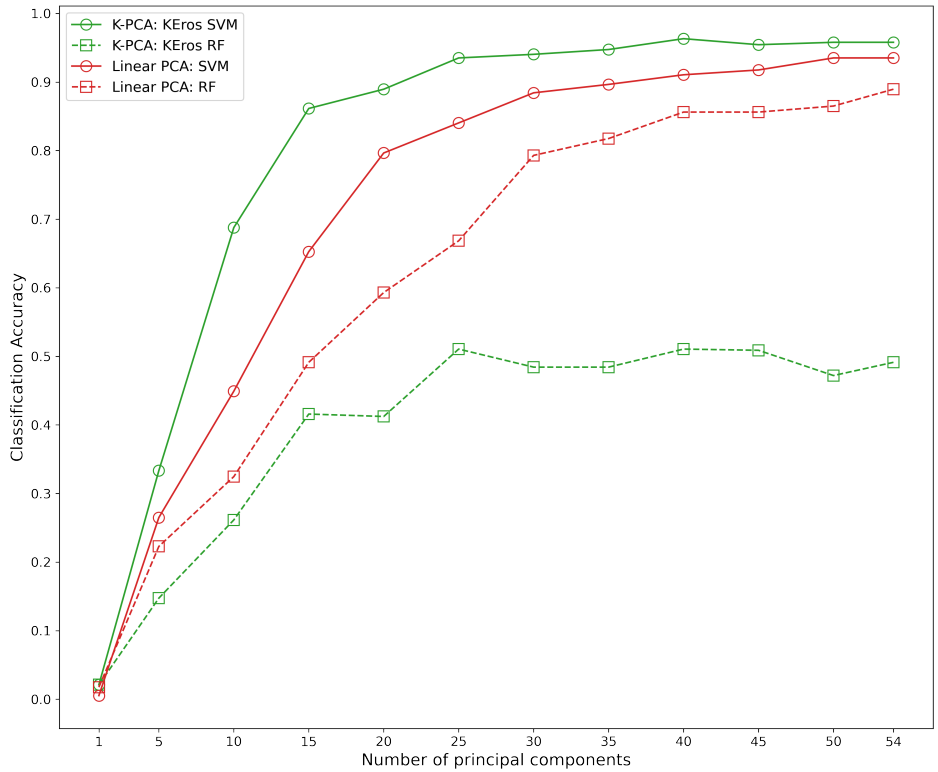


Experiments (1j) - Results

- Best hyperparameters from the 5-fold cross-validation and accuracy results on the test dataset:
  - Kernel PCA with KEros + SVM is the best performing methodology.
  - Instead, linear PCA with interpolation leads to more consistent results with both SVM and Random Forests.

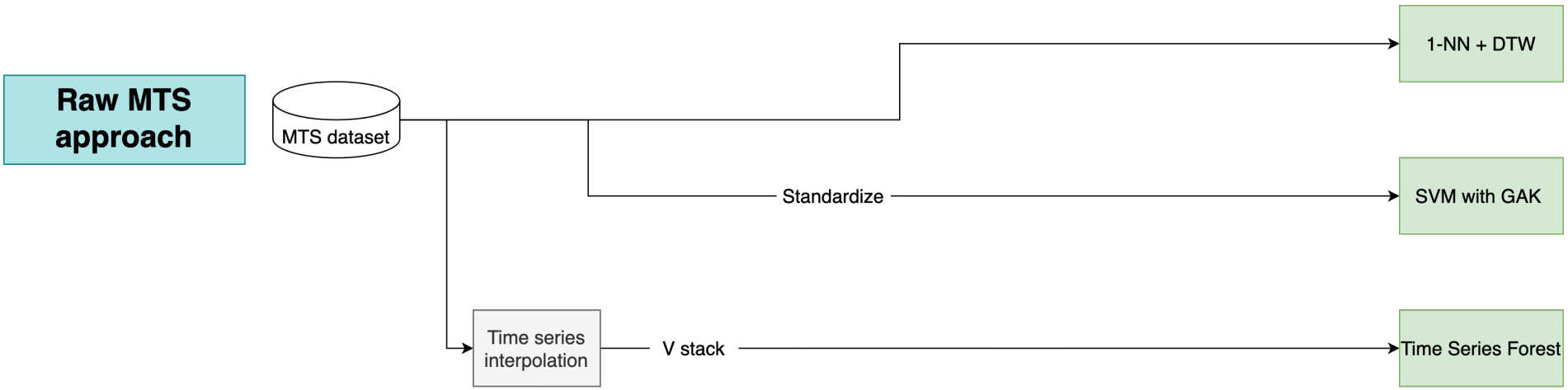
Algorithm	Hyperparameters best combination	Acc.
Linear PCA: SVM	Linear kernel, C = 1	0.935
Linear PCA: RF	Gini index, max. depth = 30, min. samples per leaf = 1, min. samples per split = 2, 200 estimators	0.889
K-PCA: KEros SVM	Polynomial kernel, C = 0.005, $\beta = 5$, degree = 30, $r = 1$	**0.958**
K-PCA: KEros RF	Entropy, max. depth = 30, min. samples per leaf = 1, min. samples per split = 10, 200 estimators	0.491

Table: Best hyperparameters combination for each method and final accuracy on the test dataset.



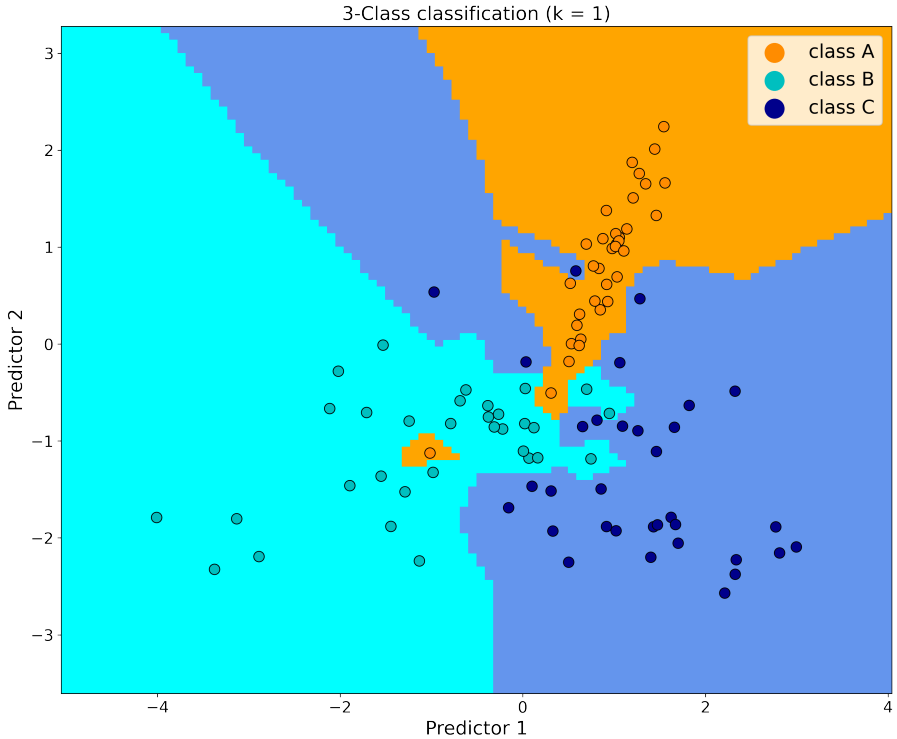
Experiments (2) - *Raw MTS* approach

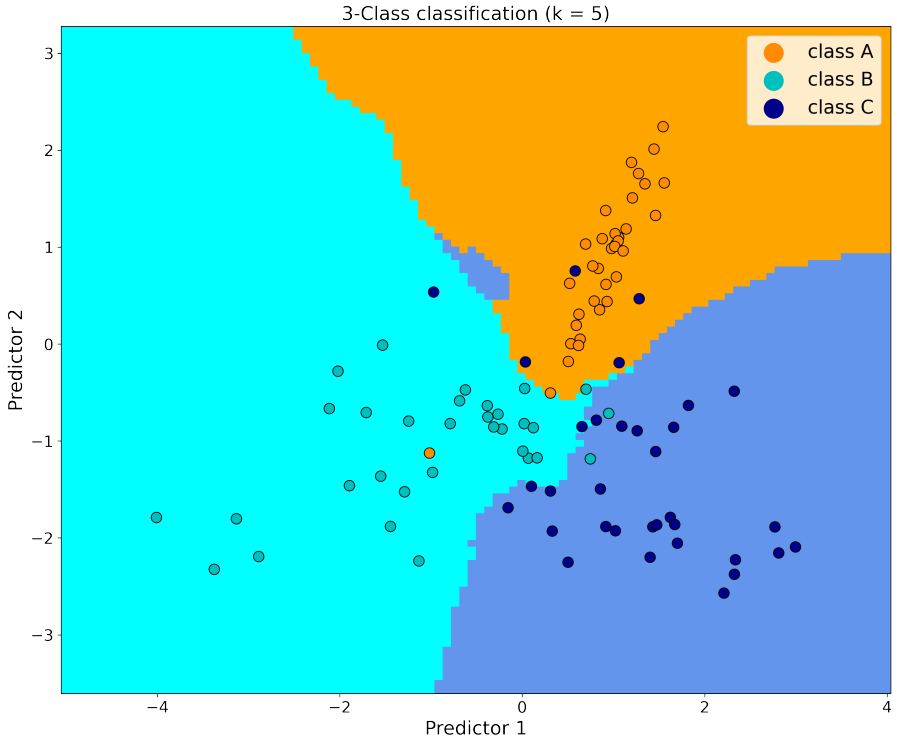
- K-Nearest Neighbors
  - Dynamic Time Warping distance metric
- Kernel SVM
  - Global Alignment Kernel
- Time Series Forest
  - A Random Forests extension for MTS
  - Time Series linear interpolation + stack



Theory (2a) - K-Nearest Neighbors

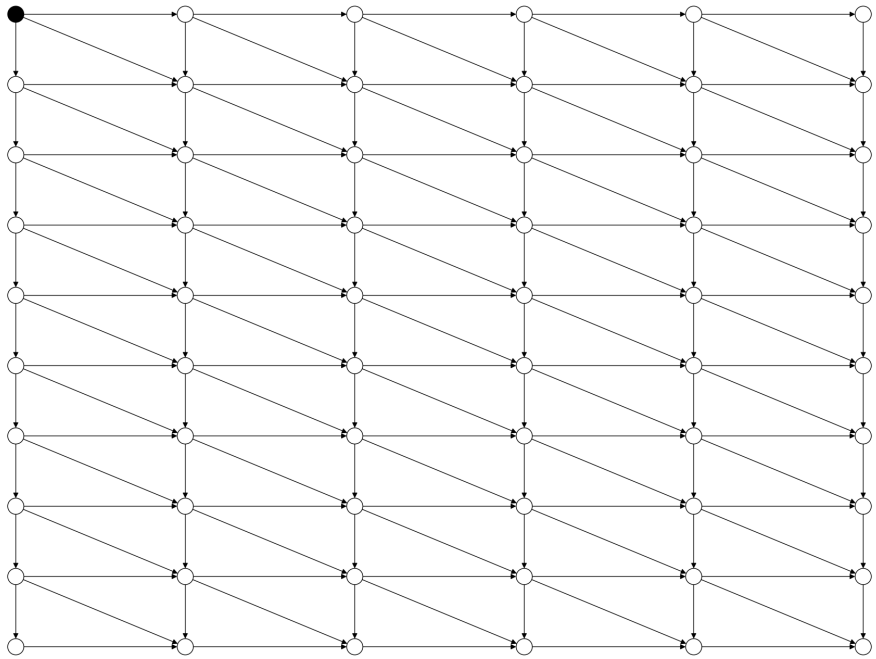
- K-NN is a classification algorithm which classifies new points by assigning the labels of its *k closest* neighbors.
- A distance metric is used to compute distances between points
  - Usually the Euclidean distance is adopted $||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\sum_{k=1}^{n}(x_{i,k} - x_{j,k})^2}$
- When $k > 1$, a majority voting scheme is adopted between the k neighbors.
  - Each neighbor vote can be weighted by the distance between itself and the new point.





Theory (2b) - Dynamic Time Warping (1)

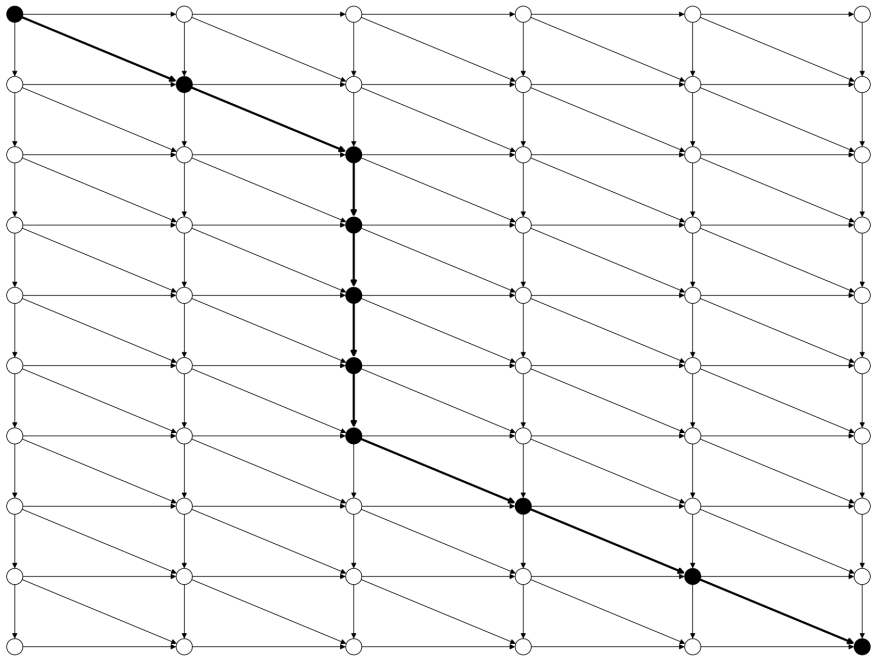
- DTW is a metric suited for comparing MTS of different lengths.
  - It aims at finding the *optimal matching* between the two sequences.
- Consider two MTS X_i and X_j of lengths m_i and m_j
  - A cost matrix $C \in \mathbb{R}^{m_i, m_j}$ is obtained by computing the squared Euclidean distance between each pair of points of the two time series.
  - We define a *warping path* as the as the sequence $p = (p_1, \ldots, p_S)$ satisfying some constraints.



Theory (2c) - Dynamic Time Warping (2)

- The cost $C_p(X_i, X_j)$ associated with a warping path p is defined as the sum of the costs of the states visited by the path.
- The DTW score is defined as the *minimum* cost among all the warping paths $p \in \mathcal{P}$:

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



Theory (2d) - SVM with *Global Alignment Kernel*

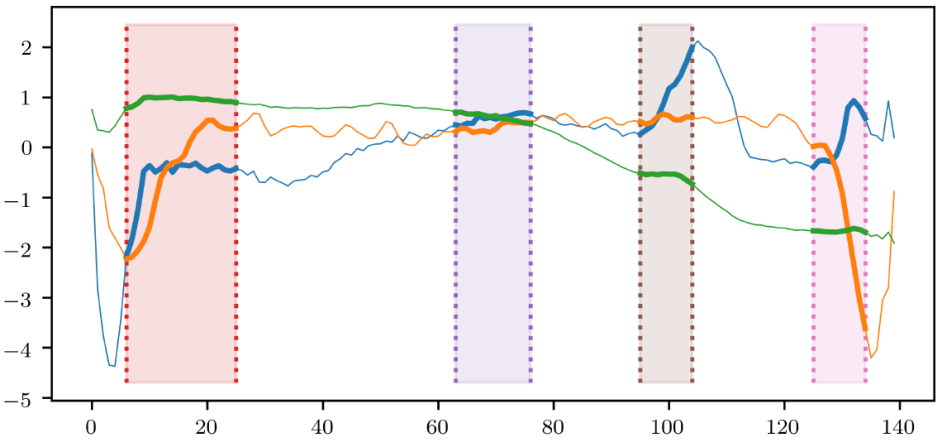
- GAK is part of a family of kernel functions that handles time series data of different lengths.
- It leverages on the DTW concepts presented before in order to map a sequence onto another, i.e. align the two sequences.
- The kernel matrix K is defined as

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

- 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 is computed.

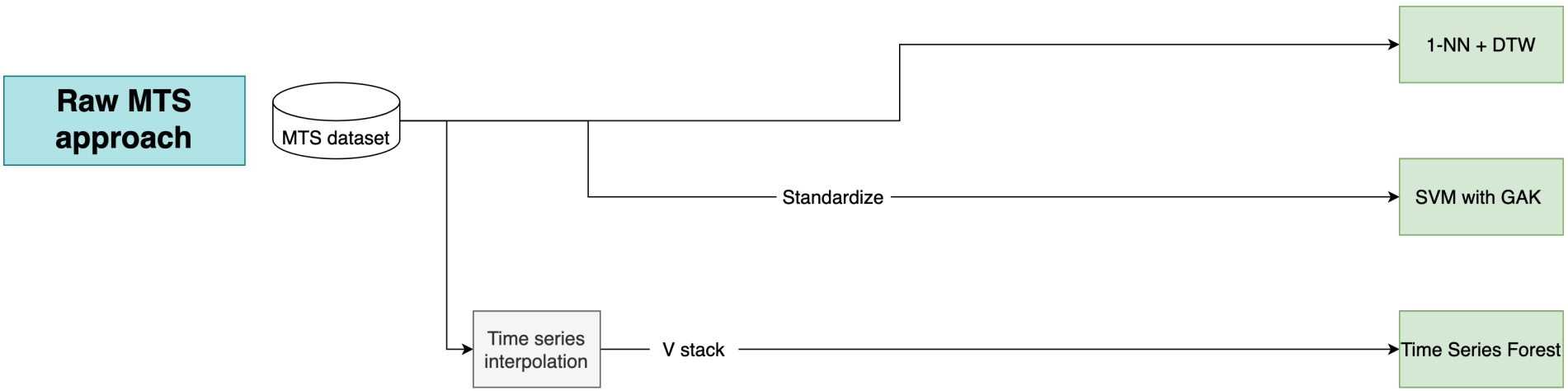
Theory (2e) - Time Series Forests

- Time Series Forests is an ensemble method which works only with datasets of MTS of equal lengths m and described by n predictors.
- The following process is repeated B times, obtaining a forest of decision trees.
  - $\sqrt{nm}$ random intervals of different lengths are generated.
  - The random intervals are applied to each time series obtaining $\sqrt{nm}$ subsequences of different lengths for each time series.
    - Mean, standard deviation and slope are computed for each subsequence, obtaining for each MTS a vectorial representation.
  - A decision tree is then grown from this new dataset.



	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

Experiments (2f) - *Raw MTS* pipeline



Experiments (2g) - Results

- Best hyperparameters from the 5-fold cross-validation and accuracy results on the test dataset:
  - KNN with DTW is often considered as a baseline in the time series classification setting. In this task it achieves a solid 0.781.
  - SVM with GAK outperforms the baseline set by KNN with DTW, reaching a 0.849.
  - The best performing algorithm for the *Raw MTS* approach is Time Series Forest which achieves an accuracy of more than 0.93.
    - The downside is that all the MTS must be of the same length.

Algorithm	**Hyperparameters best combination**	**Acc.**
KNN with DTW	Number of neighbours $= 1$, uniform weighting scheme	0.781
SVM with GAK	C $= 0.0001$, $\beta = 1$	0.849
Time Series Forest	200 estimators, min. interval length $= 4$	**0.932**

Table: Best hyperparameters combination for each method and final accuracy on the test dataset.

Experiments (3) - Final Results

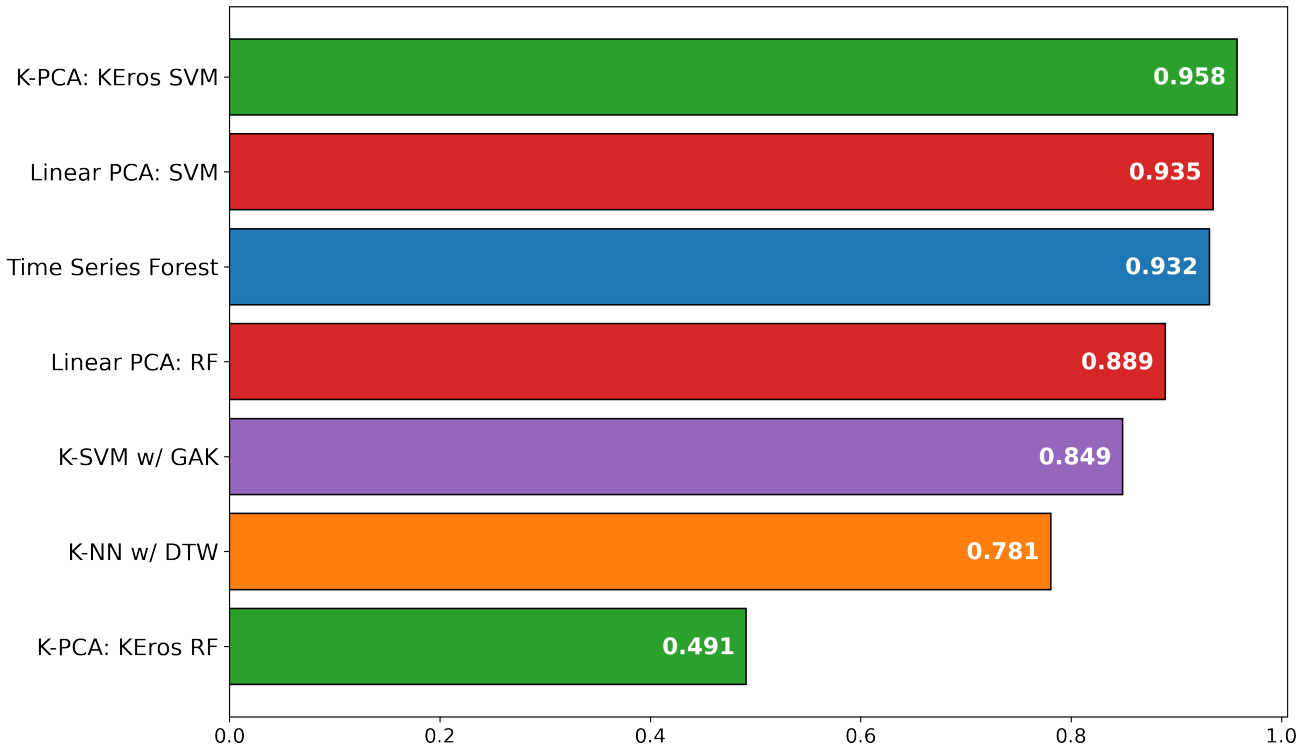


Table of Contents

Conclusion

- In this work we investigated how dimensionality reduction techniques such as PCA and K-PCA need to be extended in order to tackle time series data:
  - They helped reaching very high performances, suggesting that they are as powerful in the time series setting as they are in the standard one.
- We also investigated how classic ML classification algorithms can deal, in different ways, with MTS data:
  - Time Series Forest was the most promising one.
    - It also reached high accuracy results without the need of dimensionality reduction.
    - The downside is that it requires data to be of the same length, making it harder to be adopted in a real-life scenario.
- We consider KEros as the possible candidate method for a real-life scenario.
  - Though no new data can be introduced after training the method, since both PCA and SVM are not incremental.
- A possible application could be translating in real time AUSLAN native speakers and could also be retrained, using the same data framework, on other sign languages.

Thank you!

Lorenzo Scarciglia: lorenzo.scarciglia@studenti.polito.it
Andrea Silvi: andrea.silvi@studenti.polito.it