

# DATA MINING 2

## Time Series Classification

---

Riccardo Guidotti

a.a. 2022/2023



# Time Series Classification

---

- Given a set  $X$  of  $n$  time series,  $X = \{x_1, x_2, \dots, x_n\}$ , each time series has  $m$  ordered values  $x_i = \langle x_{t1}, x_{t2}, \dots, x_{tm} \rangle$  and a class value  $c_i$ .
- The objective is to find a function  $f$  that maps from the space of possible time series to the space of possible class values.
- Generally, it is assumed that all the TS have the same length  $m$ .

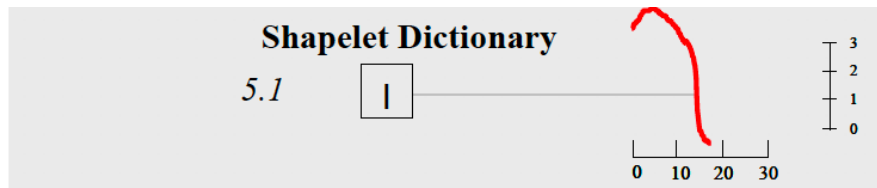
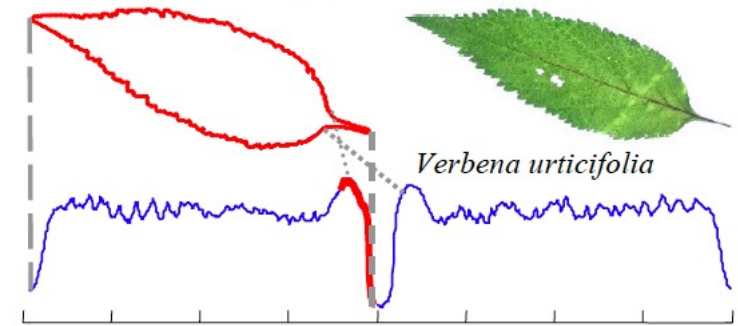
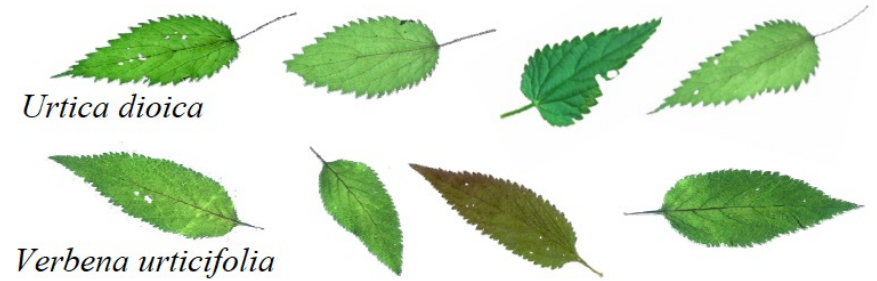
# KNN Classification

---

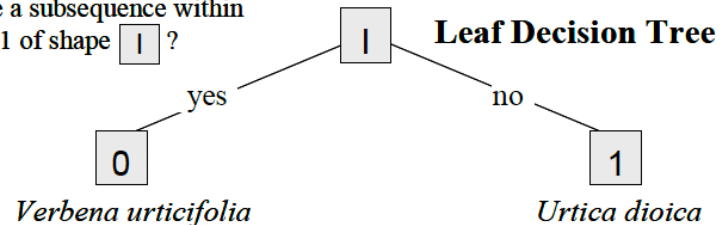
- The most widely used and effective approach for TSC consists in using KNN on the raw time series.
- Pros:
  - Simple
  - Dynamic Time Warping gives much better results than Euclidean distance on many problems.
- Cons:
  - KNN is a lazy classifier and computationally expensive on its own
  - Dynamic Time Warping is very very slow to calculate

# Shapelet-based Classification

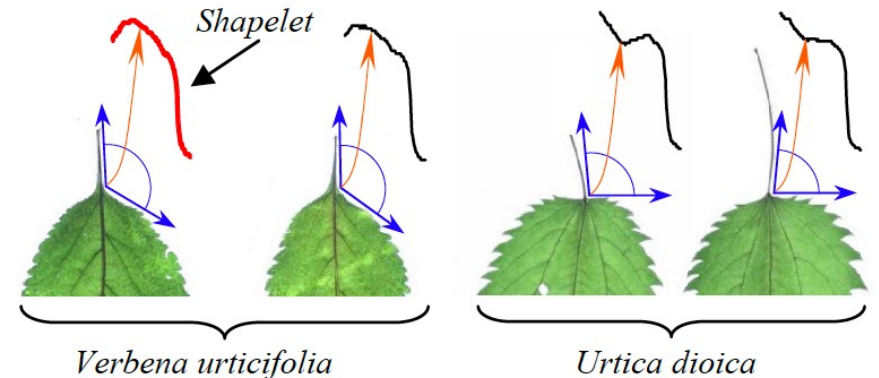
1. Represent a TS as a vector of distances with representative subsequences, namely shapelets.
2. Use it to as input for machine learning classifiers.



Does  $Q$  have a subsequence within a distance 5.1 of shape 1?

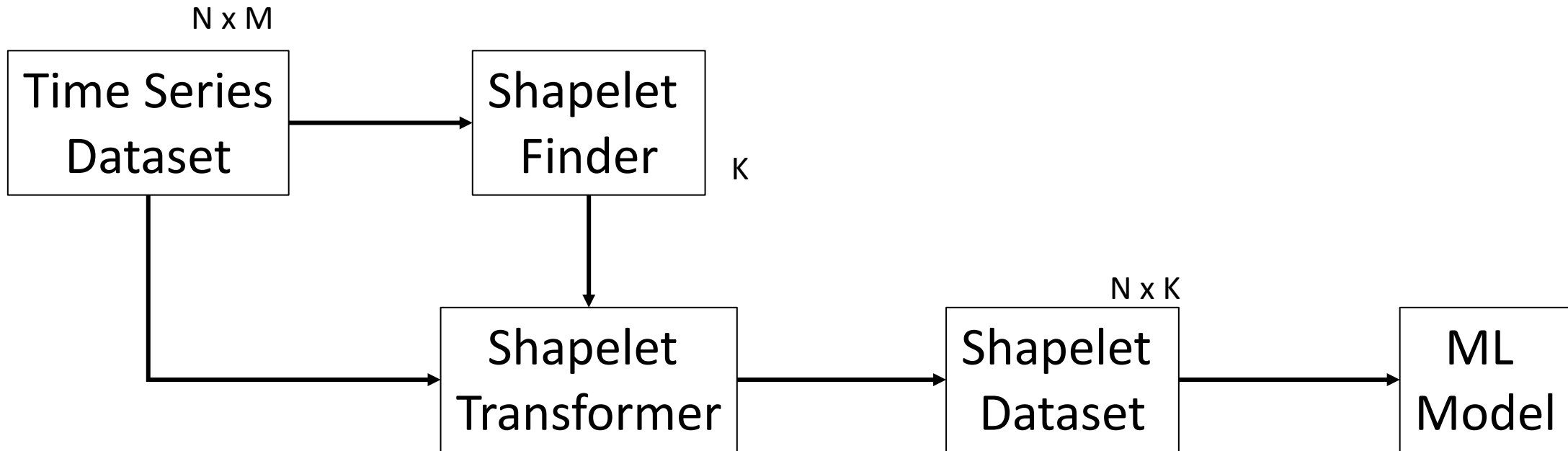


3.2	8.7
1.4	7.9
6.7	4.2
9.2	3.4



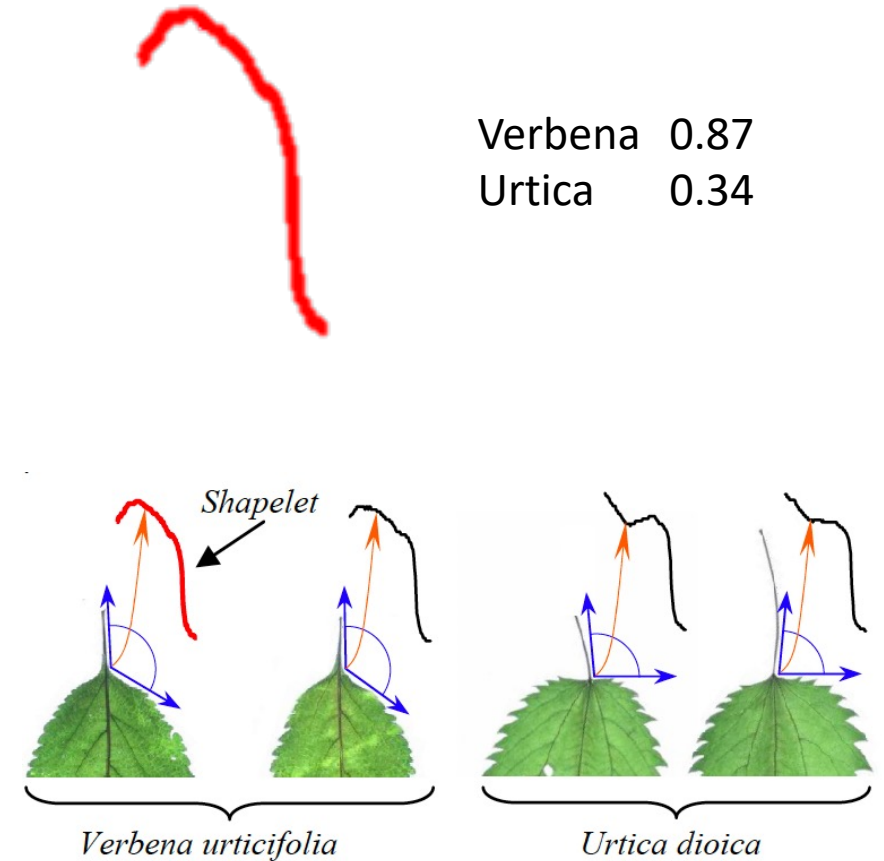
# Shapelet-based Classifiers

---

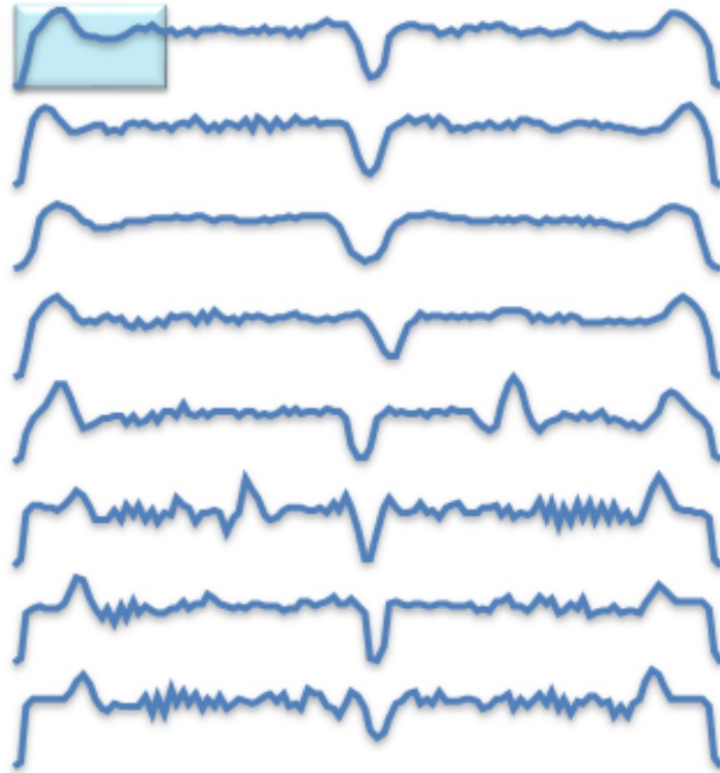


# Time Series Shapelets

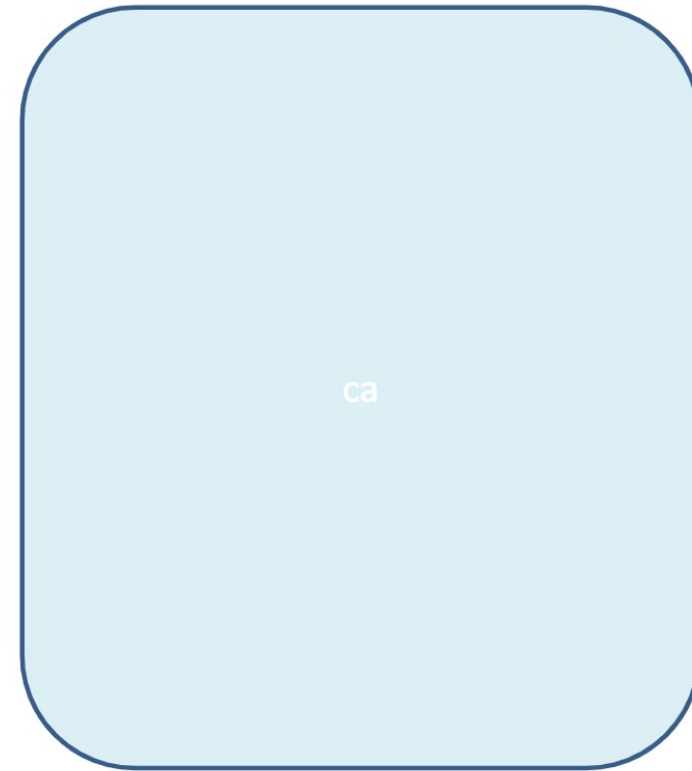
- Shapelets are TS subsequences which are maximally representative of a class.
- Shapelets can provide interpretable results, which may help domain practitioners better understand their data.
- Shapelets can be significantly more accurate/robust because they are *local features*, whereas most other state-of-the-art TS classifiers consider *global features*.



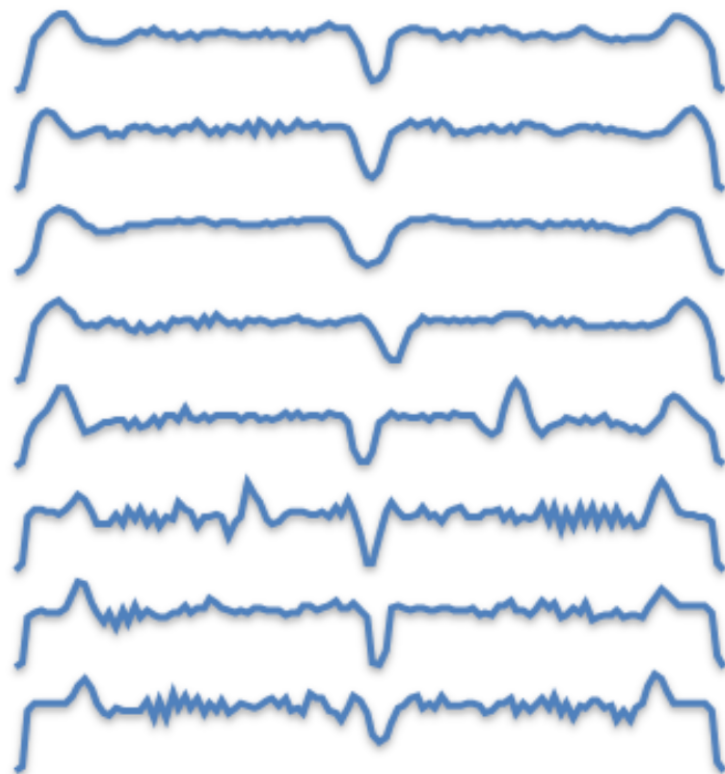
# Extract Subsequences of all Possible Lengths



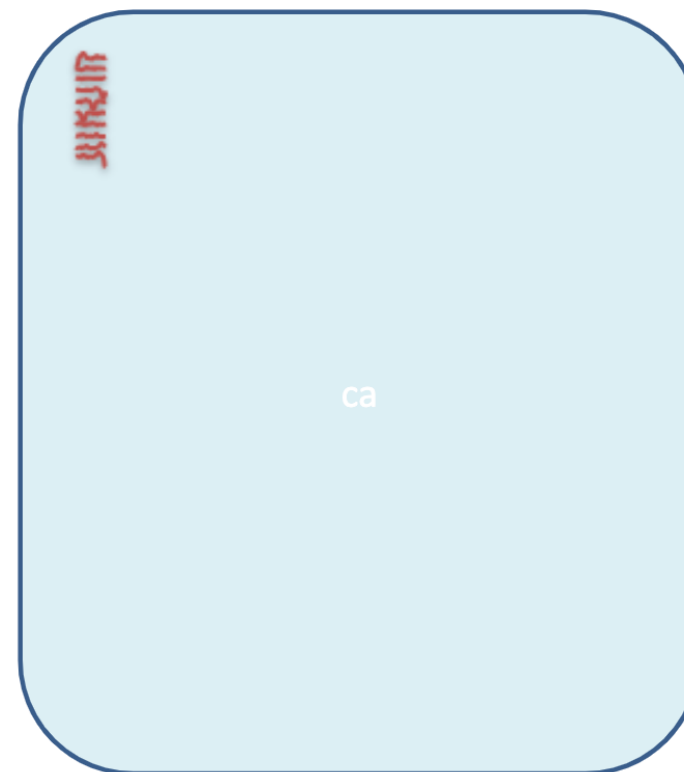
Candidates Pool



# Extract Subsequences of all Possible Lengths

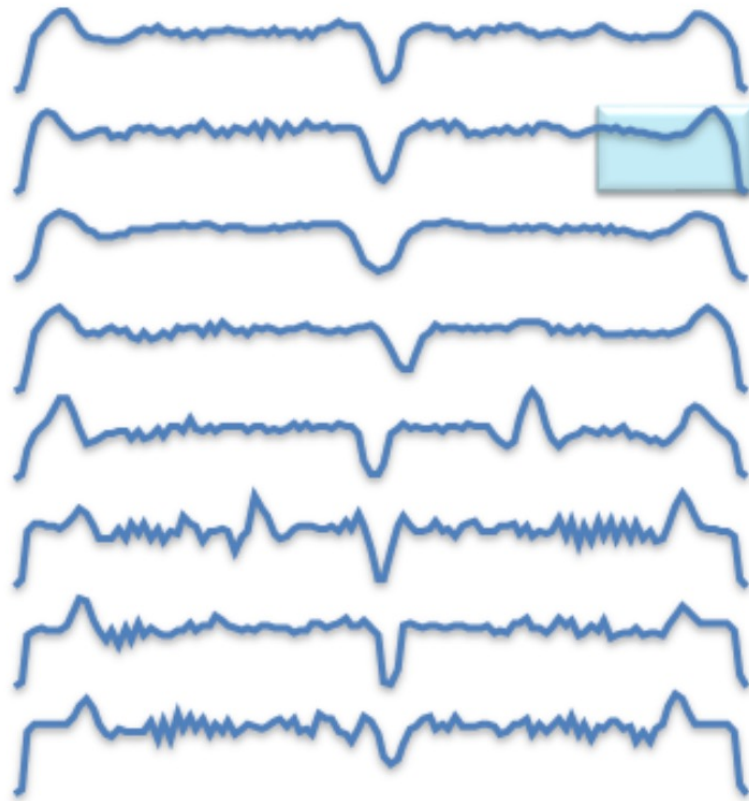


Candidates Pool

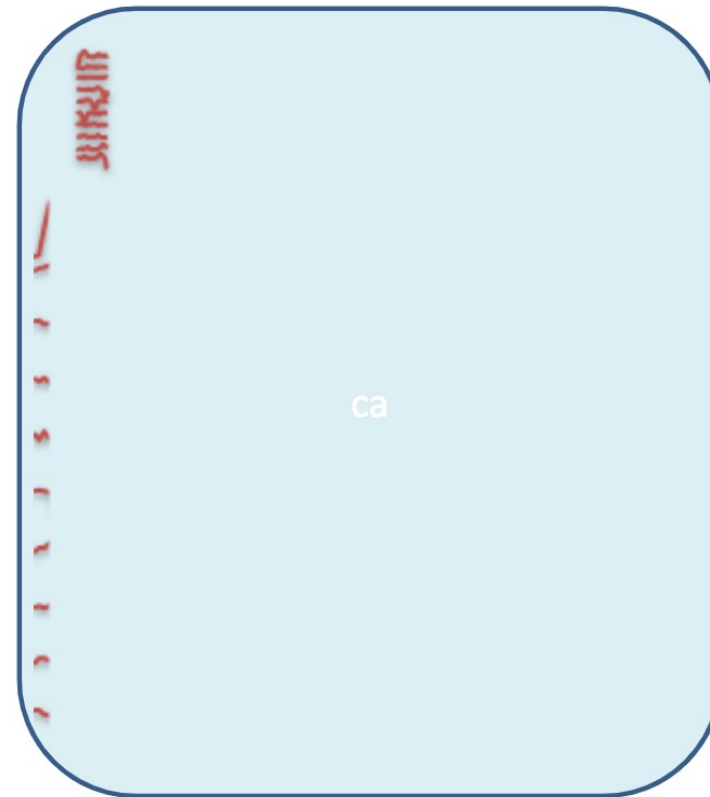




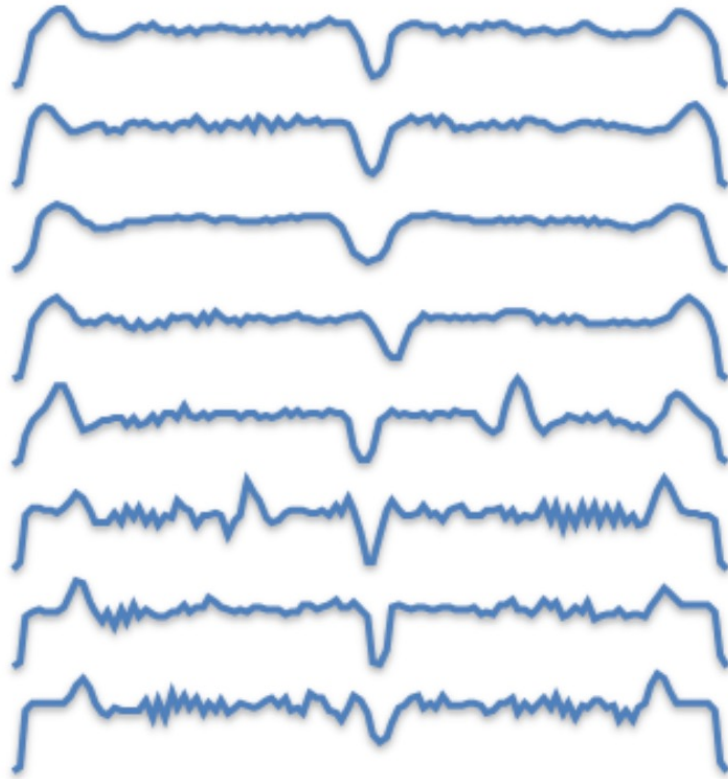
# Extract Subsequences of all Possible Lengths



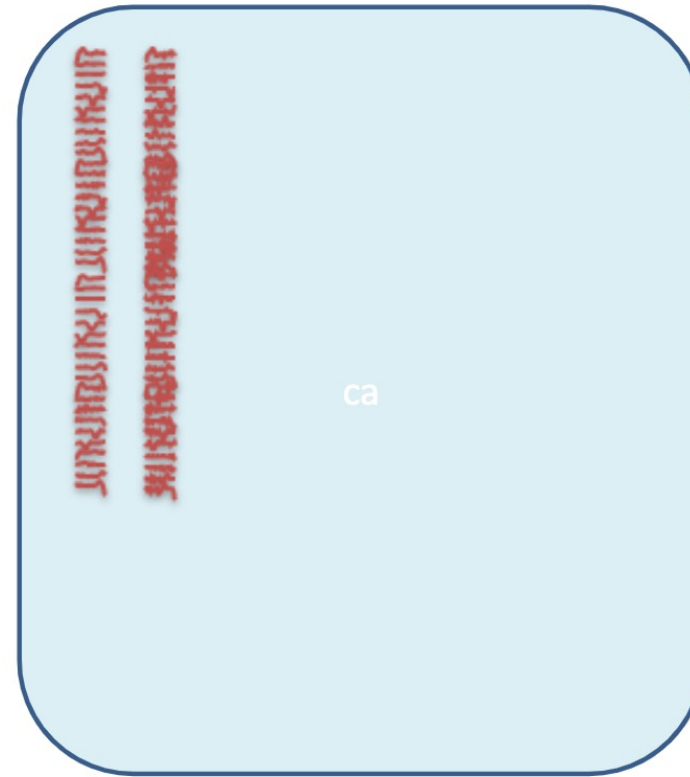
Candidates Pool



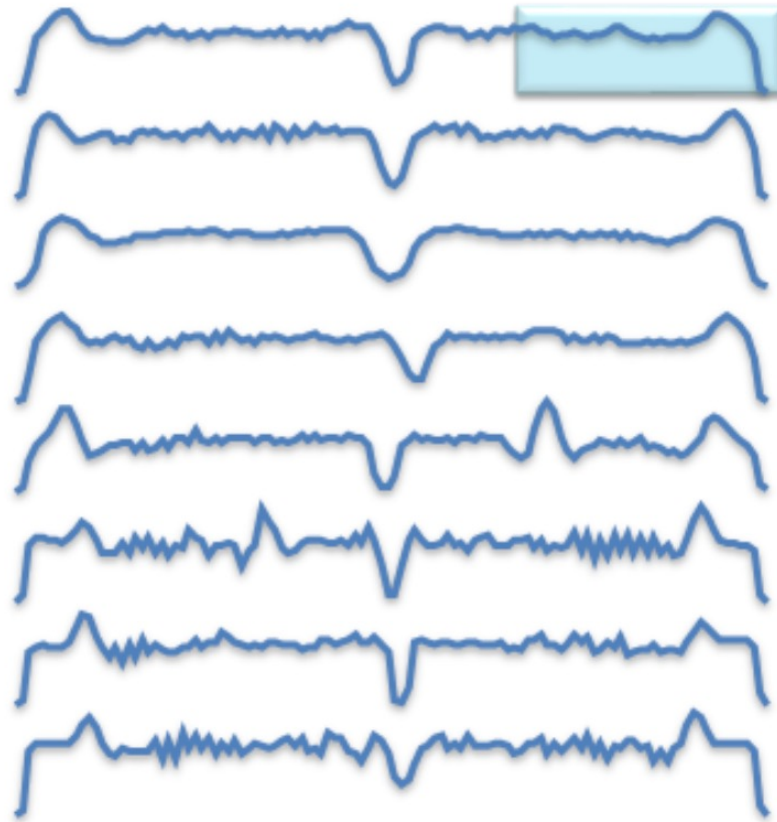
# Extract Subsequences of all Possible Lengths



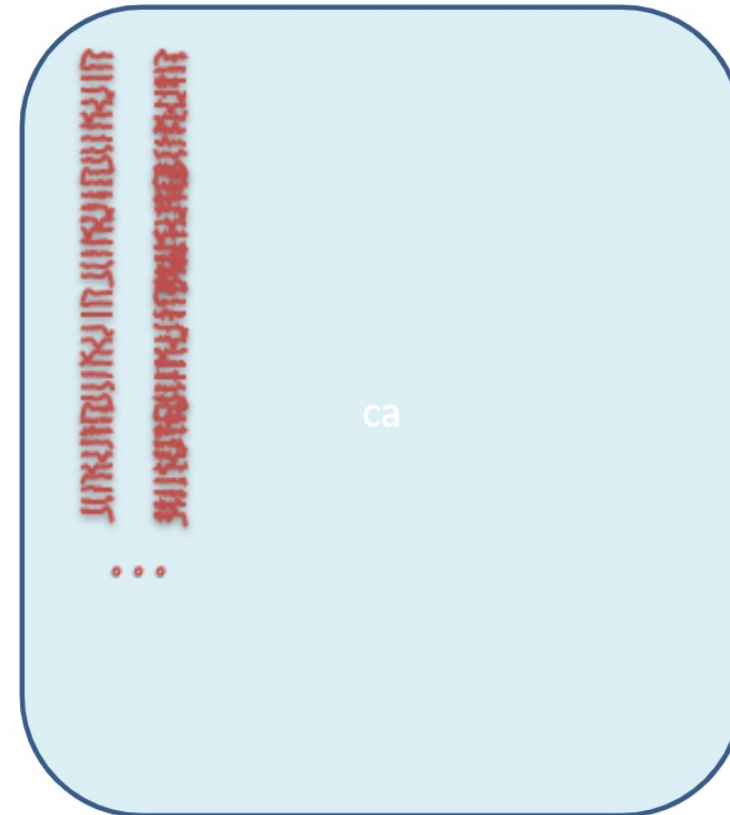
Candidates Pool



# Extract Subsequences of all Possible Lengths

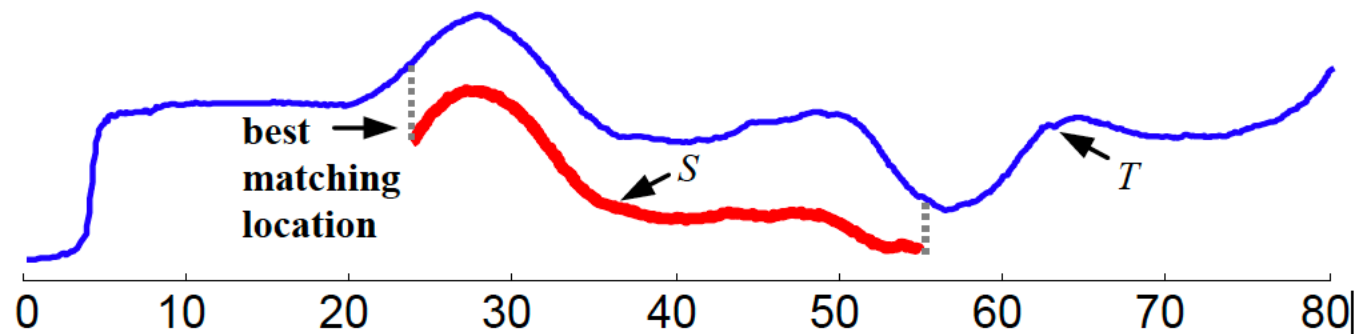


Candidates Pool



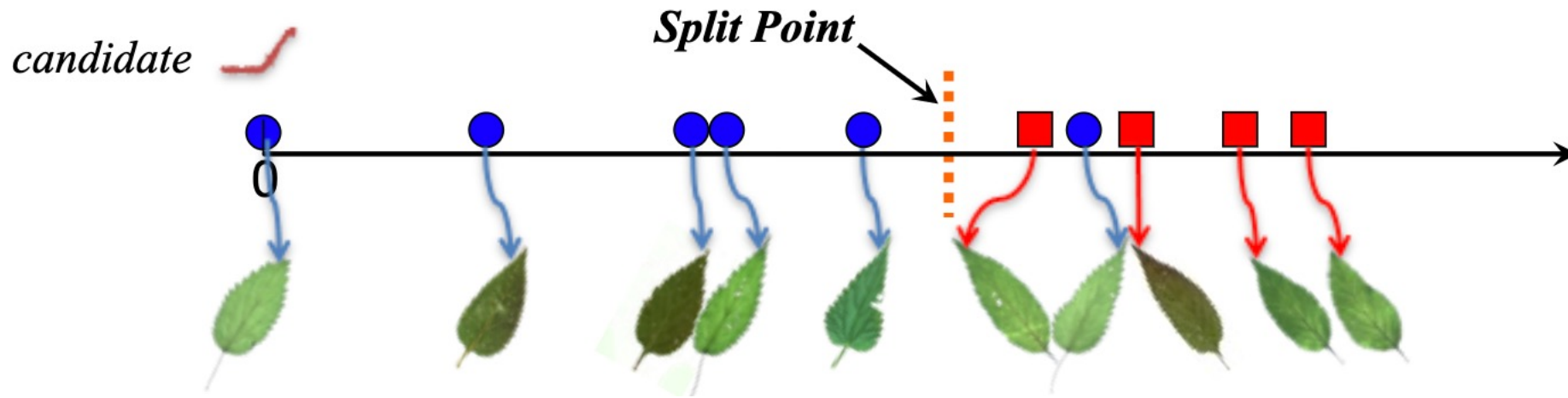
# Distance with a Subsequence

- Distance from the TS to the subsequence  $SubsequenceDist(T, S)$  is a distance function that takes time series  $T$  and subsequence  $S$  as inputs and returns a nonnegative value  $d$ , which is the distance from  $T$  to  $S$ .
- $SubsequenceDist(T, S) = \min(Dist(S, S')), \text{ for } S' \in S_T^{|S|}$
- where  $S_T^{|S|}$  is the set of all possible subsequences of  $T$
- Intuitively, it is the distance between  $S$  and its best matching location in  $T$ .

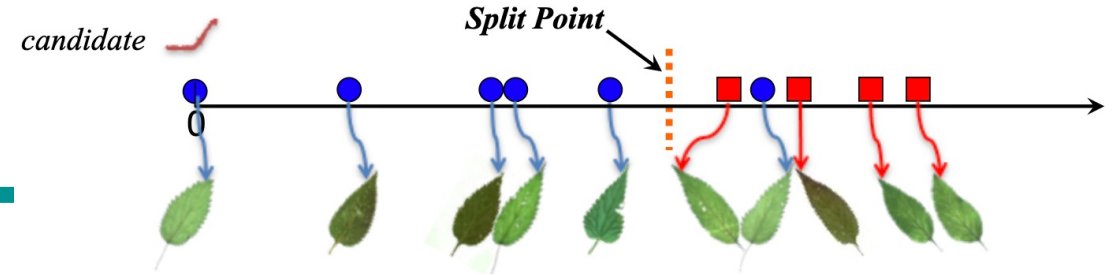


# Testing The Utility of a Candidate Shapelet

- Arrange the TSs in the dataset  $D$  based on the distance from the candidate.
- Find the optimal split point that maximizes the information gain (same as for Decision Tree classifiers)
- Pick the candidate achieving best utility as the shapelet

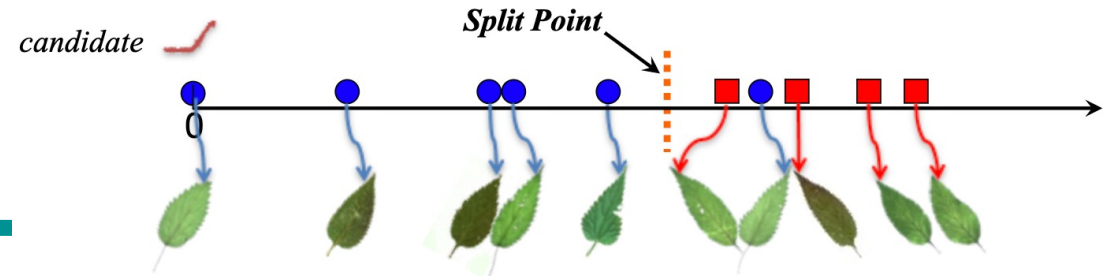


# Entropy



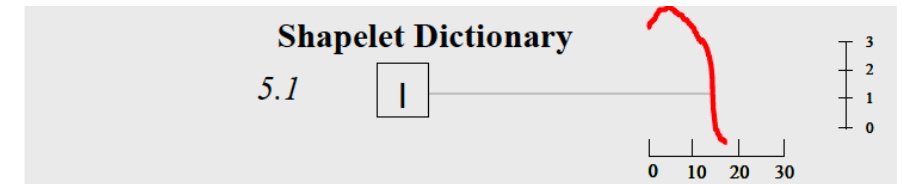
- A TS dataset  $D$  consists of two classes, A and B.
- Given that the proportion of objects in class A is  $p(A)$  and the proportion of objects in class B is  $p(B)$ ,
- The **Entropy** of  $D$  is:  $I(D) = -p(A)\log(p(A)) - p(B)\log(p(B))$ .
- Given a strategy that divides the  $D$  into two subsets  $D_1$  and  $D_2$ , the information remaining in the dataset after splitting is defined by the weighted average entropy of each subset.
- If the fraction of objects in  $D_1$  is  $f(D_1)$  and in  $D_2$  is  $f(D_2)$ ,
- The total entropy of  $D$  after splitting is  $\hat{I}(D) = f(D_1)I(D_1) + f(D_2)I(D_2)$ .

# Information Gain

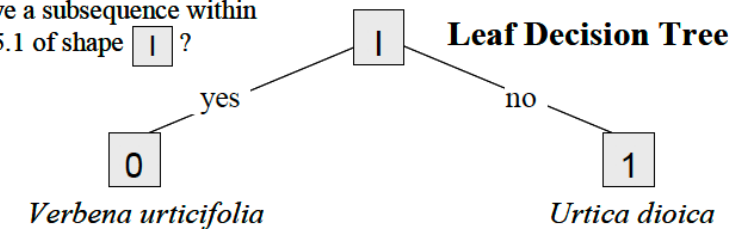


Split point  
distance from  
shapelet = 5.1

- Given a certain split strategy  $sp$  which divides  $D$  into two subsets  $D_1$  and  $D_2$ , the entropy before and after splitting is  $I(D)$  and  $\hat{I}(D)$ .
- The **information gain** for this splitting rule is:
- $Gain(sp) = I(D) - \hat{I}(D) =$
- $= I(D) - f(D_1)I(D_1) + f(D_2)I(D_2).$
- We use the distance from  $T$  to a shapelet  $S$  as the splitting rule  $sp$ .



Does  $Q$  have a subsequence within a distance 5.1 of shape  $I$ ?



# Problem

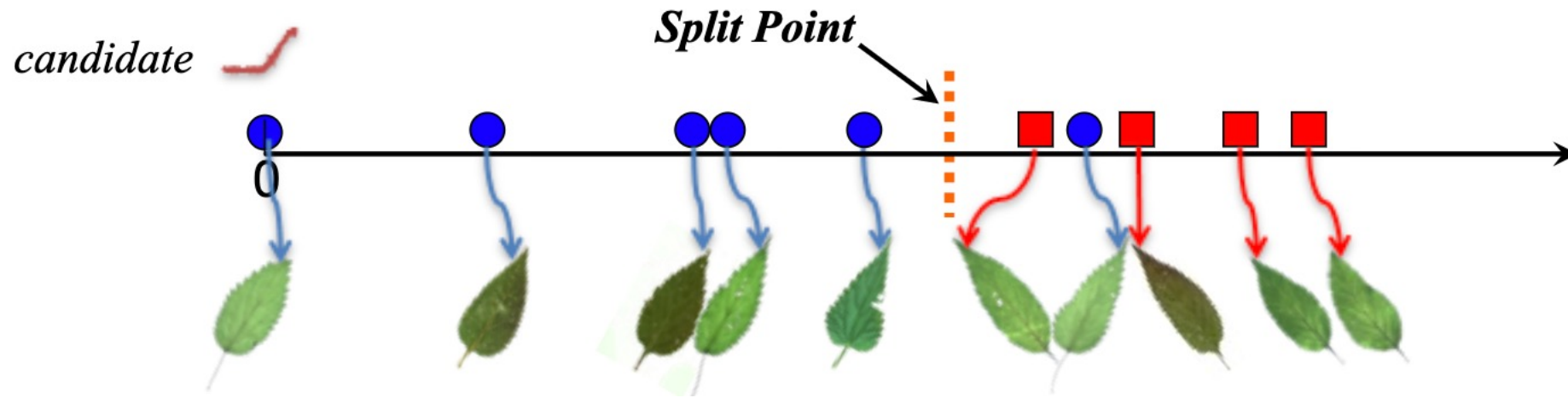
---

- The total number of candidate is 
$$\sum_{l=MINLEN}^{MAXLEN} \sum_{T_i \in D} (|T_i| - l + 1)$$
- For each candidate you have to compute the distance between this candidate and each training sample
- For instance
  - 200 instances with length 275
  - 7,480,200 shapelet candidates



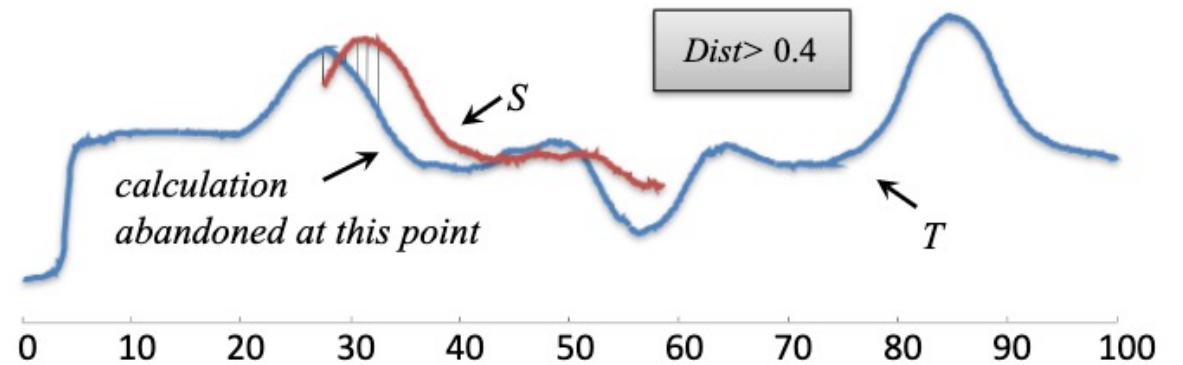
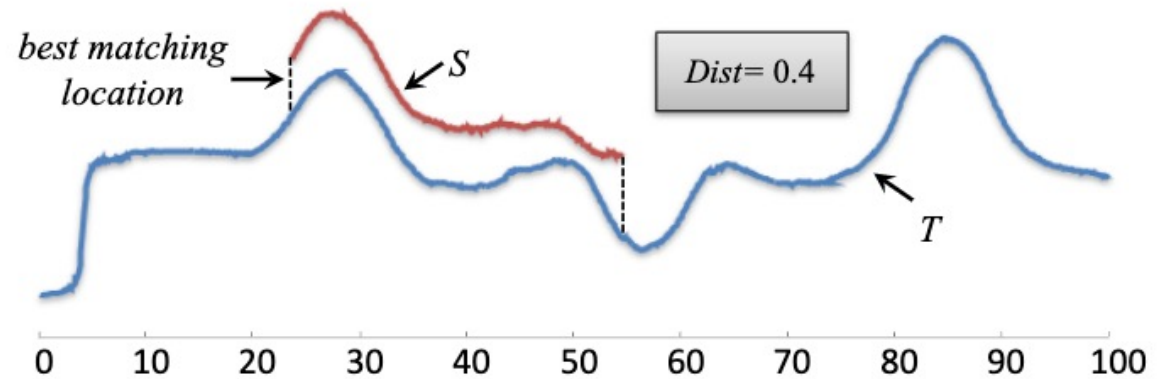
# Speedup

- Distance calculations from TSs to shapelet candidates is expensive.
- Reduce the time in two ways
  - Distance Early Abandon
  - reduce the distance computation time between two TS
- Admissible Entropy Pruning
  - reduce the number of distance calculations



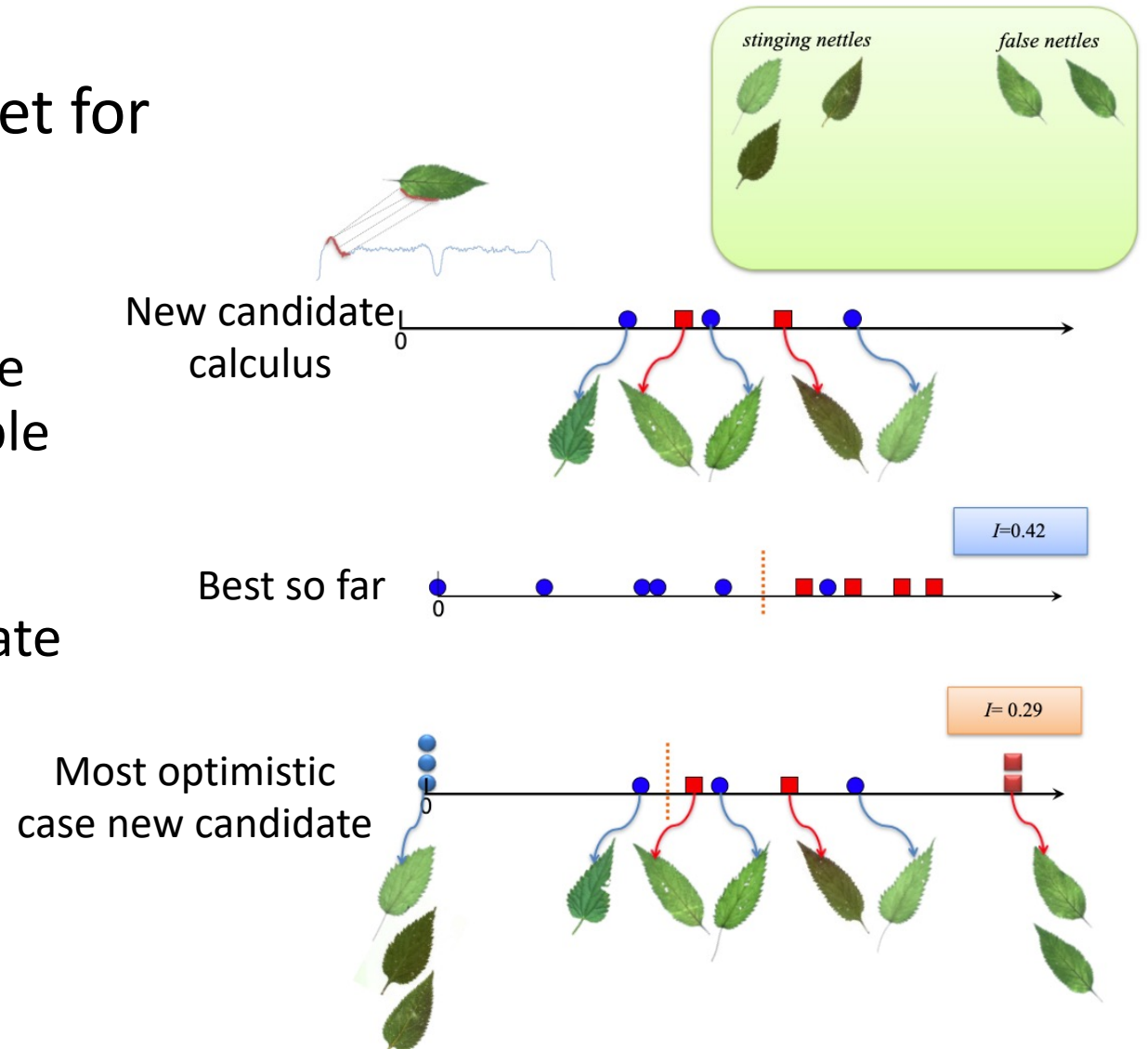
# Distance Early Abandon

- We only need the minimum distance.
- Method
  - Keep the best-so-far distance
  - Abandon the calculation if the current distance is larger than best-so-far.



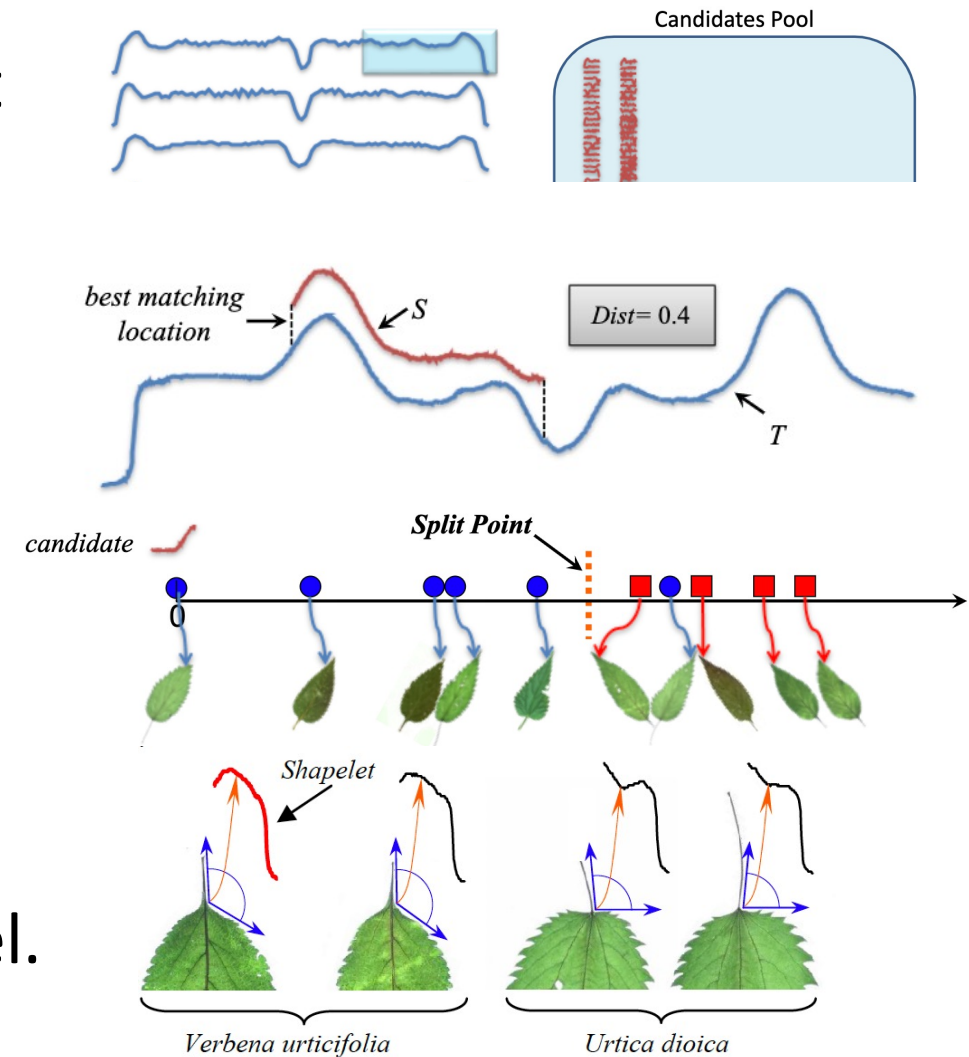
# Admissible Entropy Pruning

- We only need the best shapelet for each class
- For a candidate shapelet
  - We do not need to calculate the distance for each training sample
  - After calculating some training samples, the upper bound of information gain  $<$  best candidate shapelet
  - Stop calculation
  - Try next candidate



# Shapelet Summary

1. Extract all possible subsequences of a set given lengths (candidate shapelets)
2. For each candidate shapelet
  1. Calculate the distance with each time series keeping the minimum distance (best alignment)
  2. Evaluate the discriminatory effect of the shapelet through the Information Gain
3. Return the  $k$  best shapelets with the highest Information Gain.
4. Transform a dataset and train a ML model.



# An Alternative Way for Extracting Shapelets

- The minimum distances (M) between Ts and Shapelets can be used as predictors to approximate the TSs label (Y) using a linear model (W):

$$\hat{Y}_i = W_0 + \sum_{k=1}^K M_{i,k} W_k, \quad \forall i \in \{1, \dots, I\}$$

- A logistic regression loss can measure the quality of the prediction:

$$\mathcal{L}(Y, \hat{Y}) = -Y \ln \sigma(\hat{Y}) - (1 - Y) \ln (1 - \sigma(\hat{Y}))$$

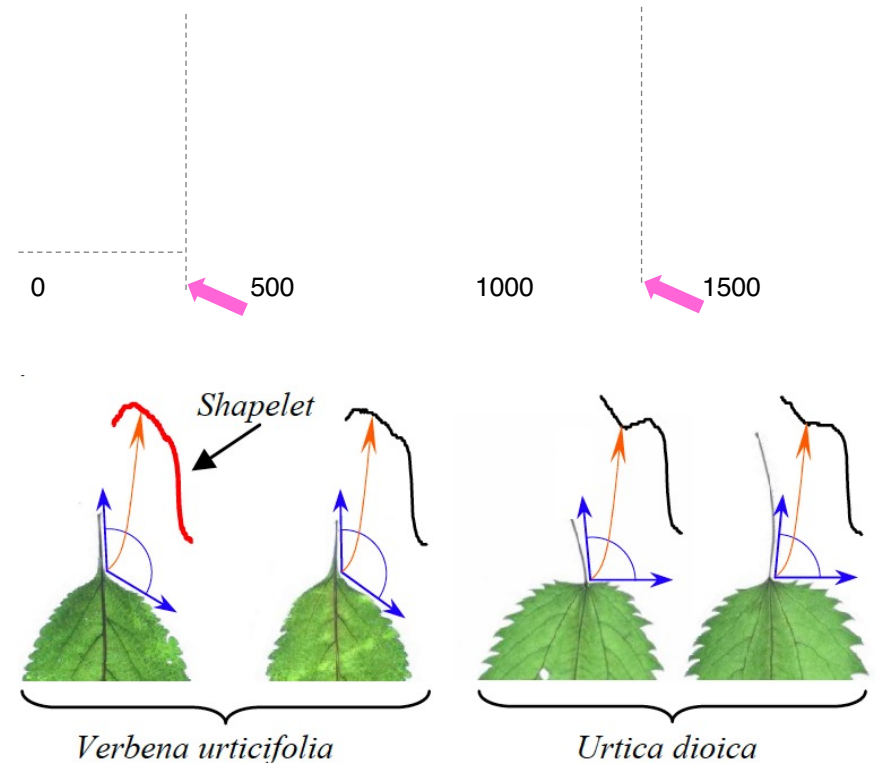
- The objective is to minimize a regularized loss function across all the instances (I) :

$$\operatorname{argmin}_{S, W} \mathcal{F}(S, W) = \operatorname{argmin}_{S, W} \sum_{i=1}^I \mathcal{L}(Y_i, \hat{Y}_i) + \lambda_W \|W\|^2$$

- We can find the optimal shapelet for the objective function in a NN fashion by updating the shapelets in the minimum direction of the objective, hence the first gradient. Similarly, the weights can be jointly updated towards minimizing the objective function.

# Motif/Shapelet Summary

- A **motif** is a repeated pattern/subsequence in a given TS.
- A **shapelet** is a pattern/subsequence which is maximally representative of a class with respect to a given dataset of TSs.



# References

- Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. Chin-Chia Michael Yeh et al. 1997
- Time Series Shapelets: A New Primitive for Data Mining. Lexiang Ye and Eamonn Keogh. 2016.
- Josif Grabocka, Nicolas Schilling, Martin Wistuba, Lars Schmidt-Thieme (2014): Learning Time-Series Shapelets, in Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2014
- Deep learning for time series classification: a review. Hassan Ismail Fawaz et al. 2019.

arXiv:1809.04356v4 [cs.LG] 14 May 2019

## Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets

Chin-Chia Michael Yeh, Yan Zhu, Lindmila Ulanova, Nurjahan Begam, Yifei Ding, Hong Anli Du, Diego Furrado Silva, Abdallah Mosen, and Eamonn Keogh

University of California, Riverside, Universidade de São Paulo, University of New Mexico (moy003, yzhu013, hdu001, abeg001, yzhu007, hdu001)@ucr.edu, denqofab@usp.br, amosen@unm.edu, eamonn@cs.ucr.edu

**Abstract**—The all-pairs-similarity-search (or similarity join) problem has been extensively studied for text and a handful of other datasets. However, surprisingly little progress has been made on similarity joins for time series subsequences. The lack of progress probably stems from the difficult nature of the problem. For even modest sized datasets the obvious nested-loop algorithm can take months, and typical speed-up techniques in this domain (e.g., indexing, lower-bounding, irregularly-inequality pruning and self-adjuncting) at best produce one or two orders of magnitude speedup. In this work we introduce a novel scalable algorithm for time series subsequence all-pairs-similarity-search. For exceptionally large datasets, the algorithm can be trivially cast as an anytime algorithm and produce high-quality approximate solutions in reasonable time. The exact similarity join algorithm computes the answer to the time series self-join and time series discord problem as a side-effect, and our algorithm incidentally provides the fastest known algorithm for both these extensively-studied problems. We demonstrate the utility of our ideas for many time series data mining problems, including motif discovery, anomaly discovery, shapelet discovery, semantic segmentation, density estimation, and contrast set mining.

**Keywords**—Time Series; Similarity Joins; Motif Discovery

**1. INTRODUCTION**  
The all-pairs-similarity-search (also known as similarity join) problem comes in several variants. The basic task is: Given a collection of data objects, retrieve the nearest neighbor for each object in the text domain. The algorithm has applications in a host of problems, including community discovery, duplicate detection, collaborative filtering, clustering, and query refinement [1]. While virtually all text processing algorithms have automatic in time series data mining, there has been surprisingly little progress on Time Series subsequence All-Pairs-Similarity-Search (TSAPSS).

We believe that this lack of progress stems not from a lack of interest in this useful primitive, but from the daunting nature of the problem. Consider the following example that reflects the needs of an industrial collaborator. A boiler's chemical refinery reports pressure once a minute. After a year, we have a time series of length 524,608. A plant manager may wish to do a similarity self-join on this data with week-long subsequences (10,800) to discover repeating regimes (summer vs. winter or light distillate vs. heavy distillate, etc). The obvious nested loop algorithm requires 112,800,602,560 Euclidean distance computations. If we assume each one takes 0.0001 seconds, then the join will take 11.8 days. The core combination of this work is to show that we can reduce this time to 4-5 hours, using an off-the-shelf desktop computer. Moreover, we show that this join can be computed and/or updated incrementally. This we could maintain this join essentially forever on a standard

This is the author's version of an article published in Data Mining and authenticated version is available online at: [https://doi.org/10.1007/978-1-4939-9809-6\\_55](https://doi.org/10.1007/978-1-4939-9809-6_55)

## Deep learning for time series classification

Hassan Ismail Fawaz<sup>1</sup> · Germain Forestier<sup>1,2</sup> · Jonathan W. Hussane Idoumghar<sup>1</sup> · Pierre-Alain Muller<sup>1</sup>

**Abstract** Time Series Classification (TSC) is an important and challenging task. With the increase of time series data availability, hundreds of TSC algorithms have been proposed since 2015 (Bagnall et al., 2017). Due to their natural temporal ordering, time series data such as text and audio can also be processed with DNNs to reap for document classification and speech recognition. In this article, the state-of-the-art performance of deep learning algorithms for TSC by present most recent DNN architectures for TSC. We give an overview of the applications in various time series domains under a unified taxonomy provide an open source deep learning framework to the TSC community and 12 multivariate time series datasets. By training 8,73 time series datasets, we propose the most exhaustive study of DNN

**Keywords** Deep learning · Time series · Classification · Review

## 1 Introduction

During the last two decades, Time Series Classification (TSC) has been considered as one of the most challenging problems in data mining (Yang and Wu, 2006; Ealing and Agon, 2012). With the increase of temporal data availability (Silva et al., 2018), hundreds of TSC algorithms have been proposed since 2015 (Bagnall et al., 2017). Due to their natural temporal ordering, time series data are present in almost every task that requires some sort of human cognitive process (Långkvist et al., 2014). In fact, any classification problem, using data that is registered taking into account some notion of ordering, can be cast as a TSC problem (Cristian Borges Gombosi, 2017). Time series are encountered in many real-world applications ranging from electronic health records (Rajkumar et al., 2018) and human activity recognition (Nweke et al., 2018; Wang et al., 2018) to acoustic scene classification (Nwe et al., 2017) and cyber-security (Susto et al., 2018). In addition, the diversity of the datasets' types in the UCR/UEA archive (Chen et al., 2015b; Bagnall et al., 2017) (the largest repository of time series datasets) shows the different applications of the TSC problem.

✉ H. Ismail Fawaz  
E-mail: hassan.ismail.fawaz@uniba.fr  
<sup>1</sup>IRDMAS, Université Haute Alsace, Mulhouse, France  
<sup>2</sup>Faculty of IT, Monash University, Melbourne, Australia

## Time Series Shapelets: A New Primitive for Data Mining

Lexiang Ye  
Dept. of Computer Science & Engineering  
University of California, Riverside, CA 92521  
lexiang@cs.ucr.edu

Eamonn Keogh  
Dept. of Computer Science & Engineering  
University of California, Riverside, CA 92521  
eamonn@cs.ucr.edu

**ABSTRACT**  
Classification of time series has been attracting great interest over the past decade. Recent empirical evidence has strongly suggested that the simple nearest neighbor algorithm is very difficult to beat for most time series problems. While this may be considered good news, given the simplicity of implementing the nearest neighbor algorithm, there are some negative consequences of this. First, the nearest neighbor algorithm requires storing and searching the entire dataset, resulting in a time and space complexity that limits its applicability to especially large datasets. Second, beyond near classification accuracy, we often wish to gain some insight into the data.

In this work we introduce a new time series primitive, time series shapelets, which addresses these limitations. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. As we shall show with extensive empirical evaluation on diverse datasets, algorithms based on the time series shapelet primitive can be interpreted, more accurate and significantly faster than state-of-the-art classifiers.

**Categories and Subject Descriptors**  
H.2.8 Database Management; Database Applications · Data Mining

## General Terms

Algorithms; Experimentation

## 1. INTRODUCTION

While the last decade has seen a huge interest in time series classification, to date the most accurate and robust method is the simple nearest neighbor algorithm (Hil2][14]. While the nearest neighbor algorithm has the advantages of simplicity and not requiring extensive parameter tuning, it does have several important disadvantages. Chief among these are its space and time requirements, and the fact that it does not tell us anything about why a particular object was assigned to a particular class.

In this work we present a novel time series data mining primitive called time series shapelets. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. While we believe there are many uses for this primitive, one obvious implication of them is to mitigate the two weaknesses of the nearest neighbor algorithm. Algorithms based on time series shapelets are multi-epoch or batch types of all in part of the data for persons in classrooms are granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear the notice and the full citation on the first page. To copy otherwise, to republish, to post in servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright 2009 ACM 978-1-60558-493-9/09/06...\$5.00.

Because we are defining and solving a new problem, we will take some time to consider a detailed motivating example. Figure 1 shows some examples of leaves from two classes, *Urtica dioica* (stinging nettle) and *Ferula urticifolia*. These two plants are commonly confused, hence the colloquial name "false nettle" for *Ferula urticifolia*.

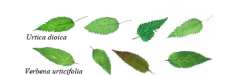


Figure 1: Sample of leaves from two species. Note that several leaves have the same-like shape.

Suppose we wish to build a classifier to distinguish these two plants, what features should we use? Since the main variability of color and size within each class completely dwarfs the inter-variability between classes, our best hope is based on the shapes of the leaves. However, as we can see in Figure 1, the differences in the global shape are very subtle. Furthermore, it is very common for leaves to have distinctive or "eccentric" due to insect damage, and these are very likely to confuse any global measures of shape. Instead we attempt the following. We first convert each leaf into a one-dimensional representation as shown in Figure 2.

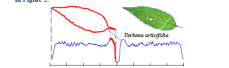
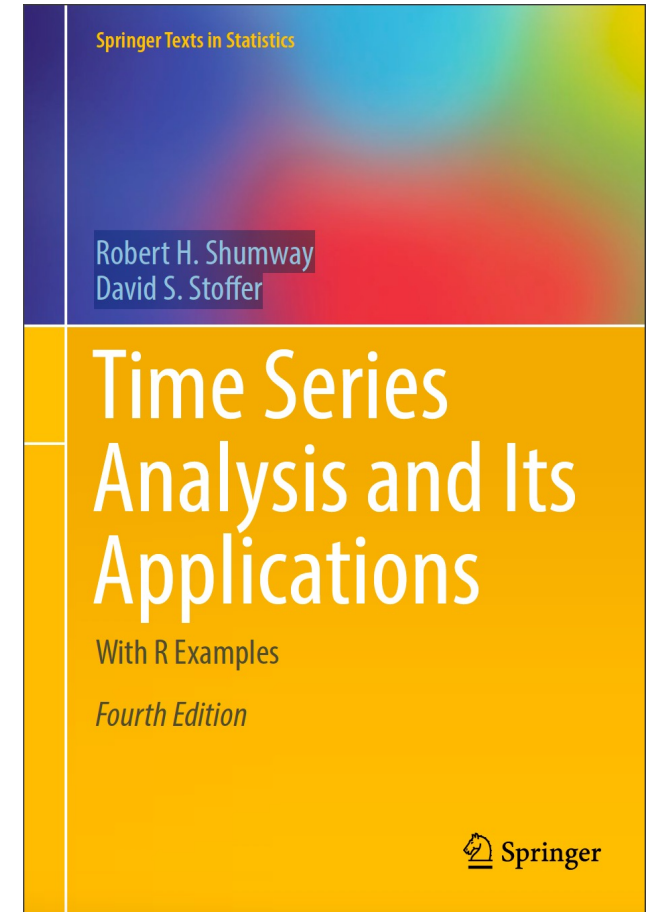


Figure 2: A shape can be converted into a one-dimensional "time series" representation. The reason for the highlighted section of the time series will be made apparent shortly.

Such representations have been successfully used for the classification, clustering and outlier detection of shapes in recent years [8]. However, here we find that using a nearest neighbor classifier with either the (position-normalized) Euclidean distance or Dynamic Time Warping (DTW) distance does not significantly outperform random guessing. The reason for this poor performance of these otherwise very competitive classifiers seems to be due to the fact that the data is somewhat noisy (i.e. insect holes, and different stem lengths), and this noise is enough to swamp the subtle differences in the shapes.

# References

- Selective review of offline change point detection methods. Truong, C., Oudre, L., & Vayatis, N. (2020). *Signal Processing*, 167, 107299.
- Time Series Analysis and Its Applications. Robert H. Shumway and David S. Stoffer. 4<sup>th</sup> edition. (<https://www.stat.pitt.edu/stoffer/tsa4/tsa4.pdf>)
- Mining Time Series Data. Chotirat Ann Ratanamahatana et al. 2010. ([https://www.researchgate.net/publication/227001229\\_Mining\\_Time\\_Series\\_Data](https://www.researchgate.net/publication/227001229_Mining_Time_Series_Data))
- Dynamic Programming Algorithm Optimization for Spoken Word Recognition. Hiroaki Sakode et al. 1978.
- Experiencing SAX: a Novel Symbolic Representation of Time Series. Jessica Line et al. 2009
- Compression-based data mining of sequential data. Eamonn Keogh et al. 2007.





# TSC State-of-The-Art

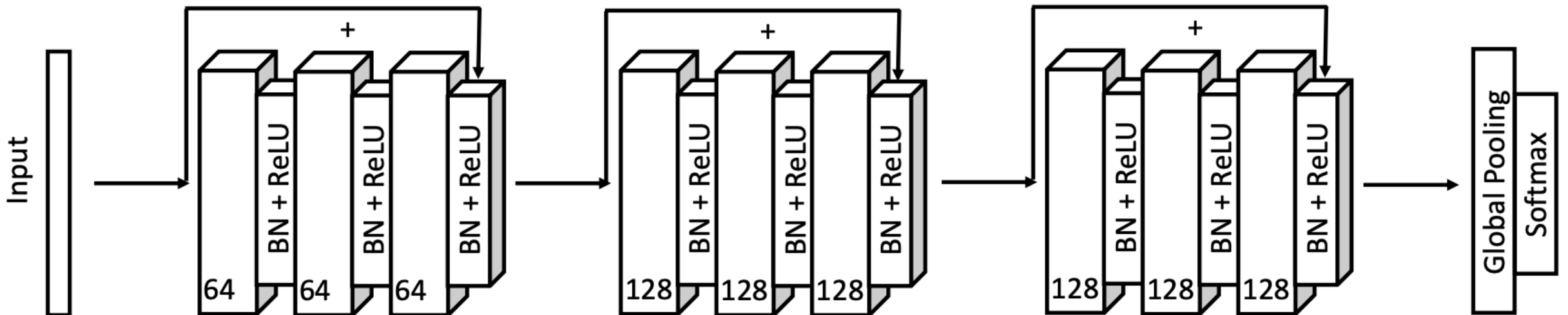
---

A special thank to Francesco Spinnato for the slides

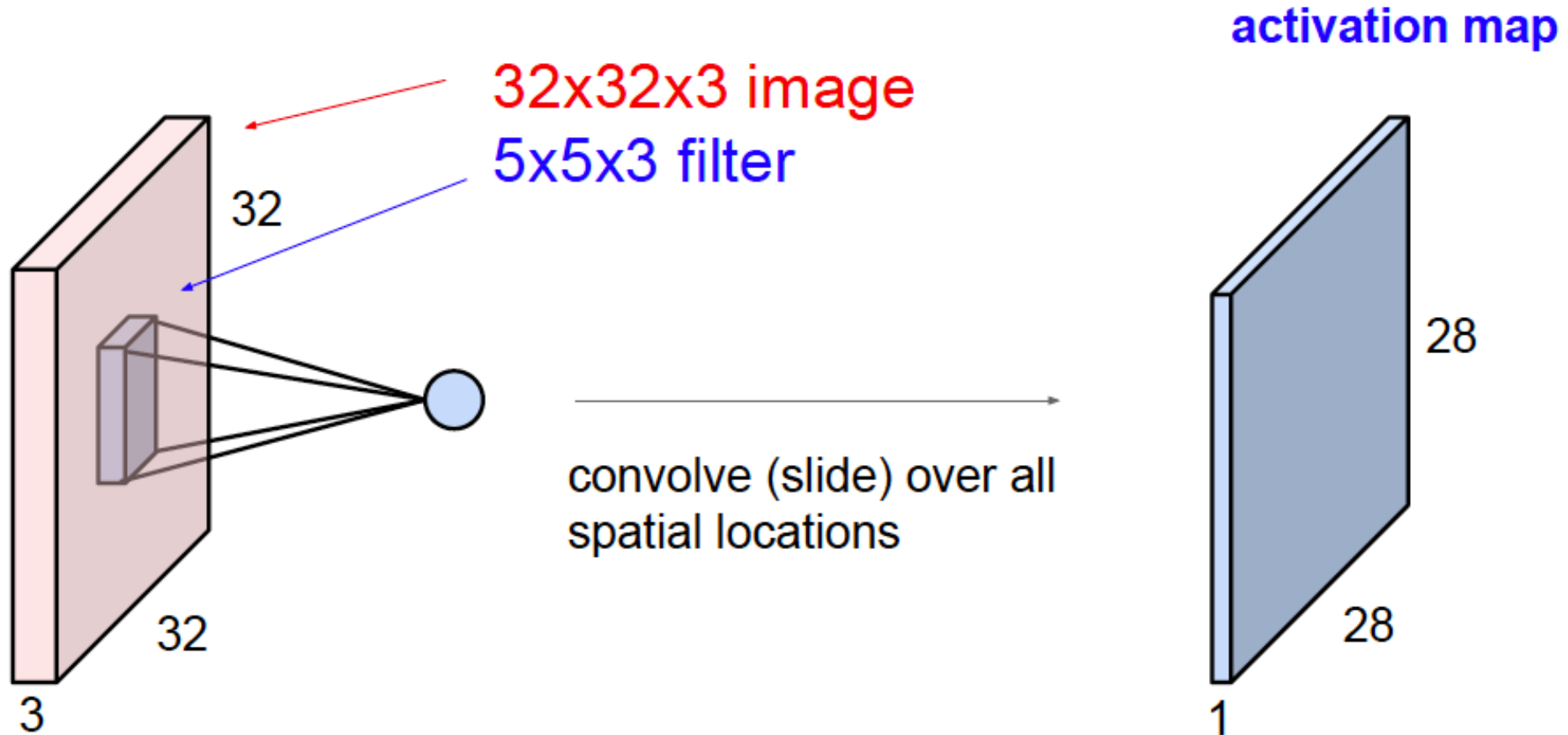


# ResNet

- Three consecutive blocks, comprised of three convolutional layers, connected by residual 'shortcut' connections.
- The blocks are followed by global average pooling and softmax layers to form features and subsequent predictions.

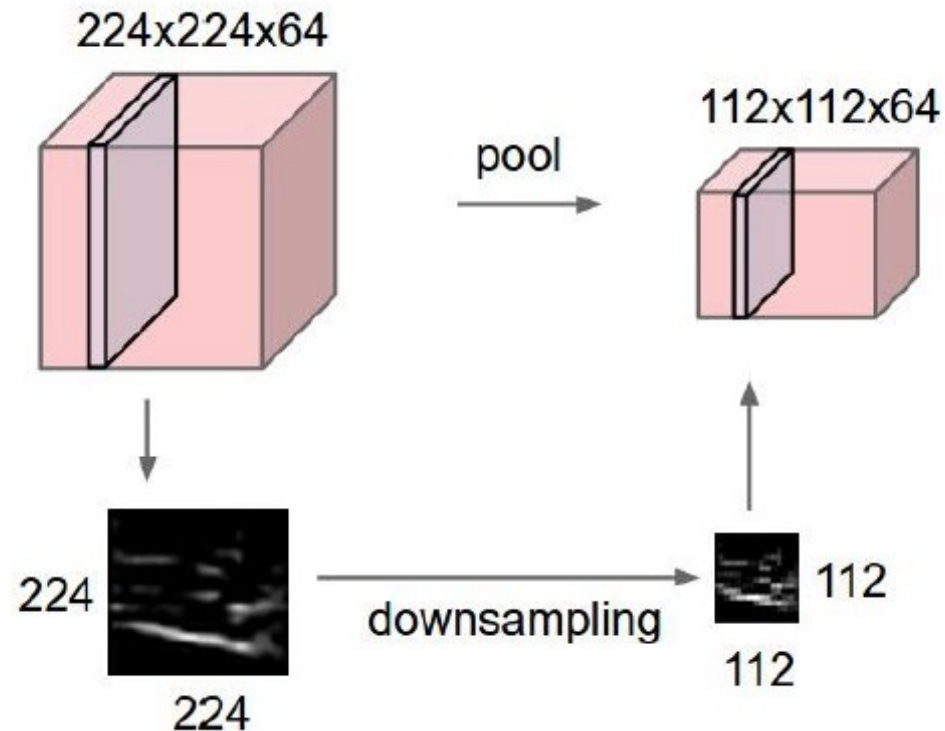


# Convolution Layer

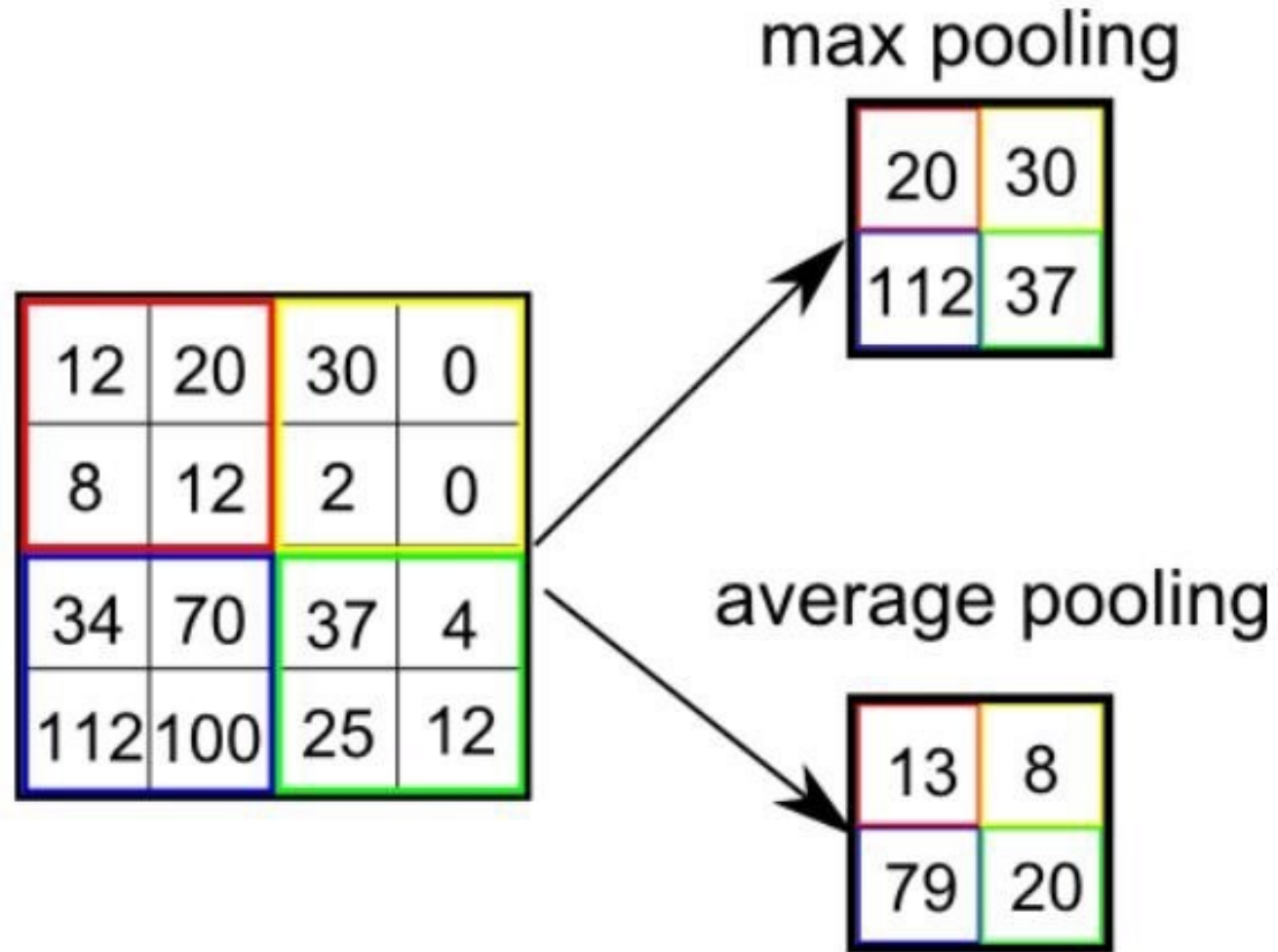


# Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



# MaxPooling and AvgPoling



# InceptionTime

---

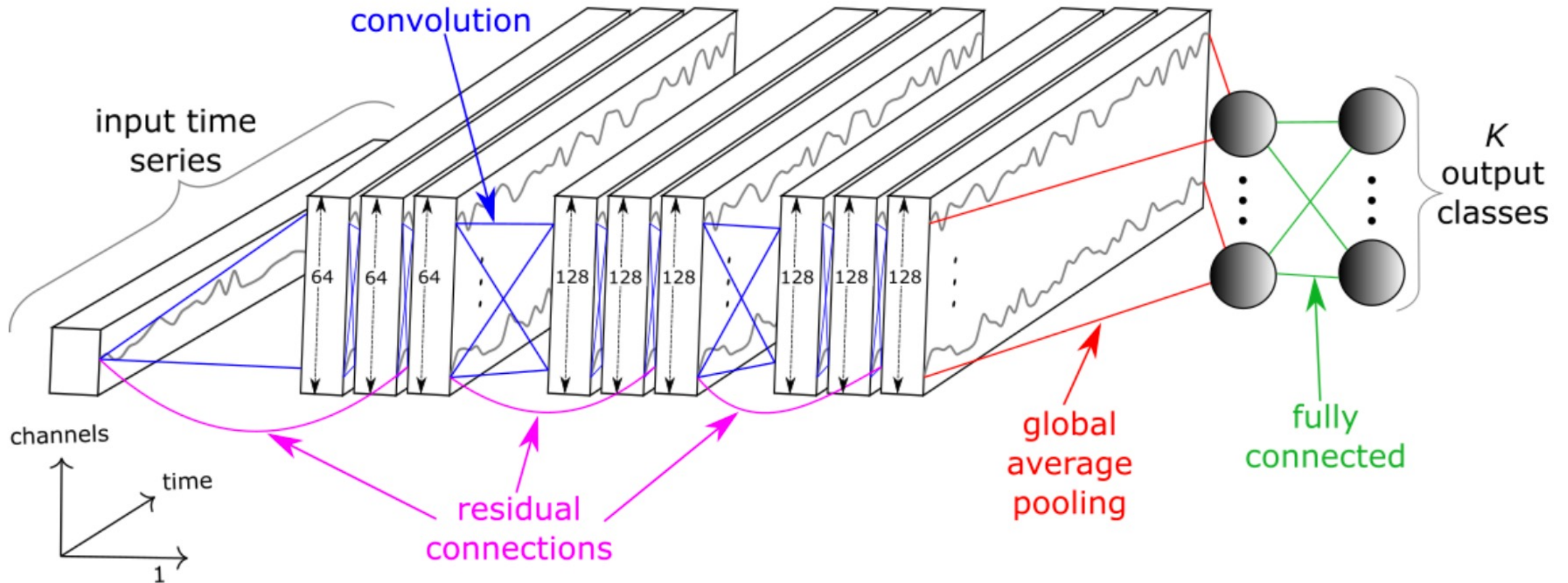
Neural network ensemble consisting of five Inception networks.

For each inception network:

- three Inception modules (6 blocks by default)
- global averaging pooling
- fully-connected layer with the softmax activation function.

Each Inception module consists of convolutions with kernels of several sizes followed by batch normalization and the rectified linear unit activation function.

# InceptionTime



# TapNet

---

Draws on the strengths of both traditional and deep learning approaches:

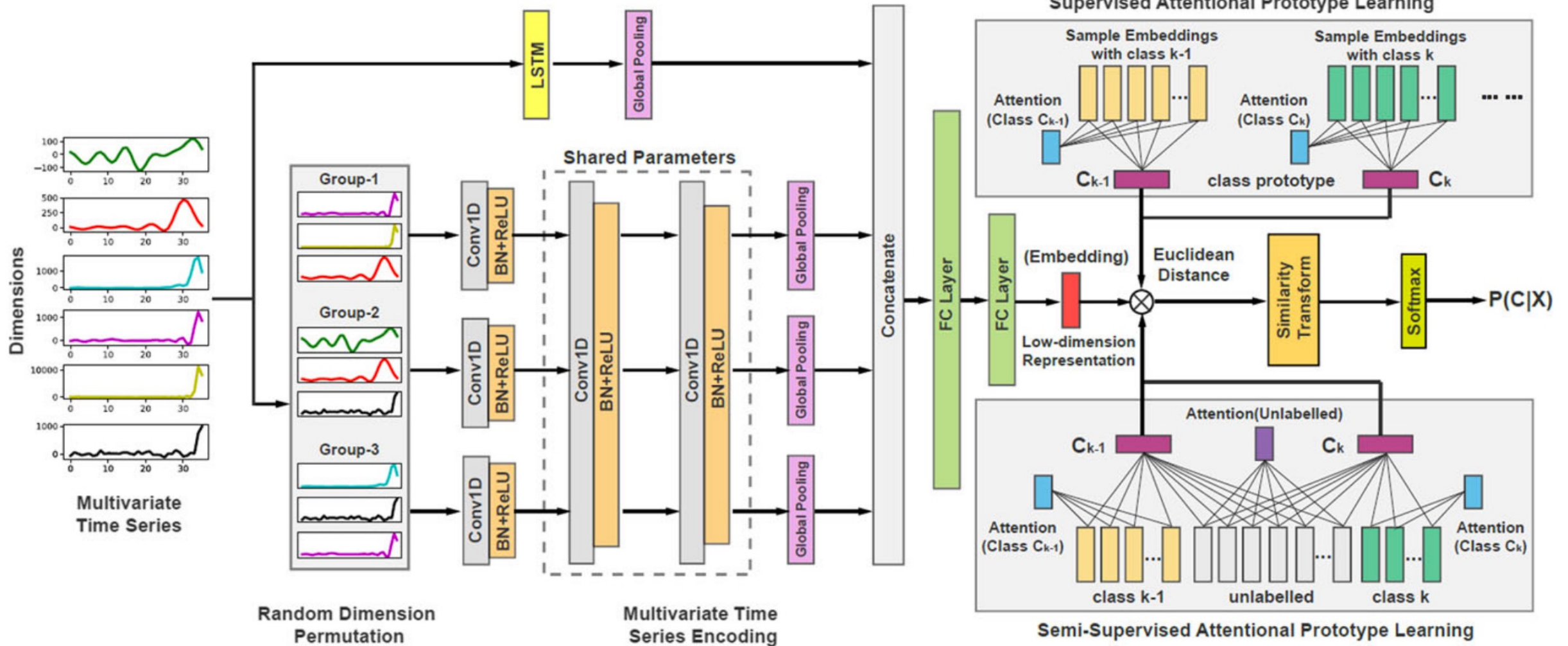
- **deep learning approaches** -> excel at learning low dimensional features without the need for embedded domain knowledge, whereas
- **traditional approaches** -> work well on small datasets.

3 distinct modules:

- Random Dimension Permutation: produce groups of randomly selected dimensions with the intention of increasing the likelihood of learning how combinations of dimension values effect class value.
- Multivariate Time Series Encoding:
  - 3 sets of 1d convolutional layers followed by batch normalisation
  - the raw data is also passed through an LSTM and global pooling layer
- Attentional Prototype Learning: used for unlabelled data



# TapNet



# Canonical Interval Forest (CIF)

---

Ensemble of time series tree classifiers built using the 22 Canonical Time-Series Characteristics (Catch22) features and simple summary statistics (mean, stdev, slope).

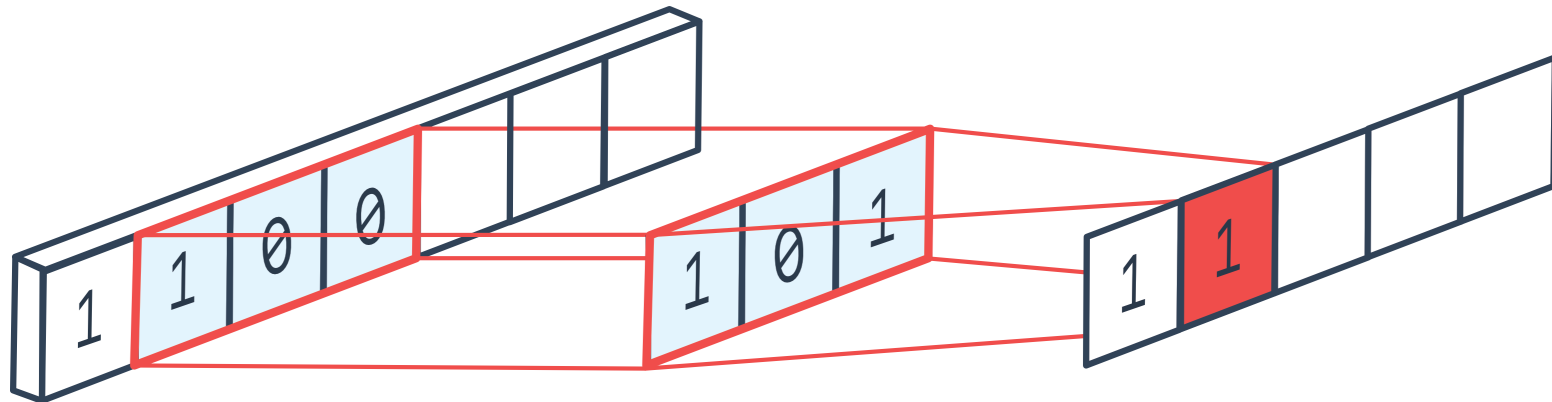
For each tree, CIF:

- samples  $k$  time series intervals of random position and length;
- subsamples 8 of the 25 features randomly;
- calculates the features for each interval, concatenates them to form a new data set;
- builds a decision tree on the feature-transformed dataset.

# ROCKET

ROCKET (Random Convolutional Kernel Transform) uses a large number of random convolutional kernels to transform the time series:

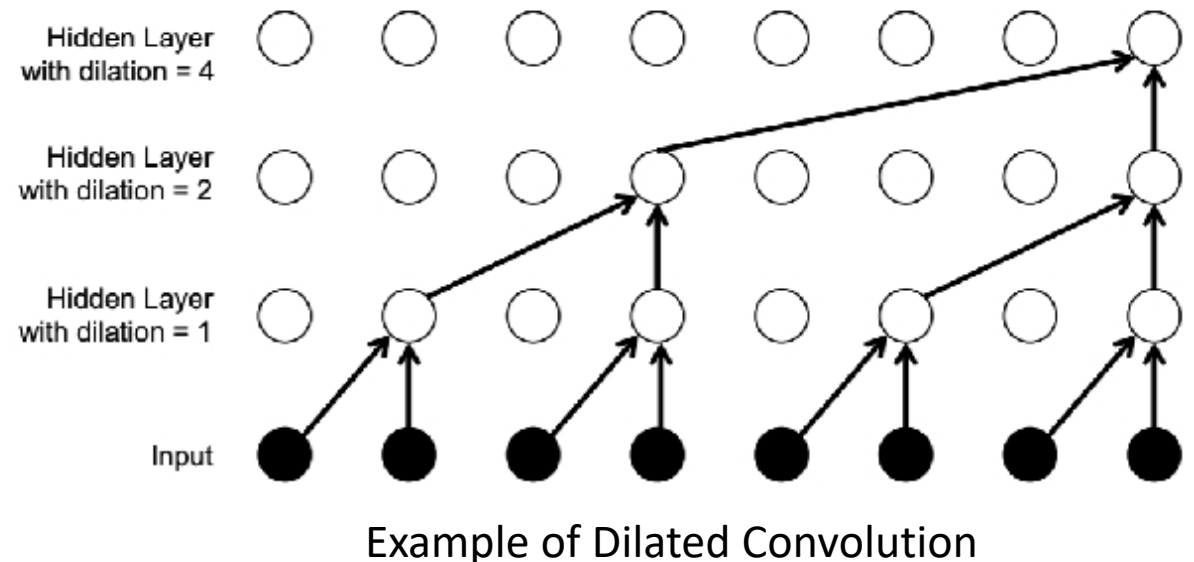
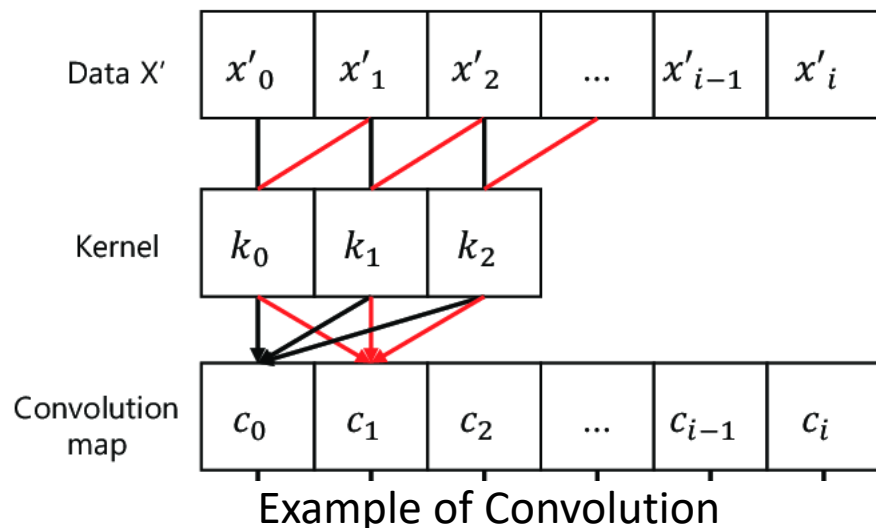
- all the parameters of all the kernels are randomly generated from fixed distributions;
- the transformed features are used to train a linear classifier (Logistic Regression or Ridge Regression Classifier);
- the combination of Rocket and logistic regression forms a single-layer convolution with random kernel weights with a trained softmax layer.



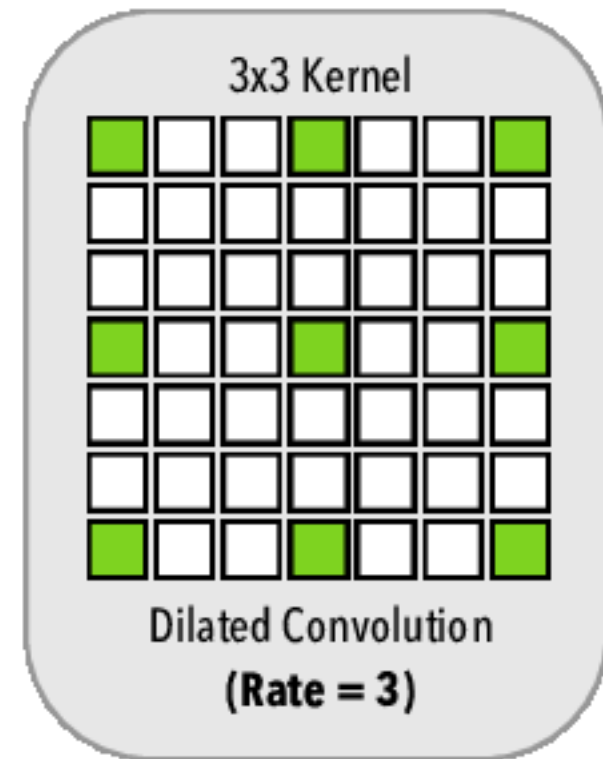
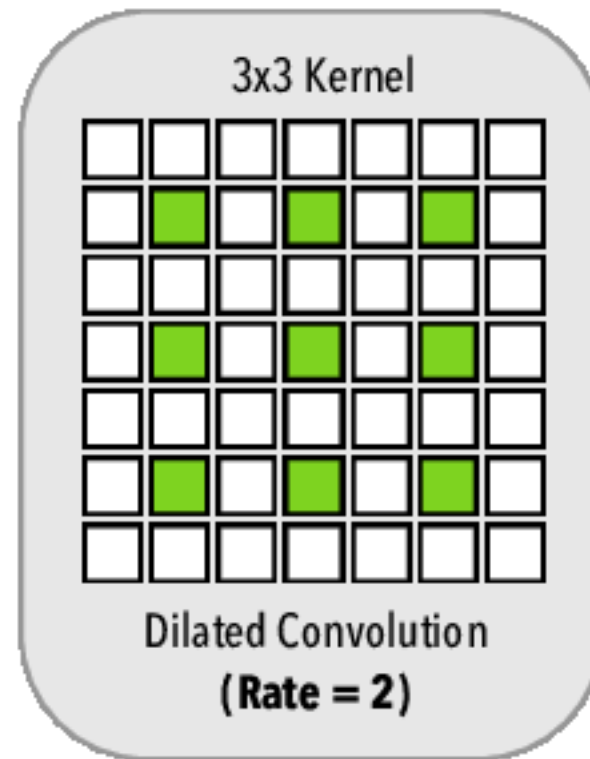
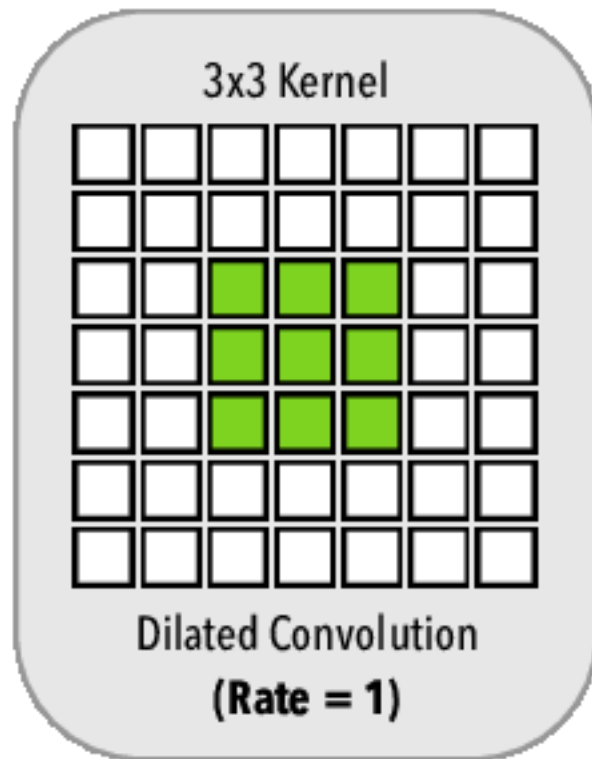
# ROCKET vs. CNN

CNNs use trainable filters/kernels optimized by stochastic gradient descent to find patterns in the input data. Rocket differs in the following ways:

- Only a single layer containing a very large number of random kernels.
- Variety of kernels: each kernel has random length, dilation, and padding, weights and biases.



# Dilated Convolution Kernels



# ROCKET vs. CNN

---

- In CNNs kernel dilation increases exponentially with depth. Rocket sample dilation randomly for each kernel, capturing patterns at different frequencies and scales.
- Rocket uses the maximum value of the resulting feature maps (~global max pooling), and the proportion of positive values (proportion of the input which matches a given pattern).
- The only hyperparameter for Rocket is the number of kernels,  $k$ .
  - $k$  handles the trade-off between classification accuracy and computation time

# MINIROCKET

---

MiniRocket removes almost all randomness from Rocket, and dramatically speeds up the transform.

- Length: uses kernels of length 9.
- Weights: restricted to two values,  $\alpha = -1$  and  $\beta = 2$ .
- Kernels: there are 512 possible two-valued kernels of length 9. Only subset of 84 is used.
- Bias: drawn from the quantiles of the convolution output for the entire training set (rather than a single, randomly-selected training example)
- Dilation: Each kernel is assigned the same fixed set of dilations, adjusted to the length of the input time series. The maximum number of dilations per kernel is 32
- Padding: half the kernel/dilation combinations use padding, and half do not.
- Features: only proportion of positive values.

# COTE / HIVE-COTE / TS-CHIEF

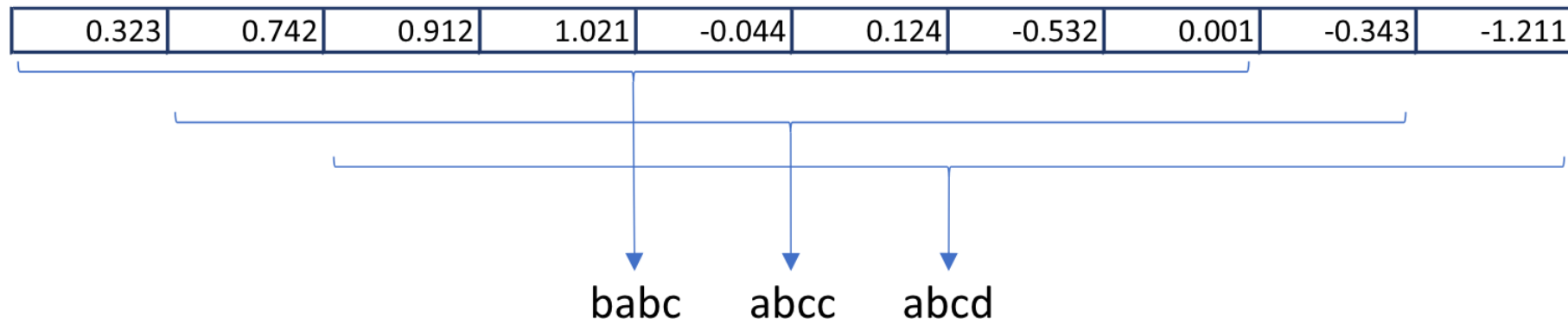
---

- Collective of Transformation-Based Ensembles (COTE) combines 35 classifiers over four data representations (similarity measures, shapelet-transform, autocorrelation features, power spectrum).
- Hierarchical Vote Collective of Transformation-Based Ensembles (HIVE-COTE) is an extension of COTE including more classifiers and a hierarchical voting procedure.
- Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF) builds a random forest of decision trees whose splitting functions are time series specific and based on similarity measures, dictionary (bag-of-words) representations, and interval-based transformations.

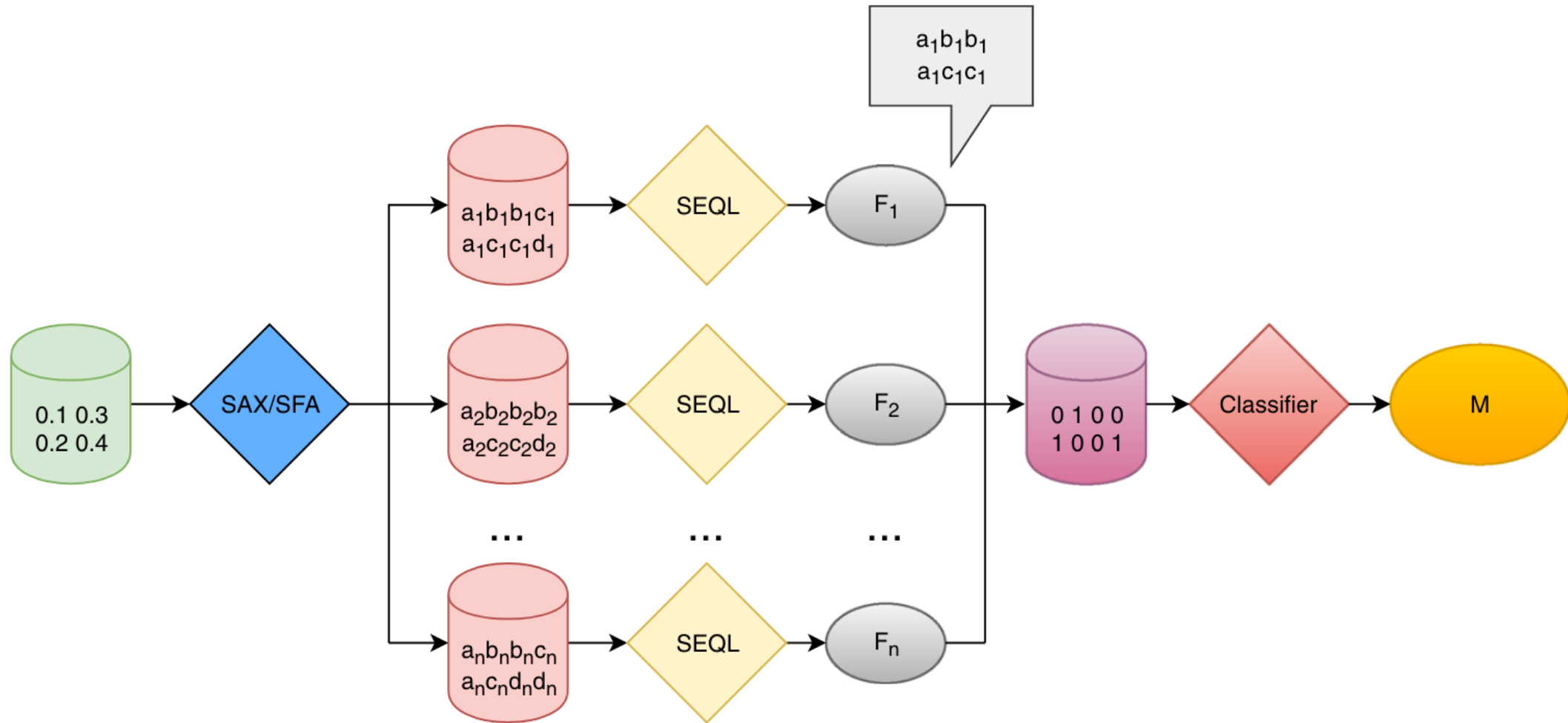


# MR-SEQL

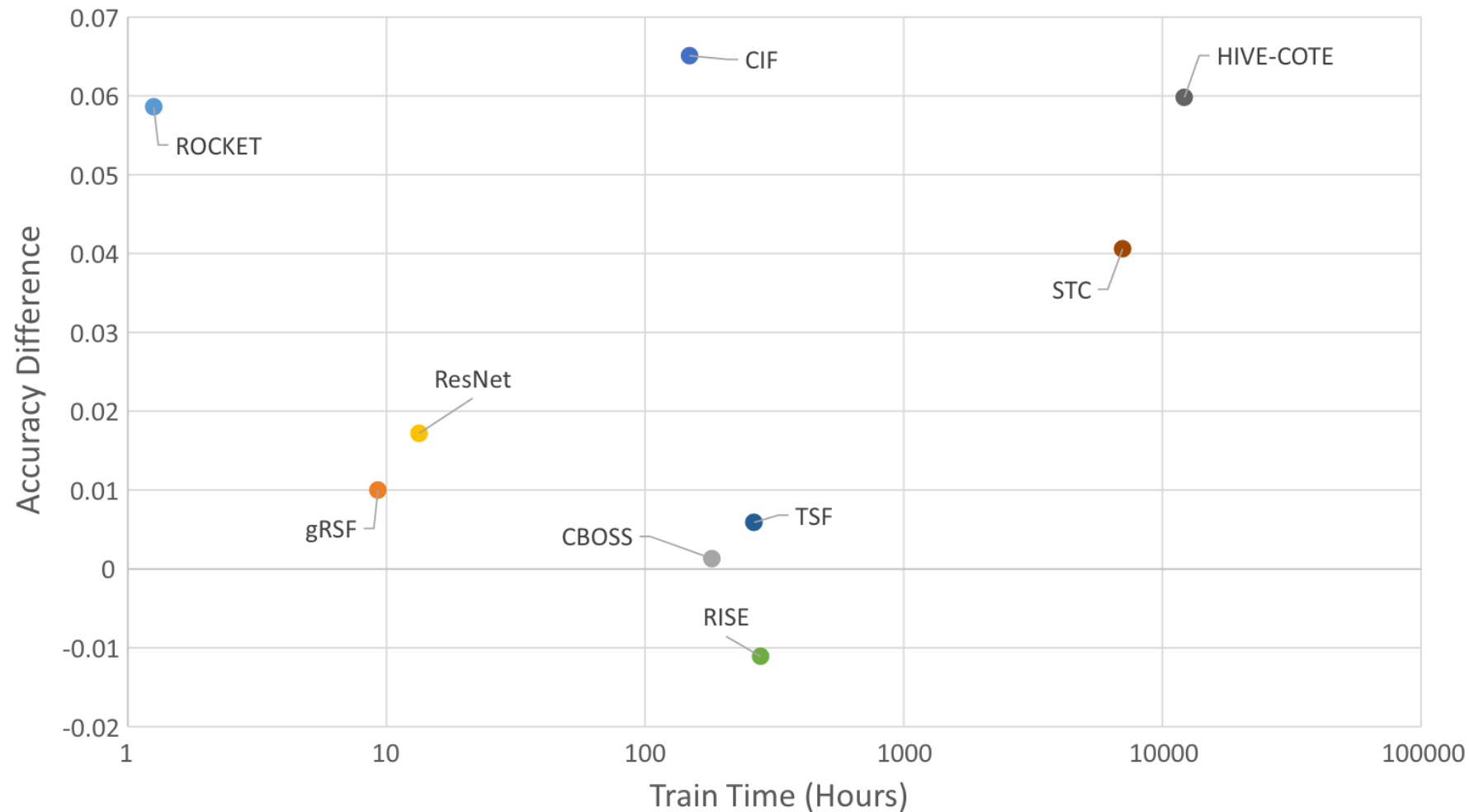
- The data is discretized into sequences of words via either Symbolic Aggregate Approximation (SAX) or SFA, using a sliding window.
- The most discriminative symbols are extracted using a SEQUENCE Learner algorithm.
- The dataset is transformed in presence/absence of subsequences (similar to a shapelet transform)
- A linear (interpretable) model is trained on this new representation



# MR-SEQL

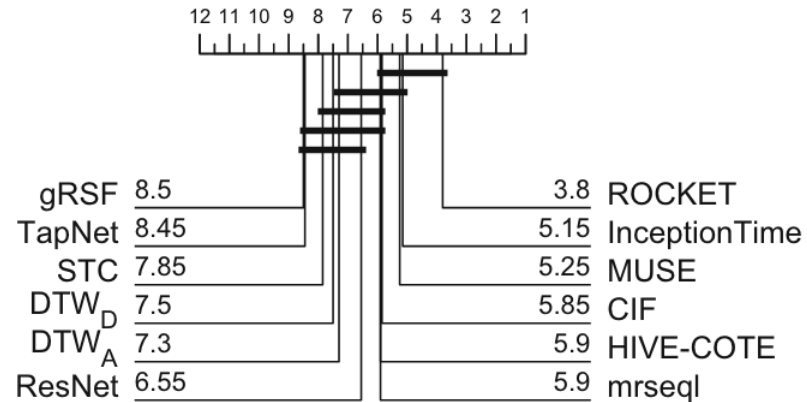


# Ranking Multivariate TSC algorithms

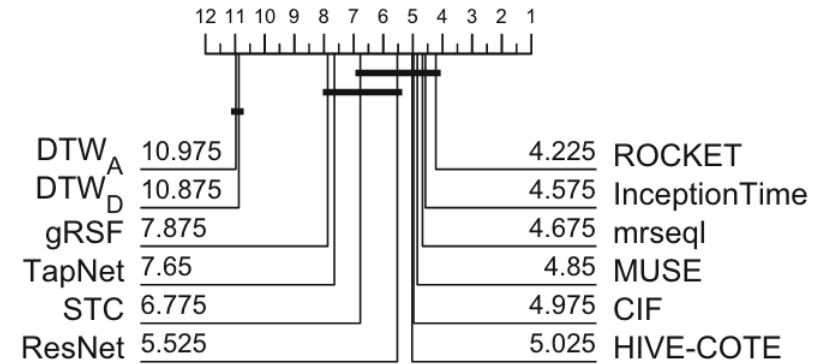


**Fig. 10** Average difference in accuracy to  $DTW_D$  versus train time for 9 MTSC algorithms

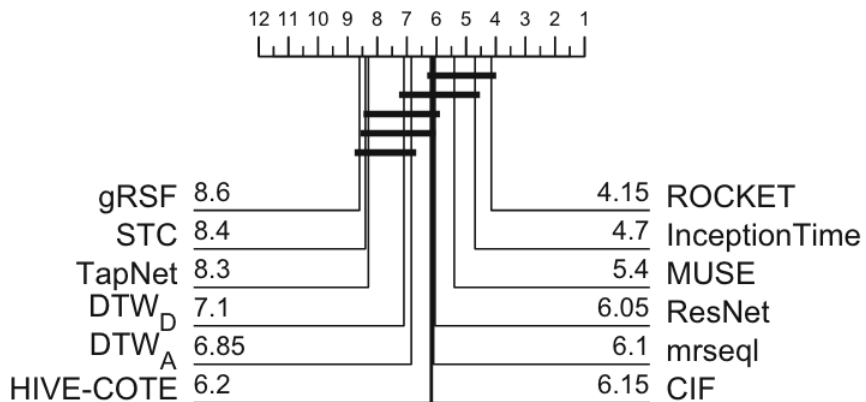
# Ranking Multivariate TSC algorithms



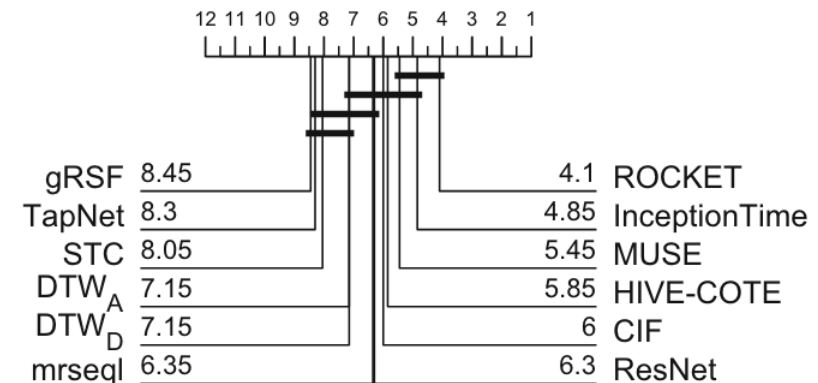
(a) Accuracy



(b) AUROC



(c) Balanced Accuracy



(d) F1

# References

---

- [1] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, “TS-CHIEF: a scalable and accurate forest algorithm for time series classification,” *Data Min Knowl Disc*, vol. 34, no. 3, pp. 742–775, May 2020, doi: 10.1007/s10618-020-00679-8.
- [2] A. Bagnall, M. Flynn, J. Large, J. Lines, and M. Middlehurst, “On the Usage and Performance of the Hierarchical Vote Collective of Transformation-Based Ensembles Version 1.0 (HIVE-COTE v1.0),” in *Advanced Analytics and Learning on Temporal Data*, vol. 12588, V. Lemaire, S. Malinowski, A. Bagnall, T. Guyet, R. Tavenard, and G. Ifrim, Eds. Cham: Springer International Publishing, 2020, pp. 3–18. doi: 10.1007/978-3-030-65742-0\_1.
- [3] T. Le Nguyen, S. Gsponer, I. Ilie, M. O’Reilly, and G. Ifrim, “Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations,” *Data Min Knowl Disc*, vol. 33, no. 4, pp. 1183–1222, Jul. 2019, doi: 10.1007/s10618-019-00633-3.
- [4] Z. Wang, W. Yan, and T. Oates, “Time series classification from scratch with deep neural networks: A strong baseline,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 1578–1585. doi: 10.1109/IJCNN.2017.7966039.
- [5] M. Middlehurst, J. Large, and A. Bagnall, “The Canonical Interval Forest (CIF) Classifier for Time Series Classification,” in *2020 IEEE International Conference on Big Data (Big Data)*, Dec. 2020, pp. 188–195. doi: 10.1109/BigData50022.2020.9378424.
- [6] A. Dempster, D. F. Schmidt, and G. I. Webb, “MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification,” *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 248–257, Aug. 2021, doi: 10.1145/3447548.3467231.
- [7] A. Dempster, F. Petitjean, and G. I. Webb, “ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels,” *Data Min Knowl Disc*, vol. 34, no. 5, pp. 1454–1495, Sep. 2020, doi: 10.1007/s10618-020-00701-z.
- [8] J. Faouzi, “Time Series Classification: A review of Algorithms and Implementations,” *Machine Learning*, p. 35.
- [9] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, “The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” *Data Min Knowl Disc*, vol. 35, no. 2, pp. 401–449, Mar. 2021, doi: 10.1007/s10618-020-00727-3.
- [10] H. I. Fawaz *et al.*, “InceptionTime: Finding AlexNet for Time Series Classification,” *Data Min Knowl Disc*, vol. 34, no. 6, pp. 1936–1962, Nov. 2020, doi: 10.1007/s10618-020-00710-y.