8-11-2020

# Time Series Mining: Shapelet Discovery, Ensembling, and Applications

Soukaina Filali Boubrahimi
*Georgia State University*

TIME SERIES MINING: SHAPELET DISCOVERY, ENSEMBLING, AND APPLICATIONS

by

SOUKAINA FILALI BOUBRAHIMI

Under the Direction of Rafal Angryk, PhD

ABSTRACT

Time series is a prominent class of temporal data sequences that has the properties of being equally spaced in time, chronologically ordered, and highly dimensional. Time series classification is an important branch of time series mining. Existing time series classifiers operate either on row data in the time domain or into an alternate data space in the shapelets or frequency domains. Combining time series classifiers, is another powerful technique used to improve the classification accuracy. It was demonstrated that different classifiers can be expert in predicting different subset of classes over others. The challenge lies in learning the expertise of different base learners. In addition, the high dimensionality characteristic of time series data makes it difficult to visualize their distribution. In this thesis we developed a new time series ensembling methods in order to improve the predictive performance, investigated the interpretability of classifiers by leveraging the power of deep learning models and adjusting them to provide visual shapelets as a by-product of the classification task. Finally, we show application through problems of solar energetic particle events prediction.

INDEX WORDS: Data Mining, Time Series, Shapelet Learning

# TIME SERIES MINING: SHAPELET DISCOVERY, ENSEMBLING, AND APPLICATIONS

by

SOUKAINA FILALI BOUBRAHIMI

A Dissertation Submitted in Partial Fulfillment for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2020

TIME SERIES MINING: SHAPELET DISCOVERY, ENSEMBLING, AND APPLICATIONS

by

SOUKAINA FILALI BOUBRAHIMI

Committee Chair: Rafal Angryk

Committee: Yingshu Li
Rajshekhar Sunderraman
Petrus Martens

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
August 2020

## DEDICATION

*To my lovely parents Driss and Samira, brother Yahya, and grandparents (Abdelmounim, Noufissa and Elhajja). Your love, prayers, and pride have been invaluable to me all my life.*

## ACKNOWLEDGEMENTS

The work presented in this dissertation was made possible with the support of many people. First, I wish to express my genuine gratitude to the best advisor one could ask for, Dr. Rafal Angryk. Dr. Angryk's confidence in my abilities, his encouragement, support during tough situations, and the push he provided helped me tremendously in the completion of this goal. I also owe a deep thank you to Dr. Piet Martens for his research guidance and motivational enthusiasm towards science.

I would also like to thank each of my committee members – Dr. Rajshekhar Sunderraman, Dr. Piet Martens, and Dr. Yingshu Li for their valuable feedback and guidance in my graduate studies.

I appreciate all the discussions, feedback, assistance, and friendship from the members of our lab over the years–Ahmet, Berkay, Dustin, Hamdi, Manolis, Max, Mike, Ruizhe, Soumitra, and Sushant. Finally, to all my close friends: Hamdi, Hind, Irina, Sara, and Badrinath, thank you.

Last, but surely not least, I would like to take a moment to thank my family members. Thank you dad for being my mentor and always trusting my potential since a very young age. Thank you mom for sharing your words of wisdom that guided me all the way and being my life coach. Thank you Yahya for always bringing joy in life and to my grandparents (Abdelmounim, Noufissa and Elhajja) for always keeping me in your prayers. I am honored to have you all in my life.

## FUNDING ACKNOWLEDGEMENT

This project has been supported in part by funding from the Division of Advanced Cyberinfrastructure within the Directorate for Computer and Information Science and Engineering, the Division of Astronomical Sciences within the Directorate for Mathematical and Physical Sciences, and the Division of Atmospheric and Geospace Sciences within the Directorate for Geosciences, under NSF awards 1443061, 1812964, 1936361 and 1931555. It was also supported in part by funding from NASA, through the Heliophysics' Living With a Star Science Program, under NASA award NNX15AF39G, as well as through the direct contract from Space Radiation Analysis Group (SRAG). In addition to that, the work has been in part sponsored by state funding from Georgia State University's Second Century Initiative, and Next Generation Program.

# TABLE OF CONTENTS

# List of Tables

# List of Figures

# CHAPTER 1

# Introduction

Time series data has always been a major object of interest in financial market price biding, meteorology, entertainment, and virtually every other field of human endeavor. With the rise of the volume and velocity of sensors, weblogs, programmatic advertising, online and offline transaction data (usually recorded in equispaced time intervals), it is of great interest to understand the potentials of time series mining algorithms. The purpose of time series data mining is to extract meaningful knowledge from the shape of the data. One of the most prominent tasks of time series mining is supervised learning on time series data. The idea behind supervised learning is to find a mapping function that links the input feature space with the output labels/classes. The main challenge of the supervised learning task on time series data is the curse of dimensionality which makes the model and data interpretability limited. Depending on the application, it is substantial to achieve good performance levels in the metrics of interest without totally compromising the model explainability. A typical way of compromising interpretability is by making use of black-box models (such as deep learning models) that do not provide any insights about the causes of the classification decision. Although, some argue that to alleviate the lack of interpretability problem, the use of latent features produced by black-box models at the end of the training in post-hoc analysis could be a solution. The latent features constitute uninterpretable knowledge too, even if when fed into transparent classifiers. Another widespread idea is that the more complex is the model, the less interpretable it becomes, which is not particularly true. Figure 1.1 illustrates a fictional negative correlation result of a study of interpretability and performance of

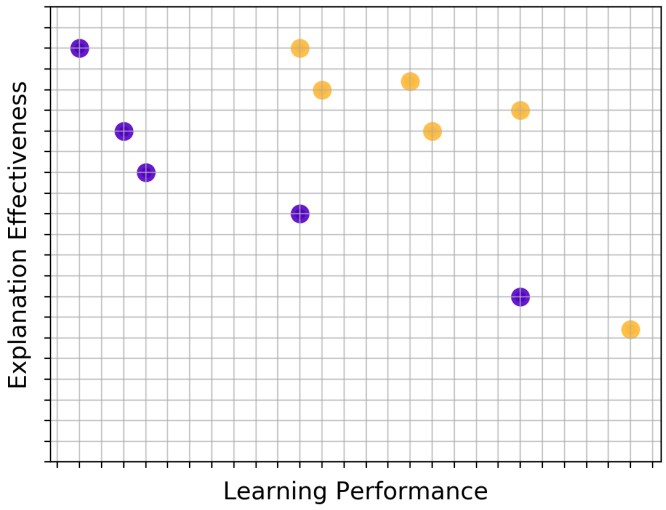



**Figure 1.1:** Fictional "accuracy-interpretability trade-off," taken from the DARPA XAI (Explainable Artificial Intelligence) Broad Agency Announcement [1]

different models, represented in scatter points. The figure is not real and the trade-off is atypical in data science applications with meaningful features [2]. With the emerging "right to explanation" data protection law, [3] it is imperative now more than ever to shed the light on explainability. In this chapter, we talk about the motivation behind our work, as well as the contribution and outline of this dissertation.

## 1.1 Motivation

The main time series machine learning tasks, as shown in Figure 1.2, includes: unsupervised learning(clustering) [4–6], supervised learning(classification) [7–12], rule discovery [13, 14], anomaly detection [15–17], and query by content [18–20]. Figure 1.3 zooms in time series classification task. Time series classification can be defined as assigning one of two or many predefined classed to an unlabeled time series Q [21]. In the literature [22], time series classifiers are grouped into 7 categories: Time-domain, shapelet, dictionary, interval, differential, deep learning and ensembling. In this thesis we worked

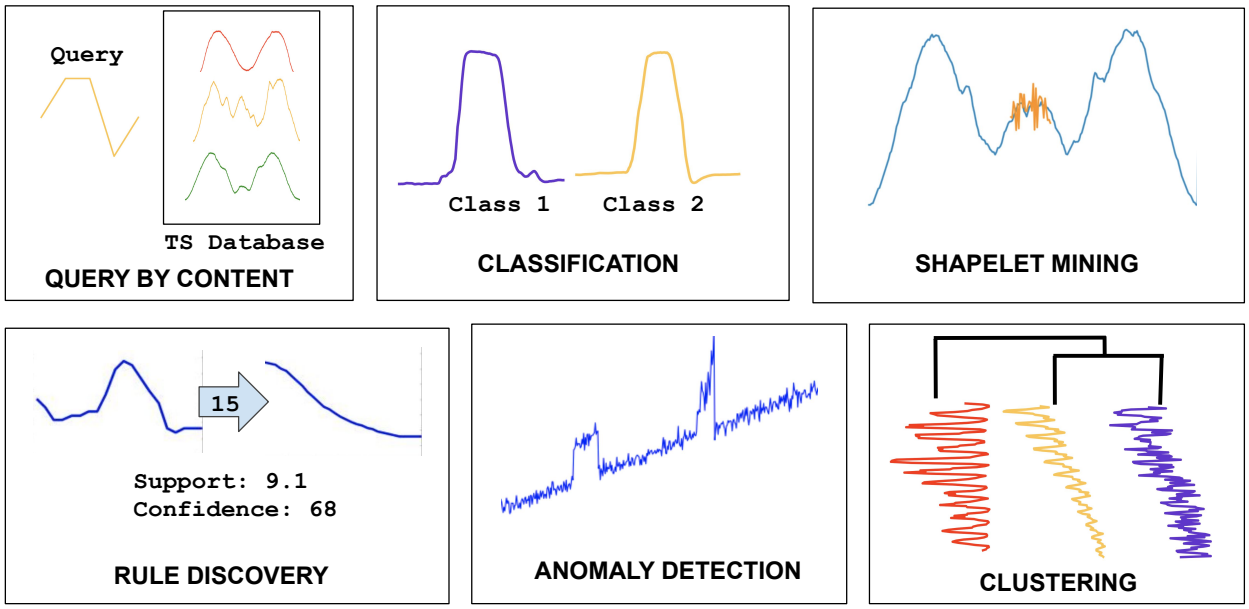



**Figure 1.2:** Time series data mining tasks and classification task sub-categories

on the time domain, shapelet, deep learning and ensembling classifiers. According to the No Free Lunch (NFL) theorem, it is generally accepted that there is no one model that dominates over all types learning problems [23]. In addition, different classifiers may be useful in representing different aspects of the problem. In other terms, while some classifiers might be powerful in classifying a given class c, other classifiers might be less accurate in classifying c but more useful when classifying other classes. One way of optimizing classifiers' accuracy is to combine them using a meaningful ensemble strategy. Another important dimension of the classification task is interpretability. Depending on the application domain, it is particularly challenging to convince the community to adopt an accurate black-box model that is lacking in terms of explainability [24]. Therefore, we are motivated to design a classification model, which will shed light on interpretability.

## 1.2 Contributions

The contributions of this thesis are listed below:

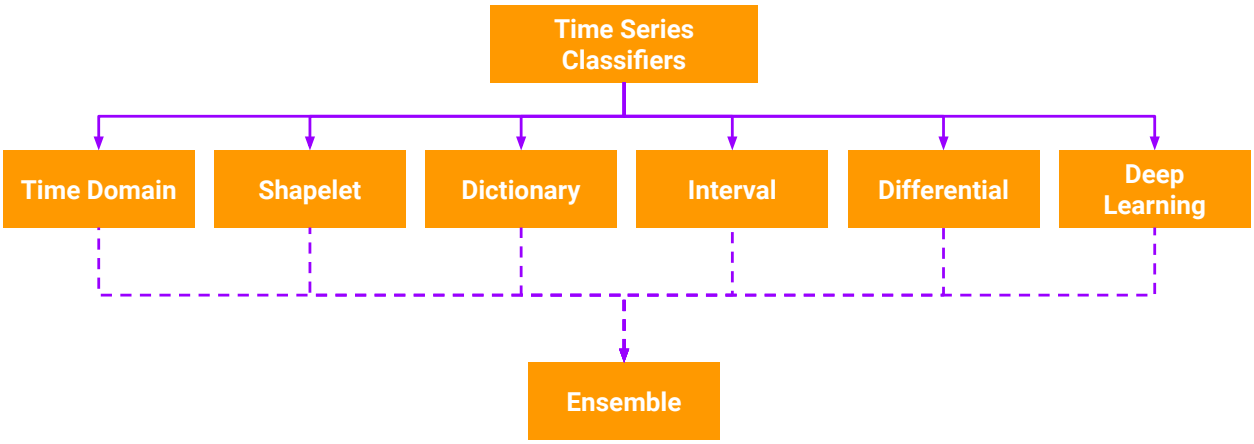



**Figure 1.3:** Time series classifiers sub-categories

1. We present three flavors of a new one-dimensional convolutional neural network (1DCNN) model for time series classification with interpretable convolutional kernels that represents mined shapelets.

2. We propose network pruning as a strategy to solely mine the most prominent various lengths shapelets. The model sparsity level (number of shapelets) is defined in a data-driven fashion following the learning curves of the model.

3. We present a new ensemble model *Neuro-Ensemble* [25, 26], that improves the predictive performance of the participating base classifiers based on ensemble stacking/meta-learning paradigm.

4. We demonstrate the use of Vector Auto-Regression (VAR) model for high dimensional time series data representation prior to the prediction task [27].

## 1.3 Outline

This thesis is organized as follows. In Chapter 2, the background on time series classification, ensembling methodologies, and deep learning methods for time series classification are introduced. This serves as the basis of further discussion of our work regarding

our proposed models. In Chapter 3, we present our proposed deep learning model for time series classification based on one-dimensional kernels. By using one-dimensional filters, we are able to add an interpretability edge that enables us to visually inspect the short-length patterns that triggered the classification decision. Then in Chapter 4 we discuss Neuro-Ensemble, a new ensemble model based on the hypothesis that some learners are more expert in predicting a subset of classes over other base learners. The idea is to leverage the strength of the base learners to optimize the final classifier accuracy. Chapter 5 introduces an application of time series mining in the prediction of solar energetic particles (SEP) events using a Vector AutoRegression (VAR) model for data transformation and decision trees for classification. Lastly, in Chapters 6 and 7, we summarize the thesis, provide future work directions and present the appendices respectively.

# CHAPTER 2

# Background

## 2.1 Time Series Data

A time sequence is a series of successive discrete-time data points that are indexed in time order. A time series is a time sequence with the additional property of equally-spaced time data points. A time sequence that violates the equispaced time property of a time series should either be interpolated or increased in granularity. Interpolation consists of filling the missing data by estimating new artificial data points within the range of the discrete set of known data points. Increasing granularity consists of dropping out some data points from the time sequence to have an equal cadence in time. Time series data are frequently used in any domain which involves temporal measurements such as signal processing, weather forecasting, earthquake prediction, and pandemic spread estimation. An example of time series data is shown in Figure 2.1 that illustrates the number of SARS-CoV-2 (severe acute respiratory syndrome coronavirus 2) confirmed positive cases in China, Germany, and Spain. An example of time sequences (and not time series) is illustrated in Figure 2.1 through the number of confirmed positive cases in the US and Italy (data source: Johns Hopkins University [1]).

1 https://coronavirus.jhu.edu/map.html

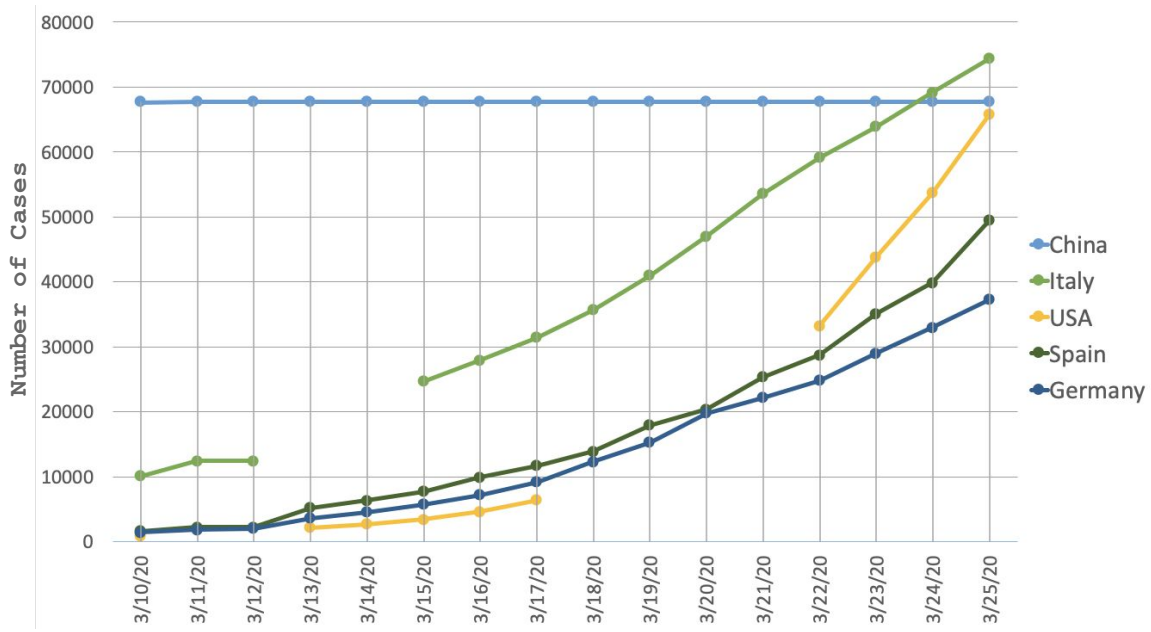



**Figure 2.1:** Time series and time sequences of number of SARS-CoV-2 daily confirmed cases

## 2.2 Datasets

In this section, we introduce three datasets associated with our experiments. The UCR Archive [28] is a general archive, containing distinct datasets from various fields whereas the SEP dataset which is more domain-specific. This shows the wide applicability of our methods.

### 2.2.1 *UCR Archive*

Most of our univariate work was initially tested using the UCR time series dataset archive [28]. The UCR archive includes synthetic data and real-world data from various domains, ranging in class numbers as well as sizes. Each dataset consists of training and testing data. The specific data processing will be mentioned with each corresponding experiments. Due to the size, the running time, and the fact that there will always be

**Table 2.1:** Characteristics of the datasets used in the experiments

| ID | Dataset | Instances | Features | $\mathcal{L}$ | Continuous | Discontinuous |
|---|---|---|---|---|---|---|
| 01 | glass | 213 | 9 | 6 | 9 | 0 |
| 02 | ecoli | 331 | 7 | 6 | 7 | 0 |
| 03 | voting-records | 435 | 16 | 106 | 0 | 16 |
| 04 | balance-scale | 625 | 4 | 3 | 4 | 0 |
| 05 | soybean | 630 | 35 | 15 | 0 | 35 |
| 06 | Australia | 689 | 14 | 2 | 14 | 0 |
| 07 | Diabetes | 767 | 8 | 2 | 8 | 0 |
| 08 | tic-tac-toe | 957 | 9 | 2 | 0 | 9 |
| 09 | vowel-context | 990 | 13 | 11 | 13 | 0 |
| 10 | ImageSeg | 2310 | 19 | 7 | 19 | 0 |
| 11 | LED-7 | 3199 | 7 | 10 | 7 | 0 |
| 12 | usps | 9297 | 256 | 10 | 256 | 0 |
| 13 | Epilepsy | 11499 | 178 | 5 | 178 | 0 |
| 14 | Nursery | 12960 | 9 | 4 | 0 | 9 |

more datasets that could be tested, not all datasets from this archive will be included in our experiments.

### 2.2.2 *UCI Archive*

The performance of the different ensemble methodologies are evaluated on 14 datasets from the UCI Machine Learning repository [29]. Table 2.1 presents the details of the datasets used in this study (dataset name, dataset size, number of attributes, number of classes, number of continuous attributes and number of discrete attributes).

### 2.2.3 *Solar Energetic Particles (SEP) Data*

Our dataset is composed of multivariate time series of X-ray, integral and differential proton flux and fluences spectra that were measured onboard of Space Environment

Monitor (SEM) instruments package of the Geostationary Operational Earth Satellites (GOES). In particular, we consider both the short and long X-ray channel data recorded by the X-ray Sensor (XRS). For the proton channels, we consider channels P6 and P7 recorded by the Energetic Particle Sensor (EPS) and proton channels P8, P9, P10, and P11 recorded by the High Energy Proton and Alpha Detector (HEPAD). Table 2.2 summarizes the instruments onboard the GOES satellites and their corresponding data channels that we used.

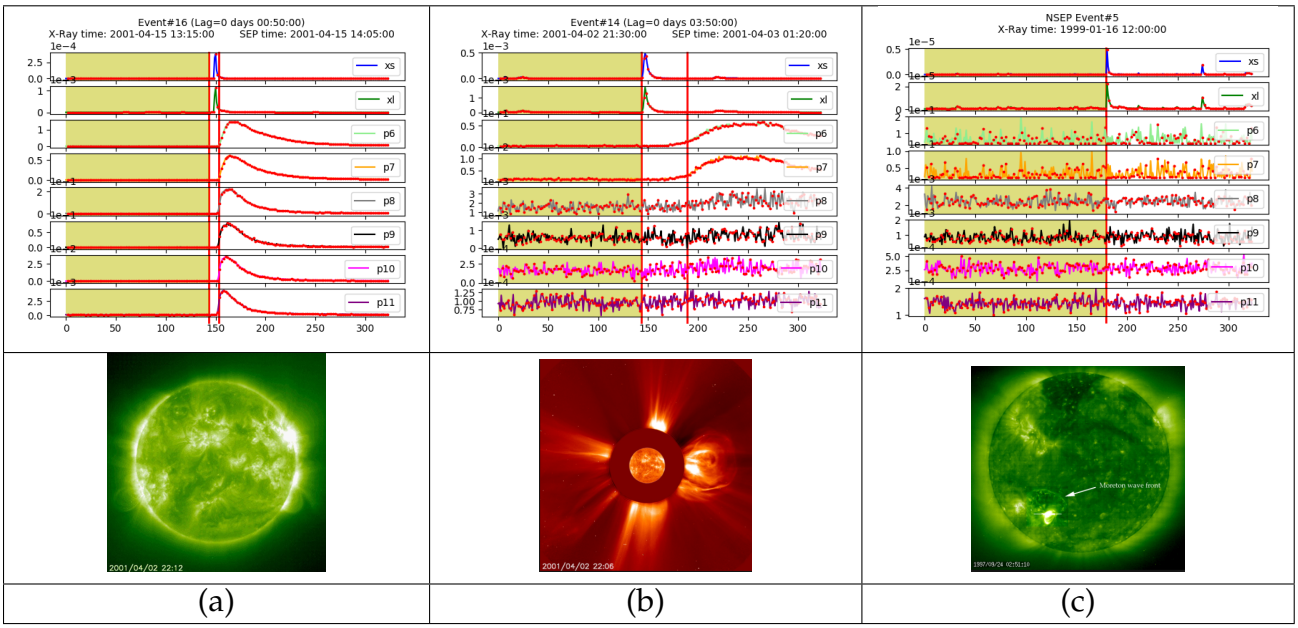



**Figure 2.2:** Example of an (a) Impulsive SEP event that started on the 2001-04-15 14:05:00 as a result of a flare that occurred in the 2011-04-15 13:15:00 shown in the SOHO EIT instument and a (b) gradual SEP event whose nearest temporal flare happened on 2001-04-02 21:30:03A and occurred as a result of a CME on the 2001-04-02 22:06:07 shown in the SOHO LASCO instrument and a (c) a flare that happened on 1999-01-16 12:00:00 that did not lead to any $>$ 100 MeV SEP event shown in the SOHO EIT instrument.

The data we collected is made publically available by NOAA in the following link: (*https://satdat.ngdc.noaa.gov/sem/goes/data/new_avg/*).The data is available in three different cadences. The full resolution data is captured every three seconds from the GOES satellites, which is aggregated and made available with one and five-minute cadences. In this paper we use the aggregated five-minute data which is the one usually cited in the

**Table 2.2:** GOES X-ray and Proton instuments and Channels.

| Instrument | Channels | Description |
|---|---|---|
| XRS | xs | Short wavelength channel irradiance (0.5 - 0.3 nm) |
| | xl | Long wavelength channel irradiance (0.1-0.8 nm) |
| HEPAD | p8_flux | Proton Channel 350.0 - 420.0 MeV |
| | p9_flux | Proton Channel 420.0 - 510.0 MeV |
| | p10_flux | Proton Channel 510.0 - 700.0 MeV |
| | p11_flux | Proton Channel > 700.0 MeV |
| EPS | p6_flux | Proton Channel 80.0 - 165.0 MeV |
| | p7_flux | Proton Channel 165.0 - 500.0 MeV |

literature [30] [31] [32]. In most cases, there are a couple a co-existing GOES satellites whose data is captured by more than one GOES satellite at a time. In this study, we always consider the data reported by the primary GOES satellite that is designated by the NOAA.

Only a portion of the collected data is used to train and test our classifier. The positive class in this study is composed of X-Ray and proton channels time series that led to >100 MeV SEP impulsive or gradual events. On the other hand, the negative class is composed of X-Ray and proton channels time series that did not lead to any >100 MeV SEP events. An example of these events are shown in Figure 2.2. In order to select such events, we used a number of catalogs. For the positive class events we used the same catalog of SEP events >100 MeV in [32] that covers the events that happened between 1997 and 2013.

Our positive class is composed of the 47 X-Ray parent events of their corresponding >100 MeV SEP events that appear in [32] and shown in Table 2.3. We use the X-Ray catalog (*https://www.ngdc.noaa.gov/stp/space-weather/solar-data/solar-features/solar-flares/x-rays/goes/xrs/*) as well as the CME catalog (*https://cdaw.gsfc. nasa.gov/CME_list/*) from the SOlar Heliospheric Observatory (SOHO) to derive the parent events of the >100 MeV SEP events. There was an exception of two SEP events that happened in August and September 1998 that we believe are gradual events but could not map to any CME report due to the missing data during the SOHO mission interruption. This latter happened because of the major loss of altitude experienced by the spacecraft due to the failure to

**Table 2.3:** $>$ 100 MeV SEP Event List with their Parent Events(CME/Flare)

| SEP Event ID | Onset Time of SEP Event | Parent X-ray Event |
|---|---|---|
| 1 | 1997-11-04 05:52:00 | 1997-11-04 05:52:00 |
| 2 | 1997-11-06 11:49:00 | 1997-11-06 11:49:00 |
| 3* | 1998-04-20 09:38:00 | 1998-04-20 09:38:00 |
| 4 | 1998-05-02 13:31:00 | 1998-05-02 13:31:00 |
| 5 | 1998-05-06 07:58:00 | 1998-05-06 07:58:00 |
| 6 | 1998-08-24 21:50:00 | 1998-08-24 21:50:00 |
| 7 | 1998-09-30 13:50:00 | 1998-09-30 13:50:00 |
| 8 | 1998-11-14 05:15:00 | 1998-11-14 06:05:00 |
| 9 | 2000-06-10 16:40:00 | 2000-06-10 16:40:00 |
| 10 | 2000-07-14 10:03:00 | 2000-07-14 10:03:00 |
| 11 | 2000-11-08 22:42:00 | 2000-11-08 22:42:00 |
| 12 | 2000-11-24 14:51:00 | 2000-11-24 14:51:00 |
| 13* | 2000-11-26 16:34:00 | 2000-11-26 16:34:00 |
| 14* | 2001-04-02 21:32:00 | 2001-04-02 21:32:00 |
| 15 | 2001-04-12 09:39:00 | 2001-04-12 09:39:00 |
| 16 | 2001-04-15 13:19:00 | 2001-04-15 13:19:00 |
| 17 | 2001-04-17 21:18:00 | 2001-04-18 02:05:00 |
| 18 | 2001-08-15 12:38:00 | 2001-08-16 23:30:00 |
| 19* | 2001-09-24 09:32:00 | 2001-09-24 09:32:00 |
| 20 | 2001-11-04 16:03:00 | 2001-11-04 16:03:00 |
| 21 | 2001-11-22 22:32:00 | 2001-11-22 19:45:00 |
| 22 | 2001-12-26 04:32:00 | 2001-12-26 04:32:00 |
| 23 | 2002-04-21 00:43:00 | 2002-04-21 00:43:00 |
| 24 | 2002-08-22 01:47:00 | 2002-08-22 01:47:00 |
| 25 | 2002-08-24 00:49:00 | 2002-08-24 00:49:00 |
| 26 | 2003-10-28 09:51:00 | 2003-10-28 09:51:00 |
| 27 | 2003-11-02 17:03:00 | 2003-11-02 17:03:00 |
| 28* | 2003-11-05 02:37:00 | 2003-11-05 02:37:00 |
| 29+ | 2004-11-01 03:04:00 | 2004-11-01 03:04:00 |
| 30+ | 2004-11-10 01:59:00 | 2004-11-10 01:59:00 |
| 31+ | 2005-01-16 21:55:00 | 2005-01-17 08:00:00 |
| 32+ | 2005-01-20 06:36:00 | 2005-01-20 06:36:00 |
| 33+ | 2005-06-16 20:01:00 | 2005-06-16 20:01:00 |
| 34+* | 2005-09-07 17:17:00 | 2005-09-07 17:17:00 |
| 35+* | 2006-12-06 18:29:00 | 2006-12-06 18:29:00 |
| 36+ | 2006-12-13 02:14:00 | 2006-12-13 02:14:00 |
| 37+ | 2006-12-14 21:07:00 | 2006-12-14 21:07:00 |
| 38 | 2011-06-07 06:16:00 | 2011-06-07 06:16:00 |
| 39 | 2011-08-04 03:41:00 | 2011-08-04 03:41:00 |
| 40 | 2011-08-09 07:48:00 | 2011-08-09 07:48:00 |
| 41 | 2012-01-23 03:38:00 | 2012-01-23 03:38:00 |
| 42* | 2012-01-27 17:37:00 | 2012-01-27 17:37:00 |
| 43 | 2012-03-07 01:05:00 | 2012-03-07 01:05:00 |
| 44 | 2012-03-13 17:12:00 | 2012-03-13 17:12:00 |
| 45 | 2012-05-17 01:25:00 | 2012-05-17 01:25:00 |
| 46* | 2013-04-11 06:55:00 | 2013-04-11 06:55:00 |
| 47 | 2013-05-22 13:08:00 | 2013-05-22 13:08:00 |

* Gradual Events.

+ Missing Data in P6 and P7.

**Table 2.4:** Non SEP Event List

| Non SEP Event ID | X-ray Event | Class |
|---|---|---|
| 1 | 1997-09-24 02:43:00 | M59 |
| 2 | 1997-11-27 12:59:00 | X26 |
| 3 | 1997-11-28 04:53:00 | M68 |
| 4 | 1997-11-29 22:28:00 | M64 |
| 5 | 1998-07-14 12:51:00 | M46 |
| 6 | 1998-08-18 08:14:00 | X28 |
| 7 | 1998-08-18 22:10:00 | X49 |
| 8 | 1998-08-19 21:35:00 | X39 |
| 9 | 1998-11-28 04:54:00 | X33 |
| 10 | 1999-01-16 12:02:00 | M36 |
| 11 | 1999-04-03 22:56:00 | M43 |
| 12 | 1999-04-04 05:15:00 | M54 |
| 13 | 1999-05-03 05:36:00 | M44 |
| 14 | 1999-07-19 08:16:00 | M58 |
| 15 | 1999-07-29 19:31:00 | M51 |
| 16 | 1999-08-20 23:03:00 | M98 |
| 17 | 1999-08-21 16:30:00 | M37 |
| 18 | 1999-08-21 22:10:00 | M59 |
| 19 | 1999-08-25 01:32:00 | M36 |
| 20 | 1999-10-14 08:54:00 | X18 |
| 21 | 1999-11-14 07:54:00 | M80 |
| 22 | 1999-11-16 02:36:00 | M38 |
| 23 | 1999-11-17 09:47:00 | M74 |
| 24 | 1999-12-22 18:52:00 | M53 |
| 25 | 2000-01-18 17:07:00 | M39 |
| 26 | 2000-02-05 19:17:00 | X12 |
| 27 | 2000-03-12 23:30:00 | M36 |
| 28 | 2000-03-31 10:13:00 | M41 |
| 29 | 2000-04-15 10:09:00 | M43 |
| 30 | 2000-06-02 06:52:00 | M41 |
| 31 | 2000-06-02 18:48:00 | M76 |
| 32 | 2000-10-29 01:28:00 | M44 |
| 33 | 2000-12-27 15:30:00 | M43 |
| 34 | 2001-01-20 21:06:00 | M77 |
| 35 | 2001-03-28 11:21:00 | M43 |
| 36 | 2001-06-13 11:22:00 | M78 |
| 37 | 2001-06-23 00:10:00 | M56 |
| 38 | 2001-06-23 04:02:00 | X12 |
| 39[+] | 2004-12-30 22:02:00 | M42 |
| 40[+] | 2004-01-07 03:43:00 | M45 |
| 41[+] | 2004-09-12 00:04:00 | M48 |
| 42[+] | 2004-01-17 17:35:00 | M50 |
| 43[+] | 2005-07-27 04:33:00 | M37 |
| 44[+] | 2005-11-14 14:16:00 | M39 |
| 45[+] | 2005-08-02 18:22:00 | M42 |
| 46[+] | 2005-07-28 21:39:00 | M48 |
| 47[+] | 2006-04-27 15:22:00 | M79 |

[+] Missing Data in P6 and P7.

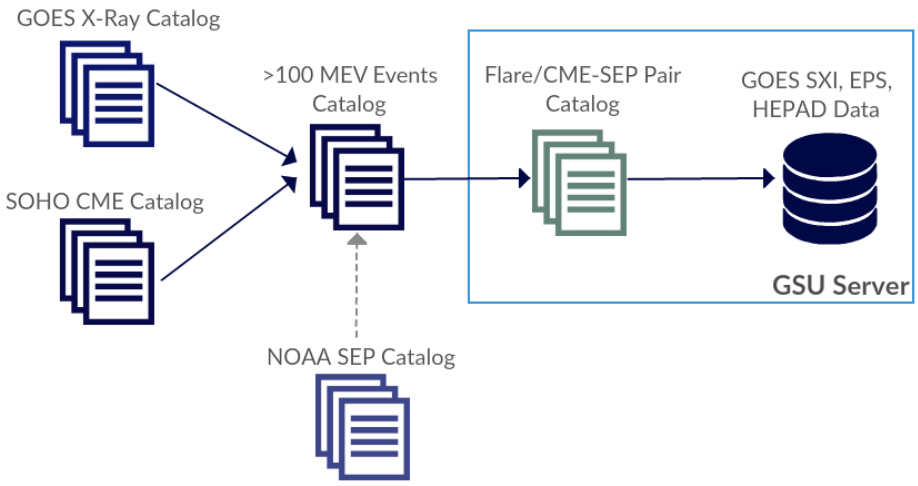



**Figure 2.3:** Catalogs Used to make the X-ray-parent event mapping. X-ray and CME catalogs for detecting the parent event report for flare and CME respectively.

adequately monitor the spacecraft status, and an erroneous decision which disabled part of the on-board autonomous failure detection [32]. It is worth to note that we consulted the NOAA-prepared SEP events catalog along with their parent flare/CME events (*ftp://ftp.swpc.noaa.gov/pub/indices/SPE.txt*). For the case of events that are missing the NOAA catalog, we made our own flare/CME-SEP events mapping. Figure 2.3 shows the three external catalogs that we used to produce our own catalog from which we generate our SEP dataset. To obtain a balanced dataset, we selected another 47 X-ray events that did not produce any SEP events that are, shown in Table 2.4. We noticed that there are nine SEP events (refer Table 2.3 ID:29-37) that happened during the period when only GOES-12 was operational. At that period, channels P6 and P7 failed and there were no secondary GOES. To make sure not to create any biased classifier that relies on the missing data to make the prediction, we made sure to choose nine events from the negative class as well that did not produce any SEP event (see Table 2.4 ID:39-47).

We make a clear distinction between the two different classes of SEP events: gradual and impulsive. We assume that an SEP event is flare accelerated, and therefore impulsive if the lag between the flare occurrence and the SEP onset time is very small and the peak flux intensity has reached a global peak a few minutes to an hour after the onset time. On

the other hand, a gradual event shows a progressive increase in the proton flux trend that does not reach a global peak; instead, the peak is maintained steadily before dropping again progressively. Finally, a non-SEP event happens when there is an X-ray event of the minimum intensity of M3.5 that is not followed by any significant proton flux increase in one of the P6-P11 channels.

## 2.3 Nearest Neighbor Classifier

A number of studies have focused on the use of k nearest neighbors classifier coupled with a distance measures for the time series classification problem. kNN is inherently based on the contiguity hypothesis which states that a test data should have the same label of the training examples in the surrounding radius. In addition to the training data, kNN requires the a priori knowledge of the k parameter and a similarity function that measures the proximity between the time series. Empirically, k=1 has been reported to achieve the best accuracies when coupled with DTW elastic distance [33]. The similarity function is a distance measure that can either be lock-step or elastic. Lock-step measures are used solely when the series have the same lengths as they perform a pairwise distance calculation of the time series data points using an L-norm distance, a generalization of other distances (e.g., Manhattan, Euclidean, ...). An example of lock-step measure is shown in Figure 2.4-a. If the time series do not have equal lengths, re-sampling or interpolation can be used to make their lengths equal. On the other hand, elastic measures are applied also for time series that are not equi-length. An example of elastic measure is shown in Figure 2.4-b. In the next subsection, we introduce all the measures that are couple with 1-NN and used as a base classifier for the EE and our NE ensembles.

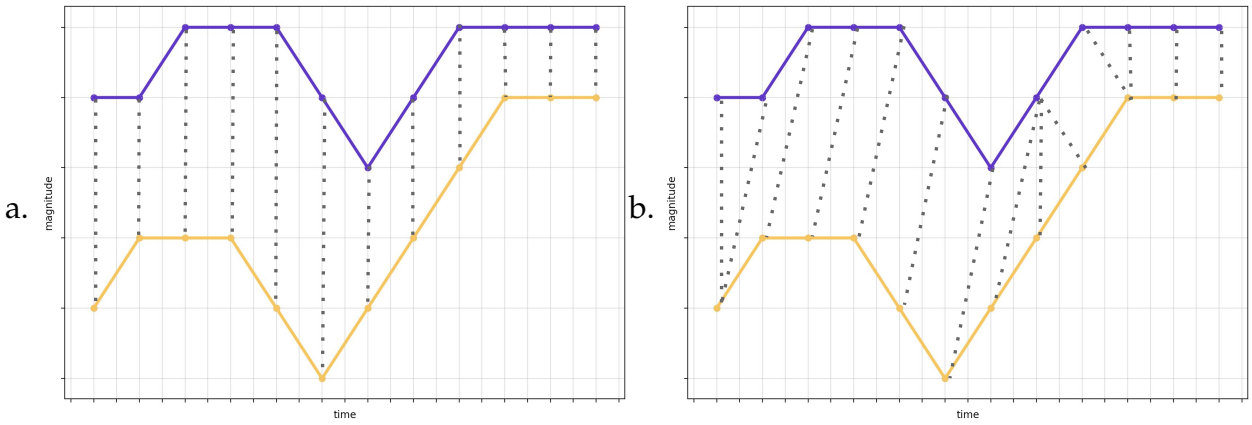



**Figure 2.4:** (a) Lock-step measure (Euclidean Warping) (b) Elastic measure (Dynamic Time Warping)

### *2.3.1 Lock-step Measures*

$L_n$ norm or lock-step measures are among the efficient methods for estimating the similarity between time series and are favored due to their simplicity and applicability in indexing mechanisms. This family of measures does not compensate for the translation between the patterns in the time series which makes is relatively cheaper than other measures. $L_n$ norm measures require that the input time series have equal lengths since they assess the time series similarity based on their pairwise distances. If the time series do not have equal lengths, re-sampling or interpolation can be used to make them equal. The general $L_n$ norm distance formula is:

$$d_{L_n}(x, y) = \left( \sum_{n=1}^{M} (x_i - y_i)^n \right)^{\frac{1}{n}}, n \in \mathbb{N}^+, \tag{2.1}$$

where $x$ and $y$ are the input time series and $M$ is the length of the time series. Equation 2.1 is also known as the L-norm distance which is a generalization of other distances (e.g., Manhattan, Euclidean, ...). When Equation 2.1 is used with $n$=1, Manhattan distance is obtained. If $n$=2, the distance is euclidean. When $n = \inf$ the Equation 2.1 represents

the Tchebychev distance. In this work, we consider Euclidean distance from the lock-step measures category.

### 2.3.2 *Elastic Measures*

Elastic measures are invariant to the non-linear variations that happen in the time dimension. kNN classifier has shown to achieve superior results when used with elastic measures. This is mainly due to the fact that time series data are high dimensional in nature and the chronological order of their values is important. Therefore, the proper alignment of sequences using a flexible measure is crucial for the classification task.

***Dynamic Time Warping (DTW)***

Dynamic time warping (DTW) is a standard elastic measure for assessing the dissimilarity between two time series [34]. Due to its effectiveness in finding an optimal match between two sequences, DTW has been used in many different domains such as shape interpolation [35] [36] [37] [38] [39] and time series matching for incomplete medical data [40]. DTW works by warping the time series in the time domain in such a way that the final warping cost is minimal. The canonical form of DTW is shown in Equation 2.2. M and N represent the lengths of the input time series $x$ and $y$. Initially the D matrix is initialized to $D_{0,0} = 0$ and $D_{i,j}$ to $D_{i,j} = \inf$.

$$D_{i,j} = f(x_i, y_j) + \min\{D_{i,j-1}, D_{i-1,j}, D_{i-1,j-1}\} \quad s.t : i \in (1, M), j \in (1, N) \tag{2.2}$$

**Algorithm 2.1** Longest Common Subsequence Measure

**Input:** First sequence $s_1$, Second sequence $s_2$
**Output:** LCSS distance dist

```
1: M_mxm ← InitializeMatrix(0)
2: for i ← m to o do
3:     for j ← m to o do
4:         M_i,j = M_i+1,j+1
5:         if s1_i = s2_j then
6:             M_i,j ← M_i,j + 1
7:         else
8:             if M_i,j+1 > M_i,j then
9:                 M_i,j ← M_i,j+1
10:            end if
11:            if M_i+1,j > M_i,j then
12:                M_i,j ← M_i+1,j
13:            end if
14:        end if
15:    end for
16: end for
17: return M_1,1
```

### *Derivative Dynamic Time Warping (DDTW)*

Another variation of the original DTW was proposed by Keogh and Pazzani [41] that transforms the input series into their estimated derivative (first-order differences). The estimate represents the average of the line slopes passing through the given point and its left neighbor and the slope of the line passing through the given point and its right neighbor. The squared difference is then applied to compare the derivative of the sequences. The derivative estimation has shown to be more robust to noise and outliers and is defined in Equation 2.3 [42].

$$D_x[q] = \frac{(q_t - q_{t-1}) + (q_{t+1} - q_{t-1})/2}{2} \tag{2.3}$$

**Algorithm 2.2** Move Split Merge

**Input:** First sequence $x_1$, Second sequence $x_2$
**Output:** MSM distance dist

▷ Parameter Initialization
$Cost(1,1) \leftarrow |x_{1_1} - x_{2_1}|$
**for** $i \leftarrow 2$ to $m$ **do**
    $Cost(i,1) = Cost(i-1,1) + C(x_{1_i}, x_{1_{i-1}}, x_{2_1})$
**end for**
**for** $j \leftarrow 2$ to $n$ **do**
    $Cost(i,1) = Cost(1,j-1) + C(x_{2_j}, x_{1_1}, x_{2_{j-1}})$
**end for**
▷ Main Loop
**for** $i \leftarrow 2$ to $m$ **do**
    **for** $j \leftarrow 2$ to $n$ **do**
        $Cost(i,j) = min\{Cost(i-1,j-1) + |x_{1_i} - x_{2_j}|\ Cost(i-1,j) + C(x_{1_i}, x_{1_{i-1}}, x_{2_j})\ Cost(i,j-1) + C(x_{2_j}, x_{1_i}, x_{2_{j-1}})\ \}$
    **end for**
**end for**
**return** $Cost m, n$

### *Weighted Dynamic Time Warping (WDTW)*

Weighted Dynamic Time Warping was proposed by Jeong et al. to suppress the underline assumption of DTW and DDTW which supposes that all the points in the time series have the same weights; therefore, it is possible for points to be matched with neighboring points of the other series that are far in proximity [43]. To mitigate this problem, WDTW introduces a positive weight bias to points that are close to the point to be matched. The weight function that is used is the modified logistic weight function. The weight value of a given point is determined by Equation 2.4.

$$w_i = \left[\frac{w_{max}}{1 + \exp -g(i - m_c)}\right]; \tag{2.4}$$

such that $w_{max}$ is the user-defined upper bound of the weights, $m_c$ is the midpoint of the time series. $m$ is the length of the time series, and $g$ is constant found empirically

and it controls the slope of the logistic function. The larger is the g parameter the more is the penalty.

### *Weighted Derivative Dynamic Time Warping (WDDTW)*

WDDTW is the combination of the aforementioned DDTW and WDTW measures. WDDTW transforms the time series into their first order differences introduced in Equation 2.3 and then applies WDTW on the sequence derivatives to assess their similarity using Equation 2.4.

### *Longest Common Subsequence (LCSS)*

LCSS is another state-of-the-art similarity measure that gained a lot of popularity in the biology field where it is used for comparing DNA sequences (genes). LCSS was later used in time series data mining and was particularly popular for its robustness to noise. LCSS finds the longest subsequence between two sequences and then defines the distance using the length of this subsequence. LCSS uses two parameters $\delta$ and $\epsilon$ that should be optimized using the validation data.$\delta$ is defined as a percentage of the original time series and it signifies the size of the sliding window used to match points across 2 sequences [44]. $\epsilon$ is constant that represents the matching threshold such that $\epsilon \in [0, 1]$. Algorithm 2.1 shows full details about the LCSS measures.

### *Move Split Merge (MSM)*

MSM computes the distance between two sequences by transforming one sequence into another one using only three operations: move, split and merge. The move operation allows a simple edit of the value of a given point in the sequence. Split operation duplicates the value of a given point to the next neighboring position in the sequence. The merge operation allows two nodes of the same values to merge in one and therefore deleting one node from the sequence. Inserting a value to the sequence requires a

split operation followed by a move to modify the newly duplicated value. MSM is a deterministic distance measure that is starting-point invariant and has been proven to obey the triangular inequality property [45,46]; therefore, it has the desirable property of being metric. The full description of the MSM metric is shown in Algorithm 2.2.

***Time Warp Edit Distance (TWE)***

TWE is another metric that was proposed by Marteau [47]. TWE adapts the popular Edit Distance measure to time series matching by introducing a penalty for insertion and deletion of values in the sequences and a *stiffness* parameter that controls the elasticity of the match on the time dimension. More details on the TWE metric can be found in [47].

***Edit Distance with Real Penalty (ERP)***

ERP, also known as the marriage of $Lp-norms$ and Edit Distance [48], is another adaptation of Edit Distance for time series data matching. ERP redefines the delete operation of the traditional Edit Distance by differentiating between deletion from sequence $x_1$ to match sequence $x_2$ (delete_$x_1$) and vice versa (delete_$x_2$). When a match is found, an *L-p norm* distance is applied; otherwise, a penalty is applied when a match is not found.

## 2.4 Shapelet based Classifier

Shapelets, also known as motifs, are relatively small-time series segments that have the property of discriminating between time series classes. Shapelets are phase independent and therefore are modeling the representative pattern that triggers the classification decision regardless of the exact time they happen. As opposed to traditional time series classifiers, shapelet-based learners provide a visual representation of the patterns that

are representative of a class. Here, we will briefly discuss the three most common shapelet-based classifiers.

### 2.4.1 *Fast Shapelets (FS)*

The first shapelet algorithm is the decision tree shapelet approach that was originally proposed by Ye and Keogh [49]. The idea is to initially mine shapelets from the dataset in a brute force fashion and uses the mined subsequences as a basis for data transformation. The second step consists of creating a new feature space containing the distances between all the mined shapelets and all the instances in the training dataset. The new features are then fed to a decision tree model for the classification. An improvement of the previous model has been proposed by Rakthanmanon and Keogh [50] that speeds up the shapelet mining step. The exhaustive search for shapelets is substituted by a simplified approximate shapelet discovery approach. The idea is to first reduce the dimensionality of the input space using SAX [51]. The new SAX words dictionary is again reduced through masking randomly selected letters. The random masking step is repeated multiple times, and a histogram of masked words it built for each class of the problem. the reduced SAX words importance is later assessed thought the frequency of the words in between classes. Finally, the top k SAX words are selected and mapped back to the original shapelets. Figure 2.5 shows an illustration of the SAX words representation and random masking.

### 2.4.2 *Shapelet Transform (ST)*

Lines et al. [11] propose a new shapelet learner that uses a decision tree learner, similar to Fast Shapelets, to classify time series data. The difference between FS and ST is that ST mines shapelets prior to the classification step. ST does not search for the most important shapelet at the level of each decision tree node, top-K shapelets are mined

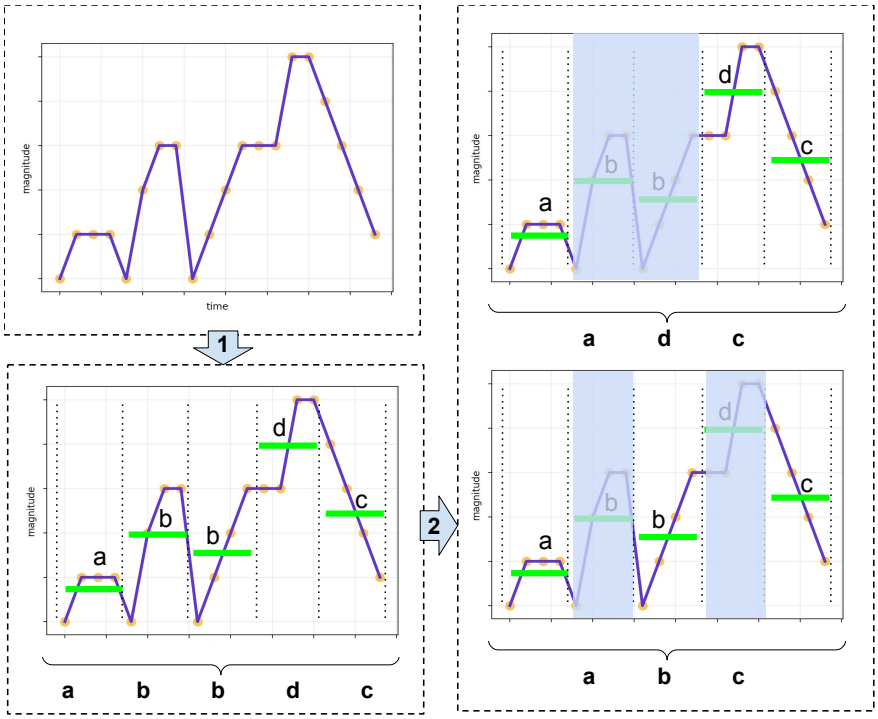



**Figure 2.5:** Fast Shapelet SAX representation and multiple random masking.

a-priori. The transform step consists of calculating the distances of all mined shapelets to all training instances. Shapelets evaluating is done in reference to every class and shapelet importance is measured with respect to the discriminatory power of the given shapelet to all the classes if the problem (one class at a time). Since the first output of the ST is the newly transformed data spaced, many classifiers were tested on the transformed data space, namely: Naive Bayes, C4.5 decision tree, Support Vector Machines with linear and quadratic basis function kernels, Random Forest (with 500 trees), and Rotation Forest.

### 2.4.3 *Learned Shapelets (LS)*

Grabocka et al. [52] are the first authors that approach the shapelet mining and time series classification from an optimization lense. The proposed algorithm, named Learned Shapelets (LS), leverages gradient descent algorithm for the shapelet search. LS algorithm

searches for shapelets that are not directly selected sub-sequences from the time series dataset. LS rather starts with an average sub-sequence and learns an approximate shape that identifies a given class pattern. The initialization of the k shapelets is obtained as a result of the centroids of the k-means clustering algorithm from the training data. LS considers the shapelet length as a hyper-parameter that needs to be identified prior to training. In other terms, one of the assumptions of this work is that there exists only one pattern size that leads to an optimal performance level. The model makes use of a novel mathematical formulation of the task via a classification objective function. The objective function is a logistic loss function that jointly learns the weights for the regression W, and the shapelets S in a two-stage iterative process to produce a final logistic regression model. A check is performed when half of the number of allowed iterations is reached to check if divergence has occurred. In addition, the authors claim that LS can learn true top-K shapelets by also capturing their interaction.

## 2.5 Deep Learning for Time Series Classification

### *2.5.1 Multi-Layer Perceptron*

The first attempt to use a neural network approach for classifying univariate time series data, as appeared in [53], consisted of a relatively simple four-layered Multi-Layer Perceptron (MLP). Each layer of the network is a fully-connected layer of 500 neurons with a ReLU activation function. To prevent overfitting, a percentage of neurons ranging from 10% to 30% are randomly dropped during each forward pass of the backpropagation. MLP achieved the worst performance level compared to all state-of-the-art univariate time series classifiers.

### 2.5.2 *Convolutional Neural Network*

Inspired from image classification task, Fully Convolutional Neural Network (FCN) models were first proposed in [53] to approach the univariate time series classification problem. The FCN architecture is composed of three convolution blocks, used as feature extractors, each of which is composed of a two-dimensional kernel followed by batch normalization [54] and ReLU activation function. The length of the input time series remains unchanged as they flow in the network (due to use of stride of 1 and zero-padding), and the size of the kernels was fixed for all the datasets. The model topology is therefore data-independent.

### 2.5.3 *Residual Neural Networks (ResNets)*

Residual network (ResNet), also proposed by [53], is the deepest architecture that extends traditional neural networks by adding a shortcut connection between each consecutive residual block that enables the flow of the gradient directly to the next layer. This special setting attenuates the problem of exploding and vanishing gradient [55]. The networks is composed of three sequential residual blocks, in their turns containing three two-dimensional fixed-sized filters, followed by a softmax layer having the number of neurons equal to the number of labels in the dataset. Similar to FCN, batch normalization and ReLU activation is applied after each kernel. ResNet is also a data-independent architecture with fixed-sized network and kernels.

## 2.6 Time Series Ensemble Methods

Combining the output of multiple predictive models, referred to as ensembling, is a common practice in several communities from different fields such as pattern recognition and knowledge discovery, statistics, and machine learning. The main reason for combining multiple models is to improve the accuracy of single classification or regression models. According to the No Free Lunch (NFL) theorem, it is generally accepted that there is no one model that dominates over all types of learning problems [23, 56]. In addition, different classifiers may be useful in representing different aspects of the problem. In other terms, while some classifiers might be powerful in classifying a given class c, other classifiers might be less accurate in classifying c but more useful when classifying other classes. The aim of the ensemble is to leverage the strength of the base learners to optimize the final classifier accuracy. Another motivation for using ensemble methods is for scaling algorithms when applied to very large databases without compromising accuracy. One of the solutions to this problem is to horizontally partition the database into manageable sections and assign every data partition to a learner. The final ensemble represents the combination of all the base learners with a given fusion strategy. When a combination of base models of the same learning algorithm is used, the ensemble is called *Homogeneous*. The difference between base learners participating in a homogeneous ensemble stems from the instances in which they are trained as well as the induced randomness in the learning algorithm. On the other hand, when the ensemble is formed from running different learning algorithms, the ensemble is called *Heterogeneous*. In this paper, we propose a new heterogeneous ensemble based on a variety of base learners.

**Problem Formalism**: The problem can be formalized as follows: Assume that a pattern space represented by a mutually exclusive set of classes $S = \{C_1 \cap C_2 \cap \ldots \cap C_M\}$, for all $i \in \{1, \ldots, M\}$. A classifier, also called **expert**, predicts the class of a data sample $x \in S$ as

$e(x) = j$ such that $j \in \{1, \ldots, M\}$. An ensemble $E$, is the **fusion** of $K$ different base experts $e_k$, $k \in \{1, \ldots, K\}$, each of which assigns a class to a data sample as $e_k(x)$.

The fusion step usually involves tracking the error of the models in the training and validation phase and use them to weight the classifier's predictions. Simple fusion strategies include weighted voting, weighted voting per class, and majority voting. Stacking [57] is another fusion methodology that involves training a meta-learner on the class predictions or class probabilities of the base classifiers. The stacking method learns the inter-dependencies that exist between base classifiers and inherently takes into consideration their diversity [58].

### *2.6.1 Classifier Selection*

One of the simplest ensemble strategies, that is comparable to other more complex strategies, consists of training base classifiers and selecting only a subset of them that shows a high accuracy potential. The selection strategy evaluates each of the classifier algorithms on the cross-validation set and selects the best ones for the final evaluation on the test dataset. A simple, but effective, method in the classifier selection paradigm is the Evaluation and Selection (ES) method. The former evaluates all the classifiers based on the validation set and selects the most accurate one to be used in the test set. Džeroski indicates that ES should be considered as a baseline when evaluating a new ensemble methodology [59]. A potential downside of ES is that it may fail to select the best classifier given that the most accurate classifier on the validation set is not necessarily the best classifier on the test set.

Another method for selecting base classifiers consists of distinguishing learning algorithms that have shown to be accurate in the same domain of the problem and selecting them. The former strategy relies on building performance records of simple and efficient learning algorithms on different set of domains and using the learners that perform the

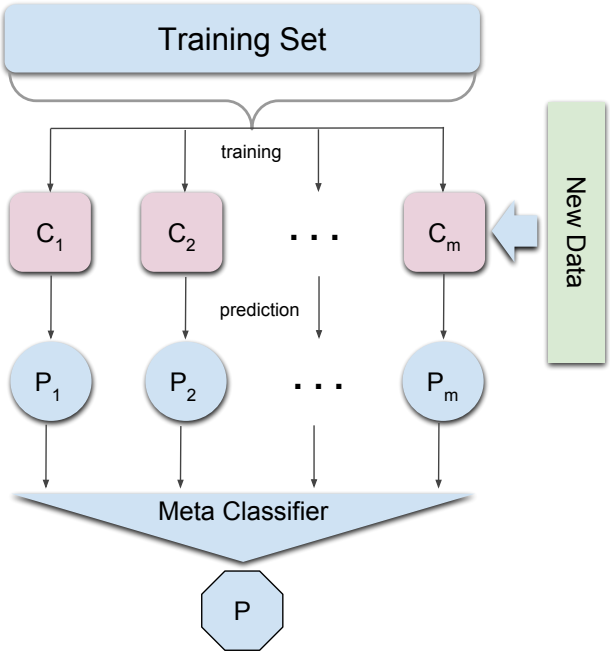



**Figure 2.6:** Stacking Flowchart

best in the domain of the problem. A number of approaches have been investigated to build the domain-specific performance record, some of which use landmarking [60], model-based data characterization [61], histograms [62], statistical and information-theoric measures [63]. Dynamic Classifier Selection is another selection method that uses different classifiers, for different instances in the test space. Classifier selection is based on the classifiers' performance on the nearest training instances' neighbors to the test instance [64].

### 2.6.2 *Classifier Fusion*

Classifier fusion paradigm allows all the classifiers to participate in the final prediction (including the less accurate ones). Voting is one of the widely used strategies when combining classifiers that are either homogeneous, such as Random Forest [65], or

heterogeneous. In voting, every classifier predicts the class of the test instance (or the class probabilities), and the class that received the most vote from all the classifiers constitute the ensemble prediction. The former type of voting is called plurality or majority voting. Weighted voting is another classifier fusion methodology that discriminates between the models based on their performance. Weighted voting assigns a weight to each classifier that is proportional to their respective accuracy levels in the validation set. Another variant of weighted voting is weighted voting per class. The former evaluates classifiers based on their accuracy on the validation set with respect to the classes in the classification problem.

**Algorithm 2.3** Stacking Classifier

**Input:** Training data $\mathcal{D} = \{x_i, y_i\}_{i=1}^{m}$ ($x_i \in \mathcal{D}$ , $y_i \in \mathcal{Y}$)

**Output:** An Ensemble classifier $\mathcal{H}$

▷ Step 1: Train Base Classifiers
**for** $t \leftarrow 1$ **To** $T$ **do**
    Train($h_t$, $\mathcal{D}$)
**end for**
▷ Step 2: Construct a New Dataset from $\mathcal{D}$
**for** $i \leftarrow 1$ **To** $m$ **do**
    $\{x'_i, y_i\}_{i=1}^{m} \leftarrow$ GenerateDataset($\{x_i, y_i\}_{i=1}^{m}$)
    where, $x'_i = \{h_1(x_i), h_2(x_i), ..., h_T(x_i)\}$
**end for**
▷ Step 3: Train Meta-Learner
Train($h'$, $\mathcal{D}'$)
**return** $H(x) = h'(h_1(x), h_2(x), \ldots, h_T(x))$

Meta-learning is a common strategy that models the learning process of classifiers [66] and use it for their selection or fusion. One of the highly-adopted meta-learning strategies

is stacking [57]. An illustration of the stacking process is shown Figure 2.6. The first step of the stacking involves training all the classifiers using the complete training data (refer Algorithm 2.3 lines 2-4). We will refer to this step throughout the paper as the Level-1 training. The output from all the classifiers, be it the predicted classes or the class probabilities, constitute the new meta-features for a new learning problem. A new dataset is generated by vectorizing the meta-features (lines 6-9). Finally, Level-2 training consists of training the meta-learner on the new dataset (line 11). A representation of the stacking is shown in Figure 2.6. Stacking was coupled with a variety of meta-learners such as logistic regression [67] and multi-response linear regression [59].

# CHAPTER 3

# 1-D Convolutional Neural Networks (1DCNN) for Shapelet Mining

State-of-the-art shapelets studies involve mining the sequences from the training data and keeping only the candidates' segments that show high predictive power. The former task is exhaustive which makes it hard to be adopted in an operational model context. A new recent line of work proposes to approach the problem from an optimization perspective by learning the shapelets (as opposed to mine them). Following the same line of thoughts, we propose a new shapelet mining learner, *1DCNN*, that exploits the idea of convolutional neural network from image processing field and generalizes it to the one-dimensional case input. 1DCNN model has the particularity to learn shapelets of different lengths using a black-box neural network model. The *1DCNN* optimizes the entire classification schema by learning the shapes of the representative patterns.

## 3.1 Network Architectures:

Our general network architecture consists of a four-layered convolutional neural network. The original convolution neural networks, initially designed for image processing, takes a two-dimensional image input that is convolved with a three-dimensional/tensor kernel in the next layer. In analogy with traditional CNN, our input is a one-dimensional time series followed by a two-dimensional/matrix kernel in the convolution layer. Figure 3.1-a illustrates the analogy of the traditional two-dimensional convolutional network and our proposed 1DCNN model in Figure 3.1-b. Following the same line of thoughts as [10],

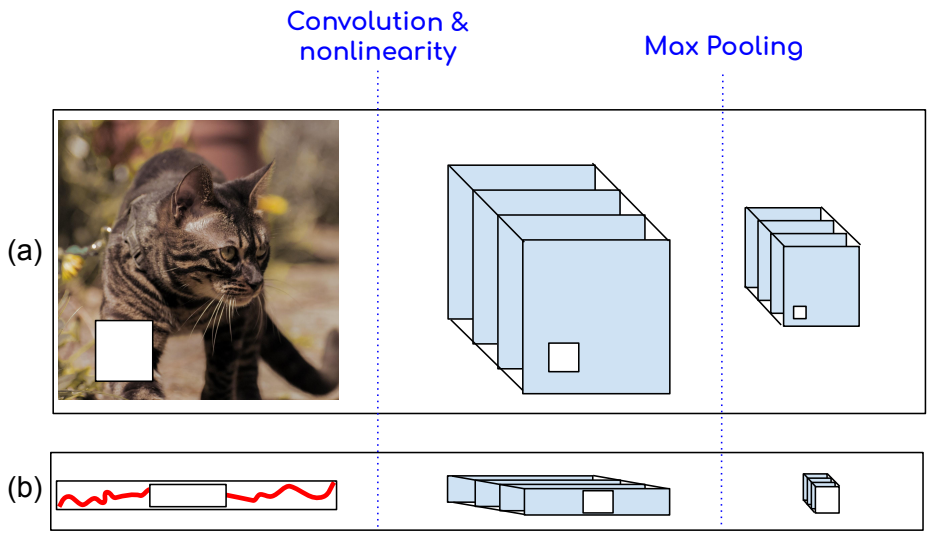



**Figure 3.1:** Analogy of two-dimensional CNN for image segmentation and 1DCNN for time series data

our hypothesis is that a dataset can have discriminative shapelets of different lengths all of which can contribute with independent information to the classification. As a matter of fact, we considered three shapelet sizes in the same model. Our model architecture is shown in Figure 3.2 where the first layer represents the input time series data matrix that is passed to the convolutional layer. The former convolves the same matrix three times using filter matrices of different sizes and pass it to the fully connected layer after applying a max-pooling and non-linear activation function. The max-pooling operation preserves only the convolution operation that led to the maximum sum when sliding the kernel over the time series. In other terms, the only similarity between shapelet and the best alignment in the time series is taken into consideration for the classification. Figure 3.3 illustrates the idea behind one-dimensional convolution and max pooling operations in the context of shapelets. A candidate shapelet is overlaid over the input time series and shifted to the right along the time dimension until all the possible alignments are considered. The result of each convolution operation between the shapelet and the time series segments are recorded in a convolution vector that is then passed to the max-pooling layer which keeps the element with the maximum value in the vector.

In Figure 3.3 9 is the maximum element of the convolutional vector that represents the fourth alignment. The concatenation of the max convolution elements of all the kernels represents the new feature space input for the fully connected layer. Finally the last layer consists of the softmax function that produces probability likelihoods of a time series belonging to a particular class. The basic convolution operations are defined in Equation 3.1.

$$\begin{aligned} y &= W \circledast x + b \\ h = \mathrm{ReLU}(y) &= \max(0, y) \\ s &= \max(h) \end{aligned} \tag{3.1}$$

where $\circledast$ is the convolution operation. We train our 1DCNN with backpropagation in conjunction with Adaptive Moment Estimation (Adam) [68] optimization algorithm. Algorithm 3.1 shows the full mini-batch gradient descent algorithm that we used. Lines 6-10 defines the steps involved in

We explored three different flavors of the same architecture where the convolutional layer contains kernels that are either: (1) sized based on the input data, (2) have a fixed long size, or (3) have a fixed wide size.

### *3.1.1 Data Dependent Variable-Length 1DCNN:*

The first architecture of the 1DCNN has a variable size of kernels in terms of both numerosity and lengths. The number of kernels in each of the convolution unit is directly proportional to the number of classes of the problem. The idea is that the more classes exist in the multiclass classification problem, the more complex the classifier should be in order to learn sufficient complex rules to discriminate between classes. We set the number of kernels in each convolutional block to be (20*k, with k being the number of classes).

**Algorithm 3.1** General 1DCNN mini-batch Gradient Descent Algorithm

**Input:** Dataset $\mathcal{D}$, learning rate $\alpha$, $len_1$ size of the first kernels, $len_2$ size of the second kernels, $len_3$ size of the third kernels,number of epochs $ep$ , mini-batch size $n_m$
**Output:** The trained $network$

$iteration \leftarrow 0$
**while** $iteration < ep$ **do**
    ▷ Propagate the input (Forward Pass)
    **for** $j = 1,\ldots, n_m$ **do**
        **for** $len$ in $[len_1, len_2, len_3]$ **do**
            $I_j = W_{len} \circledast x + \theta$
            $H_j = ReLU(I_j)$
            $O_j = maxPool(H_j)$
            $y_{hat_j} = Softmax(O_j)$
            $Loss_{ij} = -\frac{1}{N} \sum_{c=1}^{m} y_{hat_j} log(p_j)$
        **end for**
        ▷ Backpropagate the errors (Backward Pass)
        **for** $len$ in $[len_1, len_2, len_3]$ **do**
            $W_{len} = ADAM(\sum_1^{n_m} Loss_j, W_i, \alpha, \beta_1, \beta_2)$
        **end for**
    **end for**
    $iteration \leftarrow iteration + 1$
**end while**
**return** $network$

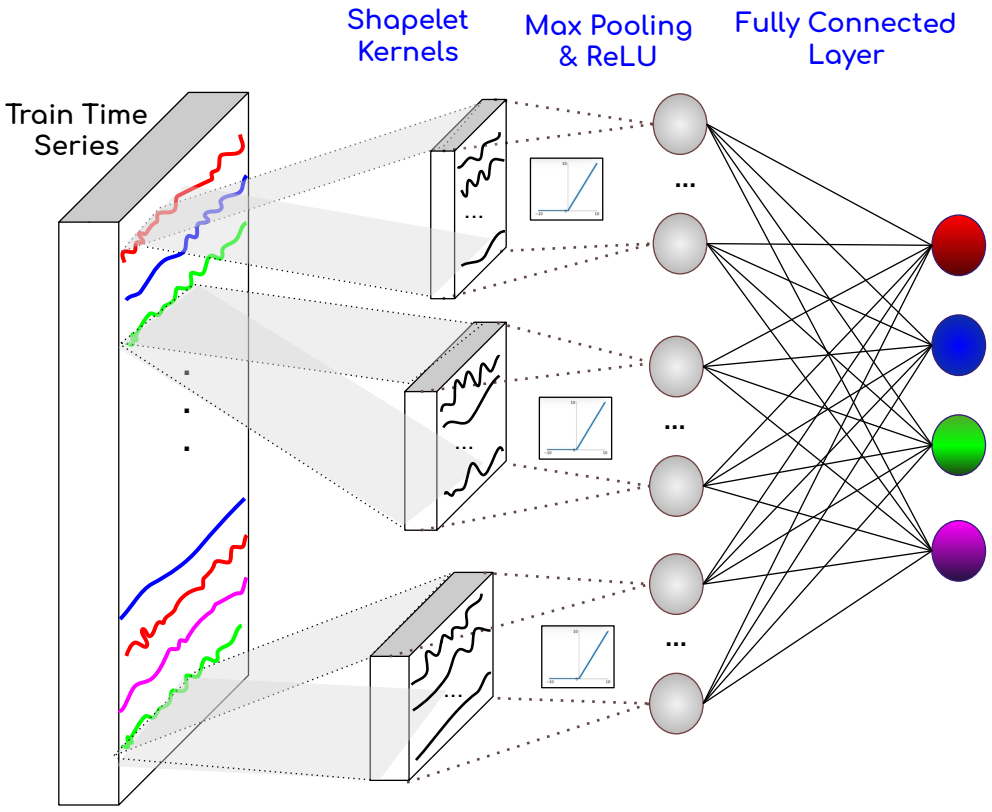



**Figure 3.2:** 1DCNN General Architecture for a four-class TSC problem

The second variation comes from the size of the one-dimensional kernel in each block. As mentioned earlier, we designed our model to be able to capture patterns of different-length following our initial hypothesis that states that a dataset can have more than one optimal pattern size as shown in [52]. The second assumption of variable 1DCNN is that patterns should be proportional to the initial time series size. In other terms, for shorter time series of size 100, shapelets of size 12 are good candidates to capture discriminate patterns, while the same size could be too small for time series of length 1000. For the former dataset, shapelets of length 12 could capture very granular micro-behaviors that could represent noise. The statement is especially true in our case since we do not use any dimensionality reduction preprocessing step on the raw time series. We used the same shapelets lengths window suggested in [52]: $len_i \in \{0.025, 0.125, 0.2\}$ x 100% of the

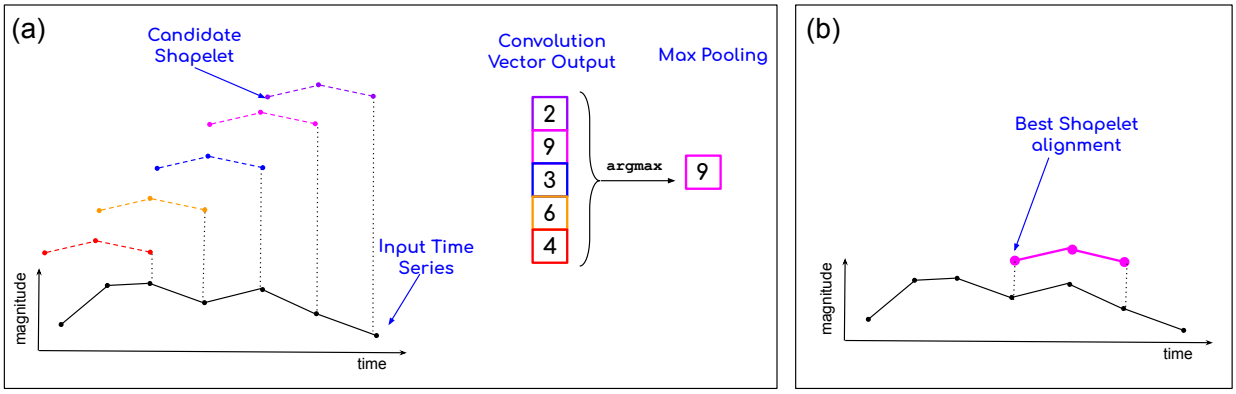



**Figure 3.3:** (a) Illustration of the convolution process of the candidate shapelet(shown in dotted line) over the input time series (shown in solid line) and the max pooling operation and (b) the best candidate shapelet alignment that resulted

time series size (where $i \in [1, 3]$). Figure 3.4-(a) illustrate the variable kernel size(width) with respect to the dataset time series size.

As a result, another indirect size variability that affects the network topology occurs in the concatenation and fully connected layers (third layer in Figure 3.2) due to the variation in size(length and width) of the former convolution layer (second layer in Figure 3.2). The number of the input neurons equals the number of max-pooling operations which equates the length of the kernels x 3 (since there are three parallel convolution blocks).

### 3.1.2 *Fixed-sized wide 1DCNN_l:*

The second flavor of the 1DCNN follows the same general network topology in Figure3.2 with the only difference from vanilla 1DCNN of the size of the kernels. We wanted to explore the effect of having a same fixed-sized kernel for all the datasets on the model performance. We set the number of kernels at each convolutional block to 100 which is within this range 20 x $[median_k, \mu_k]$. The number of kernels approximates the same number of kernels used in 1DCNN. The lengths of the kernels in the convolutional blocks has been fixed to $len_i \in \{64, 128, 256\}$ (where $i \in [1, 3]$). This architecture is similar to 1DCNN with the particularity of having long kernels, we therefore named it 1DCNN_l.

In total there are $P_{1DCNN\_l}$ = 3 x 100 x (64 + 128 + 256) parameters to optimize. An illustration of the size of the kernels relative to their size for other architectures is shown in Figure 3.4-(b).

### 3.1.3 *Fixed-sized large 1DCNN_w:*

Following the same line of thought for designing 1DCNN_l we designed 1DCNN_w network. The former has the property of being wide in terms of shapelet size and short in terms of kernel numerosity per convolutional block. We used a fixed-length kernel of $len_i \in \{6, 12, 25\}$ in the three parallel convolutional blocks respectively. For a fair comparison with 1DCNN_l, we set the number of filters to 1000. In total there are $P_{1DCNN\_w}$ = 3 x 1000 x (6 + 12 + 25) parameters to optimize ($\approx P_{1DCNN\_l}$). An illustration of the 1DCNN_l kernels with respect to 1DCNN_w, and 1DCNN.

### 3.1.4 *Data Preprocessing*

The performance of the different variants of 1DCNN methodology and other baselines are evaluated on the original 73 datasets from the UCR Machine Learning repository [69] that are available on UCR [1] website. We used the same default train and test split that was given and applied it to all benchmarks for comparability purposes. We performed z-normalization of the train and test sets prior to classification as shown in Equation 3.2.

$$Z_{norm} = \frac{x - \hat{x}_{train}}{\sigma_{train}} \tag{3.2}$$

where $\hat{x}_{train}$ and $\sigma_{train}$ are the mean and standard deviation of the training set respectively.

1 https://www.cs.ucr.edu/ eamonn/time_series_data/

## 3.2 Experimental Evaluation

### *3.2.1 Parameters Setting*

In this section, we will specify the user-defined input parameters of the algorithm. The learning rate of the algorithm has been set to a default value of $\alpha = 0.001$ and the number of epochs used to train 1DCNN variants is $ep = 3000$ to avoid premature training leading to underfitting.

We used a mini-batch size of $n_m$=16, and the kernel lengths are set depending on the 1DCNN variant as discussed in Section 3. In addition, in order to speed up the learning process, we used batch normalization [54] of the kernels before applying ReLU as the activation function for non-linearity of our model and a categorical cross-entropy cost function as defined in Equation 3.3. We trained our network with Adam optimizer with default parameters ($\beta_1$=0.9, $\beta_2$=0.999 and $\epsilon$=1e-8). We also imposed a learning rate reduction once the learning plateaus and there is no improvement for 5 consecutive epochs. Since random initialization is doing poorly leading to networks converging more slowly and towards ultimately poorer local minima, we tried to warm the initialization of the network weights and convolutional block kernels, we used Xavier initializer [70].

$$-\frac{1}{N}\sum_{i=1}^{N} \log(p_{1DCNN})[y_i \in C_{y_i}] \tag{3.3}$$

We used Tensforflow framework [71] coupled with Keras library [72] to implement our proposed network. For reproducibility purposes, the source code datasets and instructions are all made publically available.

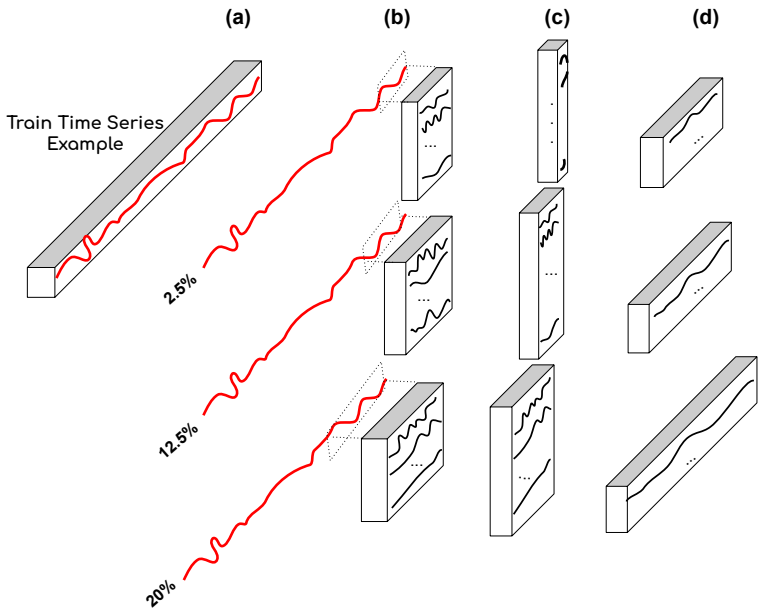



**Figure 3.4:** (a) Example of input time series used to size the convolutional layer kernels in (b) and fixed-sized (c) long and (d) wide kernels

### 3.2.2 *1DCNN Performance*

In this section, we will discuss the experiments we conducted to show the efficiency of our 1DCNN from different perspectives. We first discuss the accuracy performance of the 1DCNN in comparison with other state-of-the-art deep learning methods, we then explore the effect of having different-lengths shapelets in the network using a use-case. Finally, we compare the overall accuracy of the 1DCNN and the other shapelet-based classifiers.

Table 3.2 shows the classification error rates of the three 1DCNN model variants and the MLP, FCN, and ResNet applied on a subset of 40 datasets from UCR. The error rates have been averaged per class of the multiclass classification problem. The first observation that can be made is that looking at only the deep learning baselines (MLP, FCN, and ResNet), FCN is a clear winner in the group. This behavior is expected since the MLP model is the least complex model. MLP is a shallow network of 4 layers that feeds the initial time series to the network and applies traditional linear aggregation of input and weight

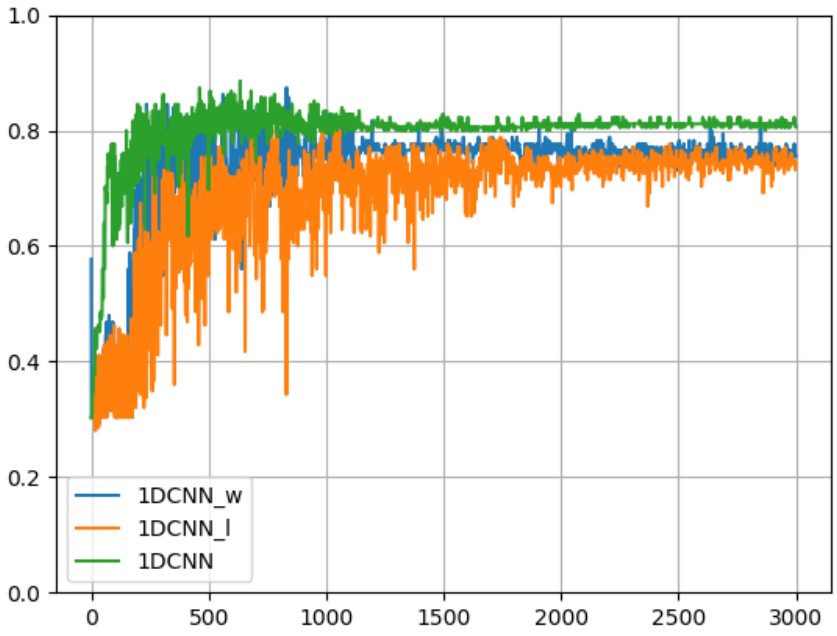



**Figure 3.5**: Cross-validation learning curves of the three 1DCNN, 1DCNN_w, 1DCNN_l variants on the ArrowHead dataset

vectors at the level of each neuron which does not necessitate the use of any kernel. As a result, due to the simplicity of the network, MLP is not able to generalize well. On the other hand, ResNet, the most complex model among the baselines, does not achieve the best performance levels in the group. contrarily to MLP, the convolutional blocks in the ResNet makes it deeper and tends to overfit the data. From the second group of classifiers (1DCNN, 1DCNN_l, 1DCNN), 1DCNN is a clear winner. Figure 3.5 shows an exampled of the learning curves of the three variants on the ArrowHead dataset. The figure shows that all the three variants follow the same general learning behavior with 1DCNN having better convergence with an increasing number of epochs. While the learning curves are noisy, they show a convergence trend at the end of the learning. This behavior is expected since the weights and kernels are updated at every relatively small mini-batch iteration (16 examples).

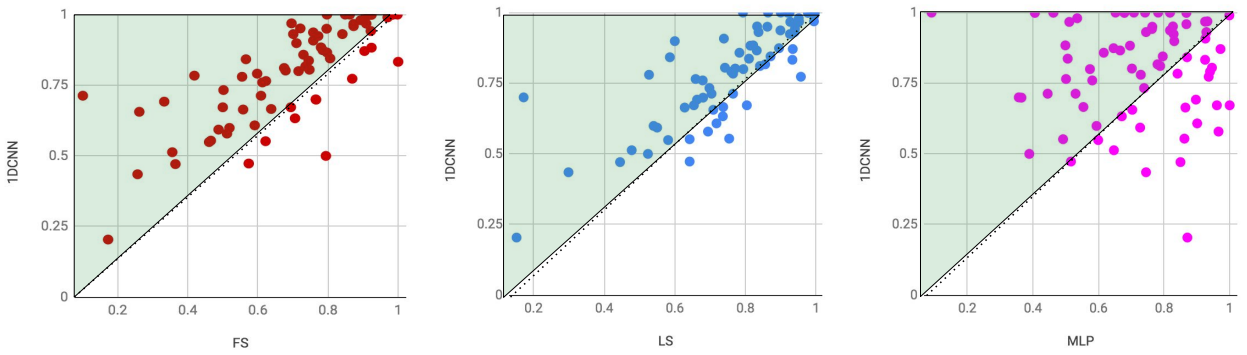



**Figure 3.6:** Accuracy of 1DCNN versus FS,LS and MLP_n. (The green area highlights the datasets where the proposed 1DCNN is winning)

The former model also achieved the best performance levels across all the other deep learning baselines. The unique characteristic of 1DCNN compared to its counterparts is that it is designed with kernels that are sized based on the dataset being fed to it. This finding supports our initial hypothesis that the network kernels, modeling the learned patterns, should be proportional to the length of the signal. We can also notice that FCN, ranked second the list (with a total of 14 wins), is performing relatively within the same performance levels of our proposed 1DCNN. However, FCN uses two-dimensional kernels that were initially designed to capture two-dimensional image features (such as corners and edges) which have a limited interpretability for the case of one-dimensional time series data.

Table 3.3 shows the performance of the three proposed 1DCNN variants in comparison to Learned-Shapelet (LS) [52] and Fast Shapelet (FS) [50] shapelet-based algorithms as well as the dataset's metadata (number of training examples, number of testing examples, time series length, number of classes). In addition, we considered another baseline based on training basic classifiers on shapelet transformed data and feeding the newly transformed feature space into a multi-layer perceptron (MLP_n) [11]. 1DCNN shows promising results overall being ranked first. Figure 3.6 illustrate all the results of Table 3.3 using a scatter plot. Every point in the plot represents a dataset having the x and y coordinate values the competitive baseline and the 1DCNN respectively.

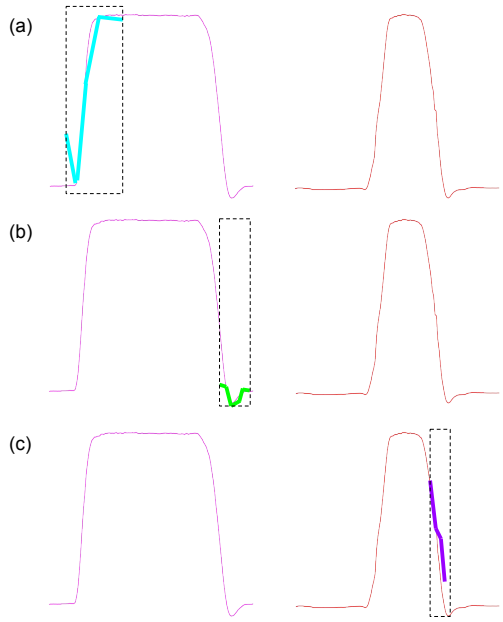



**Figure 3.7:** Example of three different lengths learned shapelets overlaid on top of the two classes of the Gun_Point dataset

Among the shapelet-based classifiers, LS, which was the first attempt for time series shapelet learning using optimization, is the second winner (with a total of 14 total wins). The mains assumption of the LS model is that there is an optimal shapelet length that better models the underlying pattern in the data. LS performs a parameter search for the optimal shapelet length using a grid-search on a window of $len \in \{0.0025, 0.075, 0.125, 0.175, 0.2\}$ prior to training. As a result, the model is trained to learn shapelets of exactly one length.

Our initial hypothesis states that a dataset might have representative shapelets of more that models their distinctive class patterns. The results of 73 datasets support our assumptions. In order to better understand this assumption, we resorted to a visual qualitative examination using a use-case. For simplicity sake, we choose to zoom in the Gun_Point dataset that has binary labels and relatively easy to classify due to the distinctive shapes of the two classes of the problem (refer to Table 3.2). Figure 3.7 shows two randomly sampled examples from the dataset each belonging to the two different classes. Figure 3.7-a shows one of the longest learned kernels belonging to the first

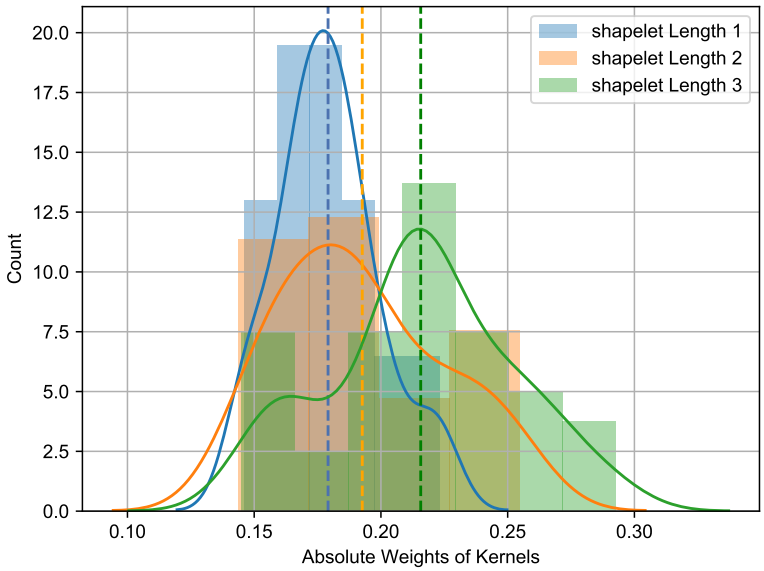



**Figure 3.8:** Histograms of relative weights of the three kernel categories

convolutional blocks of the 1DCNN network(refer Figure 3.2)overlaid on top of the time series example belonging to the first class.

Figure 3.7-b and Figure 3.7-c show the learned kernels of the second and third convolutional blocks of the 1DCNN network. We can visually notice that for the three kernels, the exact class representative pattern is contained within the kernel window. In other terms, increasing the shapelet size in Figure 3.7-c will comprise noisy data points that belong to both classes and might confuse the model. Similarly, decreasing the kernel window of Figure 3.7-a entails capturing only a partial pattern that in the best case is mostly present in the first class but also in some noisy examples of the second class. A more robust shapelet such as the one learned in Figure 3.7-a includes more data points that model the slope of the rising curve and the distinctive shape of the corner. In addition, Figure 3.8 shows the histogram of the relative weights of the three kernel lengths in one dataset, which reveals that there is no one distinctive shapelet length that dominates over the others. This finding further supports our hypothesis that a dataset can have more than one discriminative shapelet length.

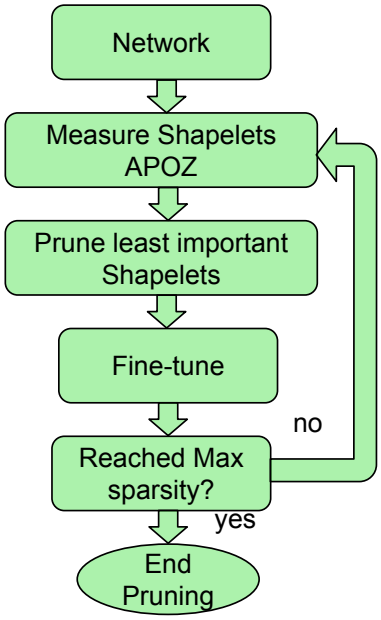



**Figure 3.9**: Model Pruning protocol

## 3.3 Network Pruning

Network pruning is used in this application to reduce the number of shapelets learned by the model. The resulting pruned model is a more robust network having a better generalization capability. In addition, the pruned model retains the exact number of shapelets necessary to keep the same performance levels of the original unpruned model. The proposed pruning methods are a four-steps procedure, as illustrated in Figure 3.9, that takes the original trained model (M) as an input. The first step of the procedure is to measure the importance of all the (initial 900) shapelets/kernels of the unpruned model using a weighting heuristic. The second step of the pruning procedure consists of deleting the r% least important shapelets in the network and then fine-tuning the new network ($M_{pruned1}$). The later step consists of using the weights of (M) as a weight initialization for the new model ($M_{pruned1}$) before training it on the same train data. Finally, the above three steps are repeated until the maximum sparsity level (K%) is reached. Table 3.1 describes the details of the original and pruned 1DCNN models.

**Table 3.1:** Original and pruned 1DCNN networks parameter. Our model uses the following values: i=30, j=50, k=70. The weight initializations are Glorot Normal and the activation function is ReLu.

| *Network* | **1DCNN** | **1DCNN**_K%pruned |
|---|---|---|
| ***Conv Layers*** | 0.3 * TS length, pool<br>0.5 * TS length, pool<br>0.7 * TS length, pool | 0.3 * TS length, pool<br>0.5 * TS length, pool<br>0.7 * TS length, pool |
| ***FC Layers*** | TS length, 300<br>TS length, 300<br>TS length, 300 | TS length, i% * 300<br>TS length, j% * 300<br>TS length, k% * 300 |
| ***Iterations*** | 3,000 | 3,000 |
| ***Batch*** | 64 | 64 |
| ***Optimizer*** | Adam 1.2e-3 | Adam 1.2e-3 |

We used the Average Percentage of Zeros (APoZ) as the pruning criterion to measure the shapelets' importance of the intermediate models. APoZ criteria was proposed by Hu et.al in [73] and it stems from the idea that since a large portion of zero activation function outputs exists in a neural networks, a network can be pruned by trimming the neurons that did not highly contribute to the decision.

$$APoZ_c^i = APoZ(O_c^i) = \frac{\sum k^N \sum j^M f(O_{c,j}^{(i)}(k))}{N * M} \tag{3.4}$$

$$\begin{cases} f(.) = 0, & \textbf{if} \quad O_{c,j}^{(i)}(k)) = 0 \\ f(.) = 1, & \textbf{if} \quad O_{c,j}^{(i)}(k))! = 0 \end{cases} \tag{3.5}$$

APoZ is formally defined as the percentage of neurons that did not fire (have zero activation) after applying the ReLu function. In addition to pointing the most discriminative shapelets in the network, APoZ is also a general measure of redundancy in the model due to the overwhelming amount of parameters in the network which is generally the cause of an overfitted model and inaccurate queries. The canonical form of the APoZ

measure is shown in Equations 3.4 and 3.5. Where $O_c^i$ is the output of the c-th channel in the i-th layer, M is the dimension of the feature map, N is the number of data points, and $f(.)$ is an indicator function that evaluates to 1 in case the output of the activation is zero as shown in Equation 3.5.

### 3.3.1 *Experimental Setup*

### 3.3.2 *Datasets*

The performance of different variants of the original and pruned 1DCNN methodology and other baselines are evaluated on 59 datasets from the UCR Machine Learning repository [69] that are available on UCR [2] website. The datasets originate from different sources and domains such as human motion data, sensor data, simulated, and medial sequences. The first columns of Table 7.1 in the appendix section presents the metadata of the datasets used in this study (dataset name, dataset size, time series length, and the number of classes). The fourth column of the table ($\mathcal{K}$%) shows the optimal sparsity percentage level that we experimentally found for each dataset.

## 3.4 Experimental Results

The contribution of the proposed model is its potential use for (1) prediction and (2) shapelet mining. In this section, we will discuss two types of experiments that we conducted. The first category aims to quantitatively show the efficiency and robustness of the algorithm for prediction-based querying purposes. The second category of experiments aims to qualitatively show the superiority of the model in miming shapelets for the sake

2 https://www.cs.ucr.edu/~eamonn/time_series_data/

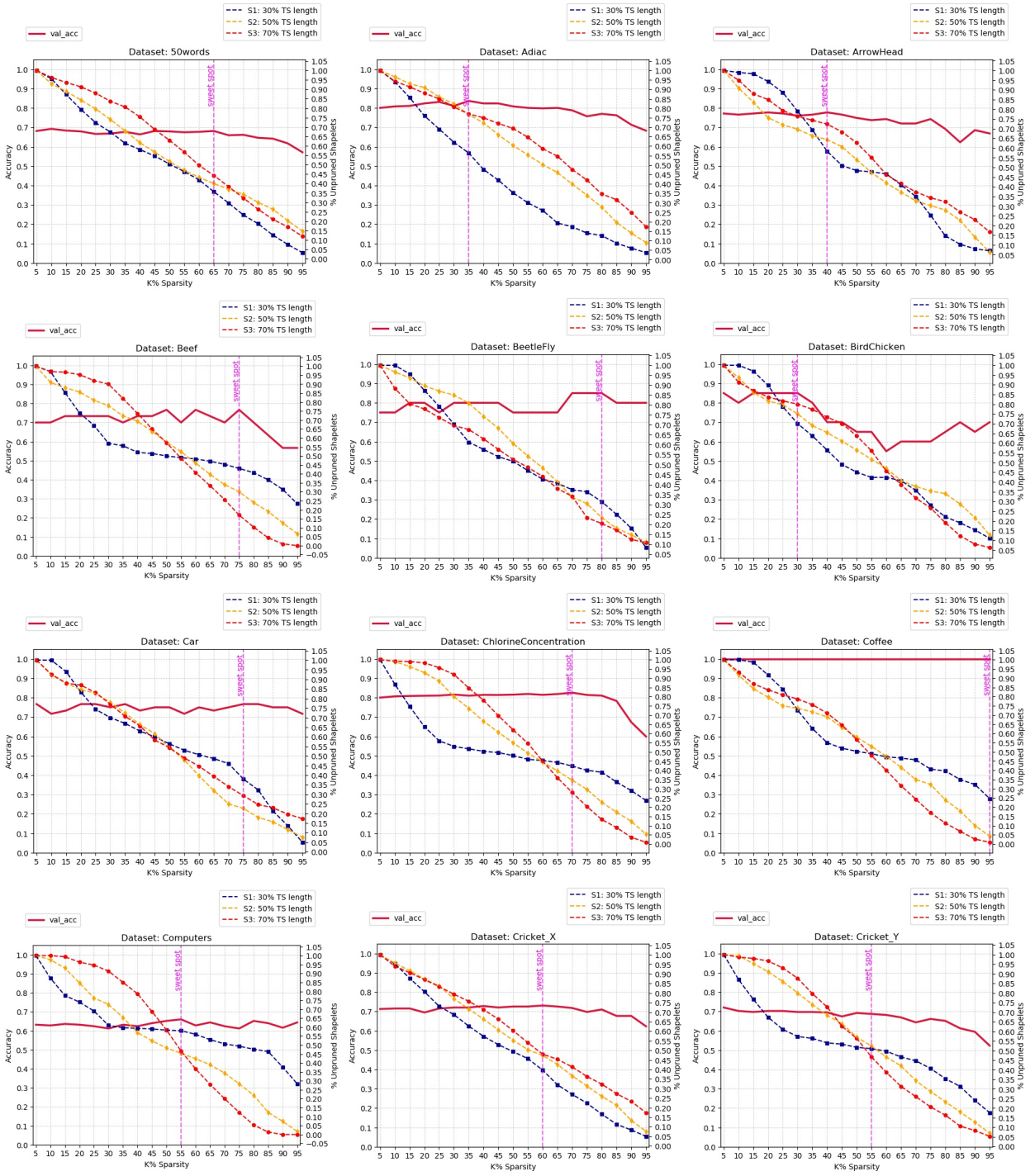



**Figure 3.10:** Validation Accuracy with respect to K% Sparsity and sparsity curves for the three convolutional blocks corresponding to the three shapelets lengths for 50words, Adiac, ArrowHead, Beef, BeetleFly, BirdChicken, Car, ChlorineConcentration, Coffee, Computers, Cricket_X, Cricket_Y.

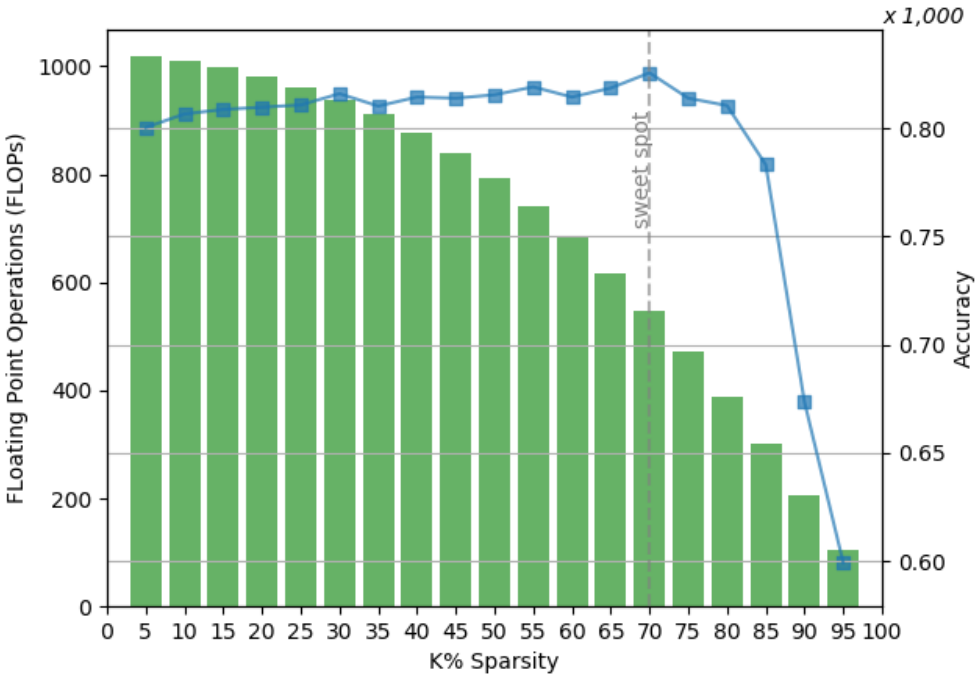



**Figure 3.11:** Floating Points Operations per Second (FLOPs) with respect to network sparsity

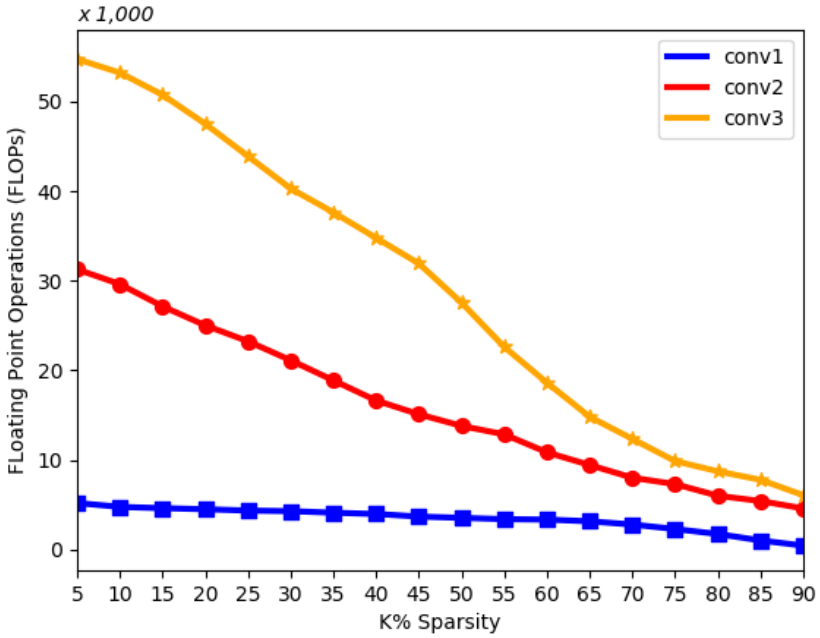



**Figure 3.12:** Floating Point Operations with the three shapelets lengths categories with respect to the network sparsity

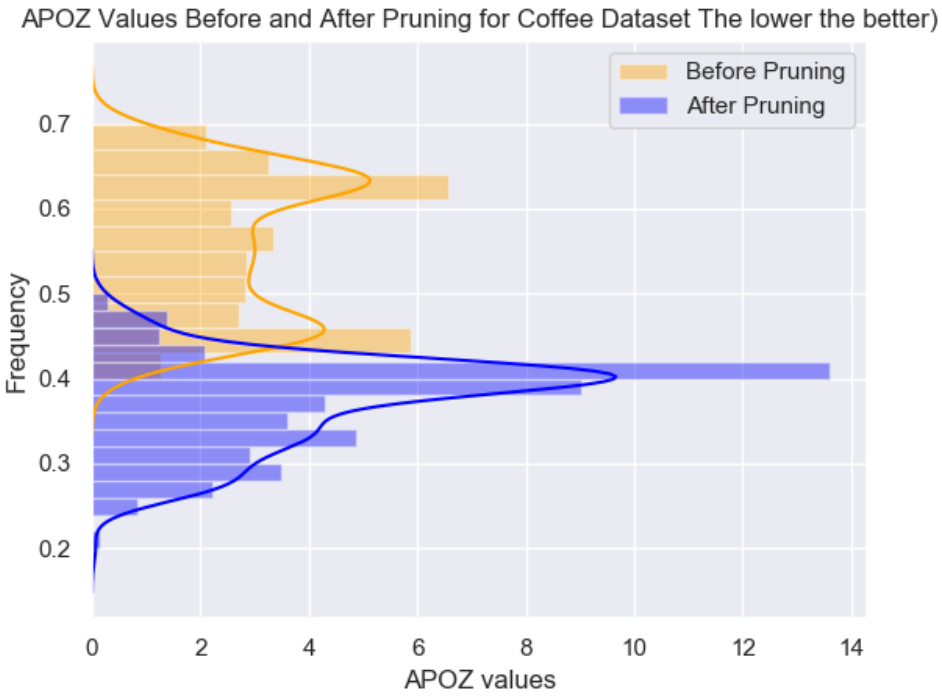



**Figure 3.13:** APOZ Values Before and After Pruning for Coffee Dataset The lower the better)

of shapelet-based querying. As discussed in Section 4, mining accurate shapelets that are truly modeling one subset class of the dataset is achieved through iterative pruning and fine-tuning of the network.

One of the prominent measures of success of the pruning is the declining average percentage of zero activation from one pruning iteration to the next increment. APoZ is a relative measure indirectly proportional to the number of remaining unpruned shapelets in the model, as shown in Equations 3.4 and 3.5. Since it is a normalized measure, it is fair to compare the values of APoZ from two different models. Figure 3.13 shows the distribution of the APoZ values of models from the two extremes, the original unpruned model (in orange) and a model pruned with the maximal percentage of pruned shapelets 95% (in purple). The first observation that can be made is that the mean of the distribution of the pruned model is more than three standard deviations away from the mean of the original unpruned model. This suggests that there is a statistically significant difference

between the two means. The shift of the pruned distribution to the bottom values is indicative of the significant drop in the network redundancy. Also, the diminishing width (standard deviation) of the distribution was diminished from the original unpruned to the pruned APoZ distributions which are indicative that the weak neurons are vanishing, a proof of the benefit gained from weight initialization as discussed in Section 3.3. The second observation that can be made is that the original unpruned model follows a bimodal distribution having the first mode peak at 0.63 and another mode at 0.45. This suggests that the at the end of the training of the original model there are two families of shapelets, the ones with high contribution to the network decision and the ones with low support. in other terms, there still exists room of improvement of the original network based on the wide and binomial nature of the distribution of its APoZ values. The 95%-sparsely pruned network follows a balanced Gaussian distribution with a clear mode of 0.41. Simply put, more than half of the shapelets are significantly contributing to the model by having non-zero activations.

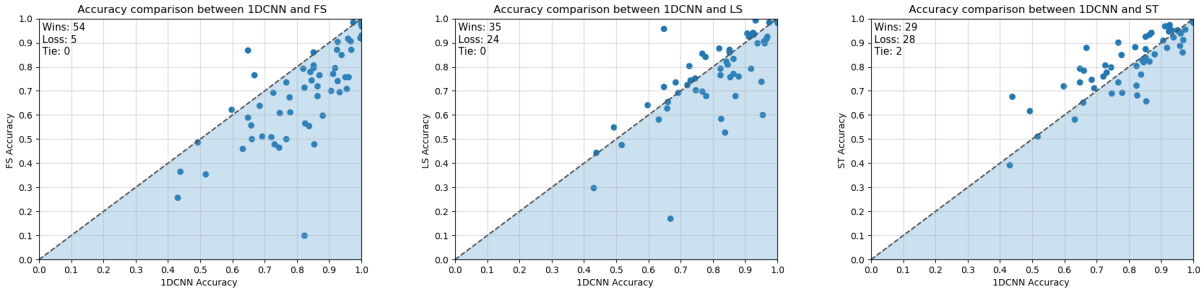



**Figure 3.14:** Accuracy results obtained from pruned 1DCNN classifier with optimal $\mathcal{K}$ parameter, in comparison with FS, LS and ST classifiers.

It is important to understand that an optimally pruned model has a good balance of simplicity (fewer parameters) and good accuracy levels, a crucial pre-requisite for precise querying. Figure 3.10 shows the learning curves of the 50words, Adiac, ArrowHead, Beef, BeetleFly, BirdChicken, Car, ChlorineConcentration, Coffee, Computers, Cricket_X, Cricket_Y datasets. In this context, the model complexity represents the sparsity level (K) of the 1DCNN algorithm. Ideally, a good K parameter is the one where the model

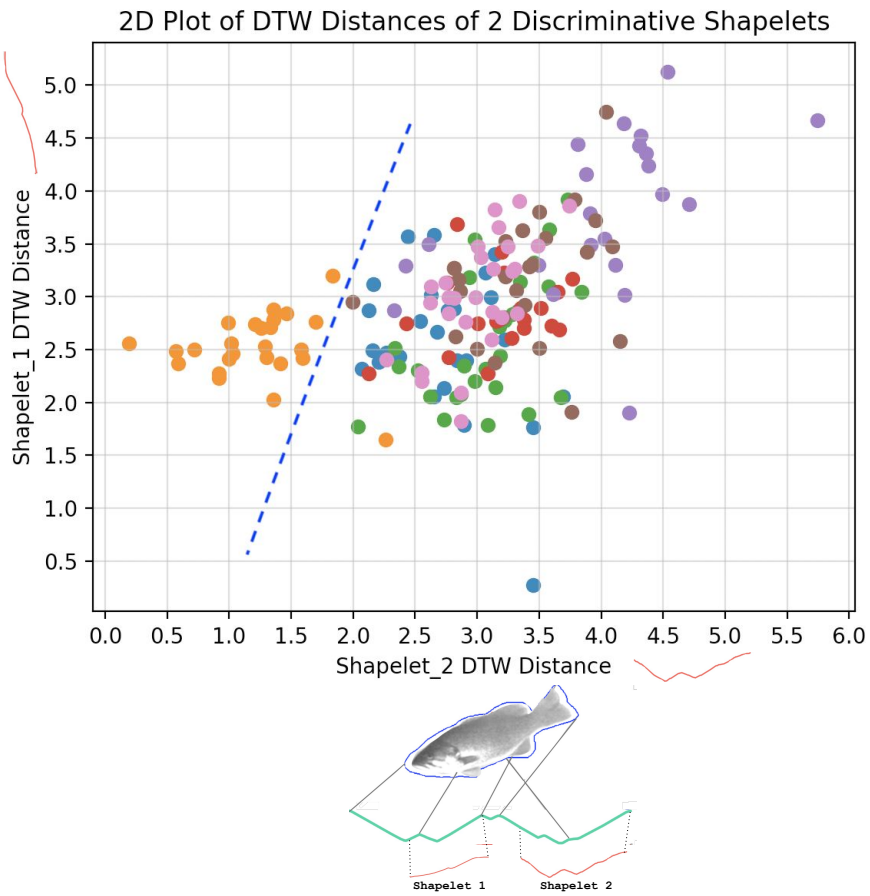



**Figure 3.15:** Dynamic Time Warping Distances of the two most discriminative shapelets for each class with respect to all instances.

sparsity is maximized without compromising accuracy. The vertical line in Figure 3.10 shows a good cut-off point for the learning curves. Figures 7.1, 7.2, 7.3, and 7.4 shows the learning curves of all the datasets we used in our experiment. Upon the completion of all experiments, we noticed that there are three categories of learning curves. The first category of learning curves is characterized by a model that is equally accurate regardless of its sparsity level. The Coffee dataset is a good example of this pattern. Maintaining the same accuracy level of the original unpruned model with reduced complexity is the desired outcome since the model complexity is drastically reduced, making the model more robust and with better ability to generalize and produce more accurate shapelets. The second family of learning curves shows a pattern of accuracy improvement as the network is sparsified. The BeetleFly dataset shows a maximum improvement of 10% from

the original network accuracy at sparsity level 80%. This category of datasets is benefiting from the network sparsity by reaching unprecedented accuracy levels. This behavior suggests that unimportant shapelets were not simply ignored (by zeroing out their activations) but rather hindering the maximal learning potential of the model. Finally, the last learning curve behavior, which is the most intuitive one, is accuracy levels having a downward trend with increasing sparsity (model simplicity). ChlorineConcentration datasets show an example of this behavior where the accuracy starts dropping starting from the $\mathcal{K}$=60% mark where the model starts deteriorating and indicating the model is too simple. Figure 3.10 also shows the decreasing percentage of shapelets from each of the three-length categories as the model is being pruned. It is clear that there is no shapelet category length that is superior over another category. All three lengths of shapelets are grasping important discriminative patterns, although, the shortest shapelet lengths (in blue) show more resistance to the pruning in the range of $\mathcal{K}$=[40,60] for the three datasets. This could be explained that the shortest shapelets are more prone to model highly granular noise that is falsely deemed as important. The fourth column of Table 7.1 ($\mathcal{K}$) shows the optimal sparsity percentage level corresponding to the vertical sweet spot line in Figure 3.10. The table does not show any particular correlations between the number of classes, dataset size, time series length with the optimal sparsity level. This means that network pruning is a data-driven process and there is no size fits all $\mathcal{K}$ parameter.

Another common way of quantifying the complexity of the neural network is to measure the floating-point operations(addition, subtraction, multiplication, or division) per second (FLoPs) required by the model. A prominent benefit of network pruning is the reduction of the number of convolutional feature maps and as therefore the total estimated floating-point operations (FLoPs). Figure 3.11 shows the FLoPs for each model with different sparsity levels and their corresponding accuracy. The optimal $\mathcal{K}$ in this dataset (ChlorineConcentration) has been identified as 70%. Having a 70% pruned model saves almost half ($\approx$ 42%) of the total operations required by the original unpruned

model. It is also worth mentioning that the FLoPs are highly dependent on the shapelet lengths that exist in the network. In the case where longer lengths shapelets are pruned on a higher rate than other smaller lengths shapelets, the FLoPs are reduced with a higher rate as well. Figure 3.12 illustrates the FLoPs operations required by each convolutional layer as the network is being sparsified. To further illustrate the idea, we can clearly see from Figure 3.12 that the first convolutional block has a smaller negative slope compared to the second and third convolutional layers. The linear scale of the figure realistically illustrates the rate of change for each shapelet's length.

We compared our proposed 1DCNN models, pruned with the optimal sparsity levels found experimentally, to three baseline models pertaining to the shapelets classifiers category: Fast Shapelets [50], Learning Shapelets [52], and Shapelet Transforms [11]. The blue area of Figure 3.14 illustrates the area where our model is outperforming the baseline counterpart. Every point in the figure represents one of the datasets defined in Table 7.1 in the appendix section. The more points are in the diagonal, the less significant is the difference in performance between the two baselines. Considering the total number of datasets, our model is more accurate than all the three baselines. It is particularly outperforming FS and LS baselines with a (54 wins/5 losses) and (35 wins/24 losses) respectively. Furthermore, we can see that the points in the blue area are more scattered and far from the diagonal which suggests that there is an important magnitude difference.

The second byproduct of our model in the mined shapelets that are the bases of shapelet-based querying. So far, we have assessed the performance of the 1DCNN model based solely quantitatively through FLoP, accuracy, and APOZ measures. To ensure that the mined shapelets are equally precise, we conducted another experiment to quantify the degree of closeness of the mined shapelet pattern to the original time series instances in the dataset. To do so we used Dynamic Time Warping to measure the shape differences.

In this context, we used DTW to compute the distance between the mined shapelets and all the instances in the time series dataset. This step is useful to show the mined

shapelet relative closeness to a subset of classes and distantness to another subset of classes. For the sake of simplicity, we showed an example of output from the FISH dataset that has 7 classes. We used the pruned model at optimal 75% sparsity level and the APOZ heuristic to sort the mined shapelets in the model and selected the two most discriminative shapelets for 2 distinct classes. Figure 3.15 shows a plot of distances between the two most significant shapelets. The dashed line shows the separation between the orange class instances and the rest of the dataset instances.

**Table 3.2:** Error rates of the 1DCNN variants compared to other Deep learning methods baselines on 40 datasets.

| # | Dataset | MLP | FCN | ResNet | 1DCNN | 1DCNN_l | 1DCNN_w |
|---|---|---|---|---|---|---|---|
| 0 | 50words | 0.29 | 0.32 | 0.27 | 0.42 | 0.43 | 0.42 |
| 1 | Adiac | 0.25 | 0.14 | 0.17 | 0.21 | 0.21 | 0.22 |
| 2 | Beef | 0.17 | 0.25 | 0.23 | 0.21 | 0.5 | 0.29 |
| 3 | CBF | 0.14 | 0.0 | 0.01 | 0.0 | 0.01 | 0.09 |
| 4 | ChlorineConc | 0.13 | 0.16 | 0.17 | 0.08 | 0.17 | 0.1 |
| 5 | Coffee | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 6 | Cricket_Z | 0.41 | 0.19 | 0.19 | 0.45 | 0.47 | 0.47 |
| 7 | DiatomSizeR | 0.04 | 0.07 | 0.07 | 0.0 | 0.1 | 0.0 |
| 8 | ECGFiveDays | 0.03 | 0.02 | 0.04 | 0.0 | 0.04 | 0.0 |
| 9 | FaceAll | 0.12 | 0.07 | 0.17 | 0.08 | 0.08 | 0.08 |
| 10 | FaceFour | 0.17 | 0.07 | 0.07 | 0.38 | 0.5 | 0.37 |
| 11 | FacesUCR | 0.18 | 0.05 | 0.04 | 0.1 | 0.1 | 0.11 |
| 12 | Haptics | 0.54 | 0.45 | 0.5 | 0.58 | 0.57 | 0.57 |
| 13 | InlineSkate | 0.65 | 0.59 | 0.64 | 0.6 | 0.61 | 0.6 |
| 14 | ItalyPowerD | 0.03 | 0.03 | 0.04 | 0.03 | 0.03 | 0.03 |
| 15 | Lighting7 | 0.36 | 0.14 | 0.16 | 0.29 | 0.3 | 0.32 |
| 16 | MALLAT | 0.06 | 0.02 | 0.02 | 0.02 | 0.06 | 0.03 |
| 17 | MedicalImages | 0.27 | 0.21 | 0.23 | 0.24 | 0.24 | 0.25 |
| 18 | NonInv_T1 | 0.06 | 0.04 | 0.05 | 0.1 | 0.09 | 0.09 |
| 19 | NonInv_T2 | 0.06 | 0.04 | 0.05 | 0.09 | 0.08 | 0.08 |
| 20 | OSULeaf | 0.43 | 0.01 | 0.02 | 0.17 | 0.2 | 0.2 |
| 21 | OliveOil | 0.6 | 0.17 | 0.13 | 0.24 | 0.6 | 0.24 |
| 22 | SonyAII | 0.27 | 0.03 | 0.02 | 0.0 | 0.04 | 0.0 |
| 23 | SonyAIII | 0.27 | 0.03 | 0.02 | 0.0 | 0.04 | 0.0 |
| 24 | SwedishLeaf | 0.11 | 0.03 | 0.04 | 0.06 | 0.06 | 0.07 |
| 25 | Symbols | 0.15 | 0.04 | 0.13 | 0.0 | 0.15 | 0.0 |
| 26 | Trace | 0.18 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 27 | TwoLeadECG | 0.15 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 28 | Two_Patterns | 0.11 | 0.1 | 0.0 | 0.03 | 0.06 | 0.04 |
| 29 | WordsSynonyms | 0.41 | 0.42 | 0.37 | 0.45 | 0.46 | 0.46 |
| 30 | synthetic_control | 0.05 | 0.01 | 0.0 | 0.0 | 0.0 | 0.01 |
| 31 | WaveGest_X | 0.23 | 0.25 | 0.21 | 0.33 | 0.36 | 0.33 |
| 32 | WaveGest_Y | 0.3 | 0.28 | 0.33 | 0.39 | 0.42 | 0.4 |
| 33 | WaveGest_Z | 0.3 | 0.27 | 0.24 | 0.33 | 0.35 | 0.34 |
| 34 | wafer | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 35 | yoga | 0.14 | 0.16 | 0.14 | 0.05 | 0.25 | 0.12 |
| 36 | CinC_ECG_torso | 0.16 | 0.19 | 0.23 | 0.22 | 0.35 | 0.26 |
| 37 | Gun_Point | 0.07 | 0.0 | 0.01 | 0.0 | 0.01 | 0.04 |
| 38 | Lighting2 | 0.28 | 0.2 | 0.25 | 0.37 | 0.41 | 0.42 |
| 39 | StarLightCurves | 0.04 | 0.03 | 0.03 | 0.0 | 0.05 | 0.0 |
| **Wins** | - | 2 | 14 | 7 | **17** | 0 | 0 |

**Table 3.3:** Descritpion of 73 Time Series Datasets and Accuracy of Baselines.

| # | Dataset | Train | Test | Len | $\mathcal{L}$ | LS | FS | MLP_n | 1DCNN | 1DCNN_l | 1DCNN_w |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 50words | 450.0 | 455.0 | 270.0 | 50.0 | 0.694 | 0.512 | 0.966 | 0.579 | 0.578 | 0.574 |
| 1 | Adiac | 390.0 | 391.0 | 176.0 | 37.0 | 0.527 | 0.555 | 0.729 | 0.78 | 0.788 | 0.747 |
| 2 | ArrowHead | 36.0 | 175.0 | 251.0 | 3.0 | 0.841 | 0.675 | 0.788 | 0.811 | 0.743 | 0.806 |
| 3 | Beef | 30.0 | 30.0 | 470.0 | 5.0 | 0.698 | 0.502 | 0.739 | 0.733 | 0.5 | 0.733 |
| 4 | BeetleFly | 20.0 | 20.0 | 512.0 | 2.0 | 0.862 | 0.796 | 0.764 | 0.95 | 0.7 | 0.65 |
| 5 | BirdChicken | 20.0 | 20.0 | 512.0 | 2.0 | 0.864 | 0.862 | 0.708 | 1.0 | 1.0 | 0.95 |
| 6 | CBF | 30.0 | 900.0 | 128.0 | 3.0 | 0.977 | 0.924 | 0.868 | 0.999 | 0.992 | 0.88 |
| 7 | Car | 60.0 | 60.0 | 577.0 | 4.0 | 0.856 | 0.736 | 0.78 | 0.817 | 0.633 | 0.767 |
| 8 | ChlorineC | 467.0 | 3840.0 | 166.0 | 3.0 | 0.586 | 0.566 | 0.87 | 0.842 | 0.827 | 0.863 |
| 9 | Coffee | 28.0 | 28.0 | 286.0 | 2.0 | 0.995 | 0.917 | 0.462 | 1.0 | 1.0 | 1.0 |
| 10 | Computers | 250.0 | 250.0 | 720.0 | 2.0 | 0.654 | 0.5 | 1.0 | 0.672 | 0.64 | 0.696 |
| 11 | Cricket_Z | 390.0 | 390.0 | 300.0 | 12.0 | 0.754 | 0.466 | 0.862 | 0.554 | 0.534 | 0.526 |
| 12 | DiatomSizeReduc. | 16.0 | 306.0 | 345.0 | 4.0 | 0.927 | 0.873 | 0.511 | 0.967 | 0.902 | 0.971 |
| 13 | DistalPhalanxA | 400.0 | 139.0 | 80.0 | 3.0 | 0.81 | 0.745 | 0.506 | 0.837 | 0.82 | 0.785 |
| 14 | DistalPhalanx | 600.0 | 276.0 | 80.0 | 2.0 | 0.822 | 0.78 | 0.499 | 0.884 | 0.883 | 0.82 |
| 15 | DistalPhalanxTW | 400.0 | 139.0 | 80.0 | 6.0 | 0.659 | 0.623 | 0.501 | 0.765 | 0.767 | 0.735 |
| 16 | ECG200 | 100.0 | 100.0 | 96.0 | 2.0 | 0.871 | 0.806 | 0.796 | 0.845 | 0.87 | 0.85 |
| 17 | ECG5000 | 500.0 | 4500.0 | 140.0 | 5.0 | 0.94 | 0.922 | 0.763 | 0.942 | 0.935 | 0.94 |
| 18 | ECGFiveDays | 23.0 | 861.0 | 136.0 | 2.0 | 0.985 | 0.986 | 0.652 | 1.0 | 1.0 | 0.964 |
| 19 | Earthquakes | 322.0 | 139.0 | 512.0 | 2.0 | 0.742 | 0.747 | 0.946 | 0.805 | 0.801 | 0.798 |
| 20 | ElectricDevices | 8926.0 | 7711.0 | 96.0 | 7.0 | 0.709 | 0.262 | 0.703 | 0.656 | 0.663 | 0.626 |
| 21 | FaceAll | 560.0 | 1690.0 | 131.0 | 14.0 | 0.926 | 0.772 | 0.829 | 0.924 | 0.916 | 0.887 |
| 22 | FaceFour | 24.0 | 88.0 | 350.0 | 4.0 | 0.957 | 0.869 | 0.936 | 0.773 | 0.5 | 0.636 |
| 23 | FacesUCR | 200.0 | 2050.0 | 131.0 | 14.0 | 0.939 | 0.701 | 0.929 | 0.931 | 0.9 | 0.895 |
| 24 | FordA | 3601.0 | 1320.0 | 500.0 | 2.0 | 0.895 | 0.785 | 0.646 | 0.874 | 0.895 | 0.861 |
| 25 | HandOutlines | 1000.0 | 370.0 | 2709.0 | 2.0 | 0.837 | 0.841 | 0.745 | 0.931 | 0.777 | 0.728 |
| 26 | Haptics | 155.0 | 308.0 | 1092.0 | 5.0 | 0.478 | 0.356 | 0.647 | 0.513 | 0.429 | 0.455 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 27 | Herring | 64.0 | 64.0 | 512.0 | 2.0 | 0.628 | 0.558 | 0.865 | 0.664 | 0.68 | 0.656 |
| 28 | InlineSkate | 100.0 | 550.0 | 1882.0 | 7.0 | 0.299 | 0.257 | 0.745 | 0.435 | 0.389 | 0.284 |
| 29 | InsectWingb | 220.0 | 1980.0 | 256.0 | 11.0 | 0.55 | 0.488 | 0.727 | 0.593 | 0.355 | 0.553 |
| 30 | ItalyPowerDemand | 67.0 | 1029.0 | 24.0 | 2.0 | 0.952 | 0.909 | 0.925 | 0.968 | 0.974 | 0.969 |
| 31 | LargeKitchenA | 375.0 | 375.0 | 720.0 | 3.0 | 0.765 | 0.419 | 0.841 | 0.784 | 0.803 | 0.747 |
| 32 | Lighting7 | 70.0 | 73.0 | 319.0 | 7.0 | 0.765 | 0.101 | 0.53 | 0.713 | 0.685 | 0.699 |
| 33 | MALLAT | 55.0 | 2345.0 | 1024.0 | 8.0 | 0.951 | 0.893 | 0.535 | 0.98 | 0.968 | 0.937 |
| 34 | Meat | 60.0 | 60.0 | 448.0 | 3.0 | 0.814 | 0.924 | 0.696 | 0.883 | 0.683 | 0.933 |
| 35 | MedicalImages | 381.0 | 760.0 | 99.0 | 10.0 | 0.704 | 0.609 | 0.445 | 0.713 | 0.758 | 0.699 |
| 36 | MiddlePAgeGroup | 400.0 | 154.0 | 80.0 | 3.0 | 0.679 | 0.613 | 0.581 | 0.76 | 0.757 | 0.72 |
| 37 | MiddlePhalanxTW | 399.0 | 154.0 | 80.0 | 6.0 | 0.54 | 0.519 | 0.594 | 0.599 | 0.604 | 0.602 |
| 38 | NonInv_Thorax1 | 1800.0 | 1965.0 | 750.0 | 42.0 | 0.6 | 0.71 | 0.831 | 0.899 | 0.913 | 0.913 |
| 39 | NonInv_Thorax2 | 1800.0 | 1965.0 | 750.0 | 42.0 | 0.739 | 0.758 | 0.926 | 0.908 | 0.917 | 0.919 |
| 40 | OSULeaf | 200.0 | 242.0 | 427.0 | 6.0 | 0.771 | 0.679 | 0.702 | 0.802 | 0.798 | 0.719 |
| 41 | OliveOil | 30.0 | 30.0 | 570.0 | 4.0 | 0.172 | 0.765 | 0.357 | 0.7 | 0.4 | 0.433 |
| 42 | PhalangesOCorrect | 1800.0 | 858.0 | 80.0 | 2.0 | 0.783 | 0.73 | 0.616 | 0.858 | 0.778 | 0.841 |
| 43 | Phoneme | 214.0 | 1896.0 | 1024.0 | 39.0 | 0.152 | 0.173 | 0.871 | 0.204 | 0.204 | 0.202 |
| 44 | Plane | 105.0 | 105.0 | 144.0 | 7.0 | 0.995 | 0.97 | 0.999 | 0.99 | 0.987 | 0.99 |
| 45 | ProximalPhalOut | 400.0 | 205.0 | 80.0 | 3.0 | 0.832 | 0.797 | 0.666 | 0.866 | 0.856 | 0.863 |
| 46 | ProximalPhaCorrect | 600.0 | 291.0 | 80.0 | 2.0 | 0.793 | 0.797 | 0.818 | 1.0 | 0.936 | 0.931 |
| 47 | ProximalPhalanxTW | 400.0 | 205.0 | 80.0 | 6.0 | 0.794 | 0.716 | 0.574 | 0.8 | 0.8 | 0.8 |
| 48 | RefrigerationD | 375.0 | 375.0 | 720.0 | 3.0 | 0.642 | 0.574 | 0.516 | 0.473 | 0.451 | 0.448 |
| 49 | ScreenType | 375.0 | 375.0 | 720.0 | 3.0 | 0.445 | 0.365 | 0.85 | 0.471 | 0.432 | 0.461 |
| 50 | ShapeletSim | 20.0 | 180.0 | 500.0 | 2.0 | 0.933 | 1.0 | 0.925 | 0.833 | 0.8 | 1.0 |
| 51 | ShapesAll | 600.0 | 600.0 | 512.0 | 60.0 | 0.76 | 0.598 | 0.94 | 0.791 | 0.796 | 0.795 |
| 52 | SmallKit | 375.0 | 375.0 | 720.0 | 3.0 | 0.663 | 0.333 | 0.897 | 0.692 | 0.659 | 0.651 |
| 53 | SonyII | 27.0 | 953.0 | 65.0 | 2.0 | 0.9 | 0.849 | 0.742 | 1.0 | 1.0 | 0.957 |
| 54 | Strawberry | 613.0 | 370.0 | 235.0 | 2.0 | 0.925 | 0.917 | 0.463 | 1.0 | 0.995 | 0.93 |
| 55 | SwedishLeaf | 500.0 | 625.0 | 128.0 | 15.0 | 0.899 | 0.758 | 0.818 | 0.937 | 0.934 | 0.944 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 56 | Symbols | 25.0 | 995.0 | 398.0 | 6.0 | 0.919 | 0.908 | 0.091 | 1.0 | 1.0 | 0.848 |
| 57 | ToeSegmentation1 | 40.0 | 228.0 | 277.0 | 2.0 | 0.934 | 0.904 | 0.972 | 0.871 | 0.851 | 0.855 |
| 58 | ToeSegmentation2 | 36.0 | 130.0 | 343.0 | 2.0 | 0.943 | 0.873 | 0.868 | 0.959 | 0.955 | 0.762 |
| 59 | Trace | 100.0 | 100.0 | 275.0 | 4.0 | 0.996 | 0.998 | 0.818 | 1.0 | 1.0 | 1.0 |
| 60 | TwoLeadECG | 23.0 | 1139.0 | 82.0 | 2.0 | 0.994 | 0.92 | 0.782 | 1.0 | 1.0 | 0.999 |
| 61 | Two_Patterns | 1000.0 | 4000.0 | 128.0 | 4.0 | 0.994 | 0.696 | 0.932 | 0.969 | 0.965 | 0.937 |
| 62 | UWaveGestureLib | 896.0 | 3582.0 | 945.0 | 8.0 | 0.68 | 0.766 | 0.365 | 0.699 | 0.665 | 0.616 |
| 63 | Wine | 57.0 | 54.0 | 234.0 | 2.0 | 0.524 | 0.794 | 0.389 | 0.5 | 0.602 | 0.5 |
| 64 | WordsSynonyms | 267.0 | 638.0 | 270.0 | 25.0 | 0.581 | 0.461 | 0.599 | 0.549 | 0.543 | 0.536 |
| 65 | Worms | 181.0 | 77.0 | 900.0 | 5.0 | 0.642 | 0.622 | 0.492 | 0.552 | 0.563 | 0.536 |
| 66 | WormsTwoClass | 181.0 | 77.0 | 900.0 | 2.0 | 0.736 | 0.706 | 0.671 | 0.633 | 0.647 | 0.586 |
| 67 | synthetic_control | 300.0 | 300.0 | 60.0 | 6.0 | 0.995 | 0.92 | 0.679 | 0.999 | 0.988 | 0.997 |
| 68 | uWaveGest_X | 896.0 | 3582.0 | 315.0 | 8.0 | 0.804 | 0.694 | 0.961 | 0.672 | 0.668 | 0.636 |
| 69 | uWaveGest_Y | 896.0 | 3582.0 | 315.0 | 8.0 | 0.718 | 0.591 | 0.901 | 0.608 | 0.6 | 0.584 |
| 70 | uWaveGest_Z | 896.0 | 3582.0 | 315.0 | 8.0 | 0.737 | 0.638 | 0.554 | 0.666 | 0.66 | 0.646 |
| 71 | wafer | 1000.0 | 6164.0 | 152.0 | 2.0 | 0.996 | 0.981 | 0.406 | 0.999 | 0.998 | 0.997 |
| 72 | yoga | 300.0 | 3000.0 | 426.0 | 2.0 | 0.833 | 0.721 | 0.827 | 0.951 | 0.883 | 0.749 |
| - | **Wins** | **-** | **-** | **-** | **-** | 14 | 4 | 21 | **25** | 6 | 3 |

# CHAPTER 4

# Neural Network Ensemble (Neuro-Ensemble)

Neuro-Ensemble is a new stacking strategy that uses a batch gradient descent optimization algorithm on a shallow neural network to learn the optimal weight combination of the classifiers. The idea is to embed additional classification logic within every neuron in the neural network. Consequently, the neural network is used solely to learn the weights of the base classifiers on a class level basis. In other words, the network should be able to learn what are the classifiers that are accurate at predicting a given class and assign relatively higher weights to them as compared to their counterparts. The main motivation of this work stems from the idea that different classifiers have different expertise on the input space [58]; therefore, weight optimization should be performed on a class level basis. We evaluated our method on a set of 43 real-world datasets from the University of California at Riverside (UCR) Time Series Classification Archive[1] [28] and 14 vector-based datasets from the University of California Irvine (UCI). We found that the Neuro-Ensemble is competitive with the state-of-the-art ensemble method and the other baseline methods [25, 26, 74].

## 4.1 Neural Network Meta-learning

The goal of the Neuro-Ensemble meta-learner is to model the expertise of each base learner and use it to optimize the final accuracy. Our hypothesis is that some learners are

1 http://www.cs.ucr.edu/~eamonn/time_series_data

more expert in predicting a subset of classes over other base learners. We approach the problem of ensembling from an optimization perspective. Given a set of base learners, the goal is to optimize the cost function and update the weights of each learner based on the class they are the most expert on.

Our method is based on a feedforward Multi-Layer Perceptron (MLP) that we specifically adjusted to be trained on the class probabilities meta-features generated from the level-1 training of the base classifiers. We used a shallow MLP of 3 layers that we trained with backpropagation in conjunction with batch gradient descent optimization algorithm. We defined a new **super-neuron** entity that encapsulates a number of neurons equal to the number of classes in the multi-class classification problem. In addition to encapsulating a number of neurons and performing the activation computations, a super-neuron is embedded with classification logic. In other words, the super-neuron has the ability to classify the samples itself and aggregate its vote with votes (class probabilities) it received from previous layers. All the super-neurons in the network receive the same training data and make their own predictions on it. The input and output of a super-neuron are class probabilities. An illustration of the super-neuron is shown in Figure 4.2-a where every inner neuron represents a class. Conceptually, our proposed approach can be thought of as multiple layers of MLPs trained in parallel, where each MLP layer is specialized in learning the expertise of classifiers in a given class/label. The number of MLP layers corresponds to the number of classes involved in the classification problem. The final ensemble vote constitutes the representative class of the MLP that received the highest probability. Figure 4.1 illustrates the conceptual idea of the Neuro-Ensemble method. It can be noted that at every MLP layer, only 1 class is used for weights optimization at a time. The active neurons within the super-neurons at the level of each layer are shown in red in Figure 4.1.

Traditionally an artificial neuron entity receives a set of inputs $x_i \dots x_n$ weighted with their respective weights $w_i \dots w_n$. The processing involves summing the weighted in-

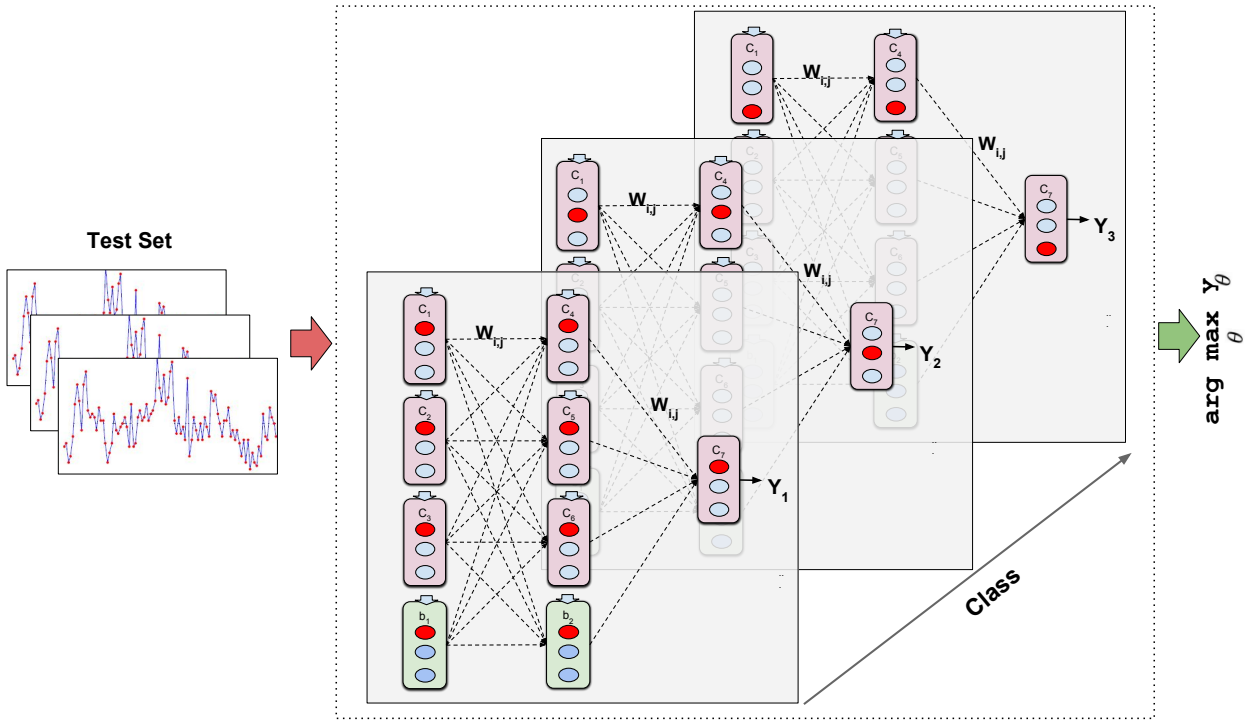
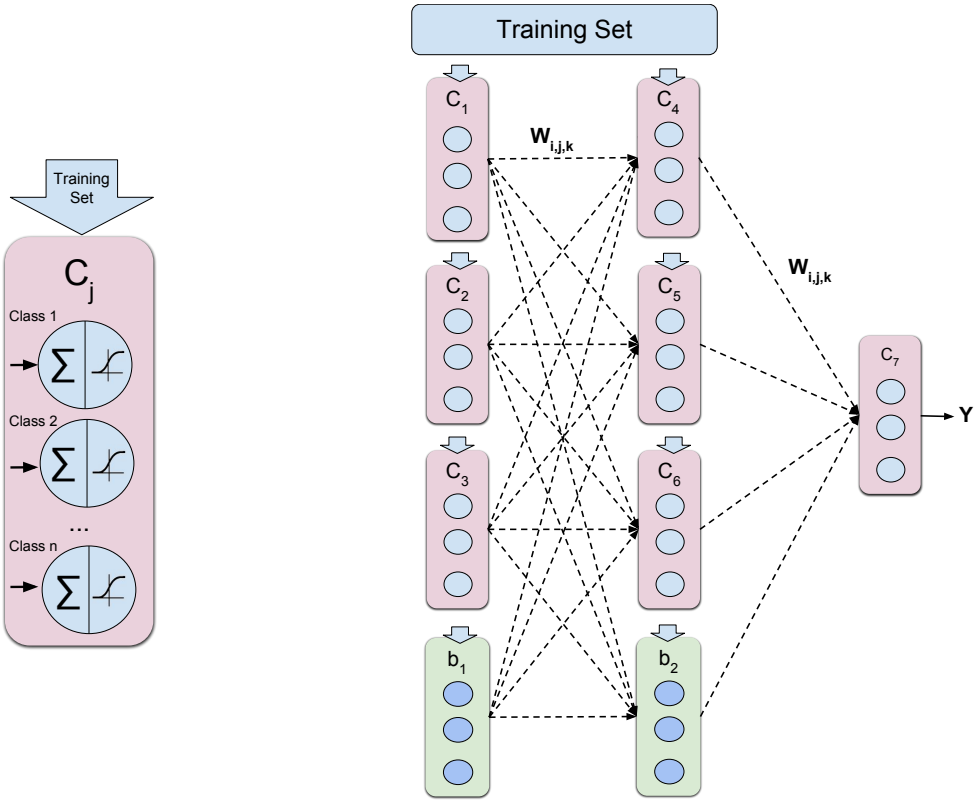



**Figure 4.1:** Neuro-Ensemble Flowchart

put and feeding it to a non-linear activation function. In analogy with a neuron, the super-neuron entity receives a set of input vectors $X_i \ldots X_n$ and their associated weight vectors $W_i \ldots W_n$. The dimension of the vector is equal to the number of classes in the classification problem. The input weighting is done by performing a dot product of the input and weight vectors. At this stage, the super-neuron also participate by its own vote $v$ that is added to the weighted sum vector. A component of the resulting vector is defined in Equation 4.1. After summing the weighted input vectors, the activation function is applied to the new vector component-wise as shown in Equation 4.2. We used a sigmoid function as the activation function (Equation 4.3).

$$n_{jk}^{(\ell)} = \sum_{i=1}^{N_{\ell-1}} a_{ik}^{(\ell-1)} w_{ijk}^{(\ell)} + v_{jk}^{(\ell)} + b_{jk}^{(\ell)}, \qquad (4.1)$$
$$j = 1, 2, \ldots, N_\ell. \qquad k = 1, 2, \ldots, N_c.$$




(a) (b)

**Figure 4.2:** (a) Super-neuron closeup with N neurons corresponding to the number of classes in the multiclass classification problem. Every neuron contains aggregation and activation functions. (b) Neuro-Ensemble overall architecture with one hidden layer and three super-neurons in the input and hidden layers.

$$a_{jk}^{(\ell)} = f^{(\ell)}(n_{jk}^{(\ell)}) = f^{(\ell)}(\sum_{i=1}^{N_{\ell-1}} a_{ik}^{(\ell-1)} w_{ijk}^{(\ell)} + v_{jk}^{(\ell)} + b_{jk}^{(\ell)}). \tag{4.2}$$

$$f^{(\ell)}(n_{jk}^{(\ell)}) = \frac{1}{(1 + e^{(-n_{jk}^{(\ell)})})} \tag{4.3}$$

The class probability vectors are propagated forward through the network using the Equations 4.2-4.3 to obtain the corresponding network output, $\mathbf{y}(t)$. The weights and biases are then adjusted using the batch gradient descent algorithm to minimize the squared error of the training vector in Equation 4.4.

$$E = \|\mathbf{y}(t) - \mathbf{t}(t)\|^2 \tag{4.4}$$

where $\mathbf{t}(t) = \mathbf{t}^{(q)}$ is the corresponding target vector for the chosen training vector $\mathbf{s}^{(q)}$. This square error $E$ is a function of all the weights and biases of the entire network since $\mathbf{y}(t)$ depends on them. The set of updating rules based on the steepest descent algorithm are then applied on the weight and bias vectors as shown in Equation 4.5.

$$\begin{aligned} w_{ijk}^{(\ell)}(t+1) &= w_{ijk}^{(\ell)}(t) - \alpha \frac{\partial E}{\partial w_{ijk}^{(\ell)}(t)} \\ b_{jk}^{(\ell)}(t+1) &= b_{jk}^{(\ell)}(t) - \alpha \frac{\partial E}{\partial b_{jk}^{(\ell)}(t)}, \end{aligned} \tag{4.5}$$

where $\alpha(> 0)$ is the learning rate. To compute these partial derivatives, we need to understand how $E$ depends on the weights and biases. First, $E$ depends explicitly on the network output $\mathbf{y}(t)$ (the activations of the last layer, $\mathbf{a}^{(L)}$), which then depends on the net input into the $L-$th layer, $\mathbf{n}^{(L)}$. In turn $\mathbf{n}^{(L)}$ is given by the activations of the

preceding layer and the weights and biases of layer $l$. The explicit relation is: for brevity, the dependence on step $t$ is omitted

$$\begin{aligned} E &= \|\mathbf{y} - \mathbf{t}(t)\|^2 = \|\mathbf{a}^{(l)} - \mathbf{t}(t)\|^2 = \|f^{(l)}(\mathbf{n}^{(l)}) - \mathbf{t}(t)\|^2 \\ &= \left\| f^{(l)} \left( \sum_{i=1}^{N_{l-1}} a_{ik}^{(\ell-1)} w_{ijk}^{(\ell)} + v_{jk}^{(\ell)} + b_{jk}^{(\ell)} \right) - \mathbf{t}(t) \right\|^2 \end{aligned} \tag{4.6}$$

It is then easy to compute the partial derivatives of $E$ with respect to the elements of $\mathbf{W}^{(L)}$ and $\mathbf{b}^{(L)}$ using the chain rule for differentiation. For a general layer, $\ell$, we can write

$$\frac{\partial E}{\partial w_{ijk}^{(\ell)}} = \sum_{n=1}^{N_\ell} \frac{\partial E}{\partial n_n^{(\ell)}} \frac{\partial n_n^{(\ell)}}{\partial w_{ijk}^{(\ell)}}$$

$$\frac{\partial E}{\partial b_{jk}^{(\ell)}} = \sum_{n=1}^{N_\ell} \frac{\partial E}{\partial n_n^{(\ell)}} \frac{\partial n_n^{(\ell)}}{\partial b_{jk}^{(\ell)}}$$

for $j = 1, 2, \ldots, N_\ell$ and $k = 1, 2, \ldots, N_c$.

An example of a super-neuron based MLP architecture on a 3 class classification problem is shown in Figure 4.2-b. In this paper, we are particularly interested in heterogeneous classifiers fusion. Therefore, every super-neuron embed a different learning algorithm. We used the **best-first** heuristic to order the classifiers in the network. The best-first heuristic places the most accurate base classifiers on the level-1 validation in the hidden layer and the less performant classifiers are placed in the former layer (input layer). The full adjusted batch gradient descent algorithm that we used is shown in Algorithm 4.1.

The next section introduces the base classifiers used by the Neuro-Ensemble that are the same ones used by the Elastic Ensemble [75]. The base learners are all variations of 1-NN classifier coupled with different distance measures and metrics.

**Algorithm 4.1** Adjusted Batch Gradient Descent Algorithm

**Input:** Dataset $\mathcal{D}$, learning rate $\alpha$, number of epochs $ep$
**Output:** The trained $network$

```
1: iteration ← 0
2: while iteration < ep do
3:       ▷ Propagate the input (Forward Pass)
4:     for each vector component c do
5:         for each input layer unit i do
6:             for each hidden layer unit j do
7:                 I_jc = Σ_i W_ijc v_ic + v_jc + θ_jc
8:                 O_jc = 1 / (1 + e^(−I_jc))
9:             end for
10:             ▷ Backpropagate the errors (Backward Pass)
11:            for each hidden layer unit j do
12:                Err_ic = O_jc(1 − O_jc) Σ_k Err_kc W_jkc
13:            end for
14:            for each weight w_ij in the network do
15:                δw_ijc = α Err_jc O_jc
16:                w_ijc = w_ijc + δw_ijc
17:            end for
18:            for each bias θ in the network do
19:                δθ_jc = α Err_jc
20:                θ_jc = θ_jc + δθ_jc
21:            end for
22:        end for
23:    end for
24:    iteration ← iteration + 1
25: end while
26: return network
```

## 4.2 Neuro-Ensemble on Time Series Data:

### *4.2.1 Sampling*

In order to compare the performance of different ensemble algorithms, we have set some desiderata. The first requirement when evaluating an ensemble methodology is that there should be still room for improvement of every base classifier participating in the ensemble. Theoretically, the Naive Bayes error represents the optimal level of learning of a classifier where no more improvement can be made [23]. To make the problem challenging, the base models should be trained with few instances before they reach the Bayes optimal error. In order to meet this requirement, we used a stratified 25% of each dataset for training the base classifiers participating in the ensemble.

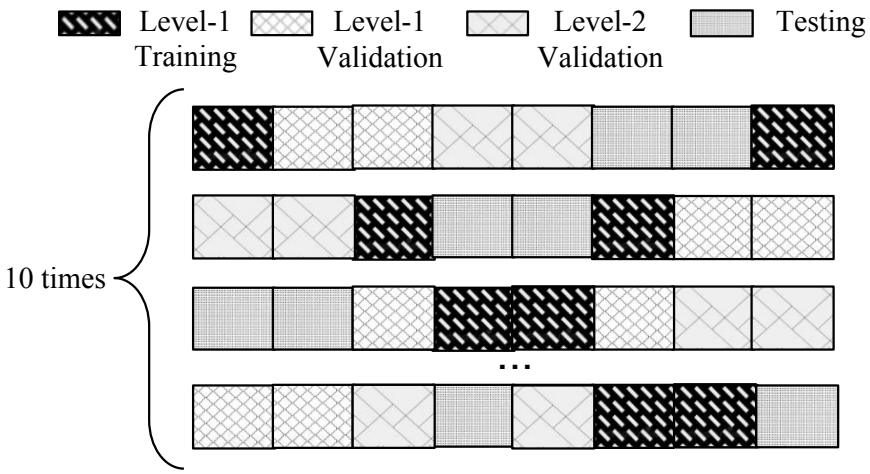



**Figure 4.3:** Sampling methodology (25/25/25/25)

As mentioned earlier, our proposed Neuro-Ensemble meta-learner involves two levels of learning. Level-1 learning consists of training the base learners and validating them. We reserved another stratified holdout sample of 25% of the dataset for validating the level-1 base learners. Level-2 learning involves using the meta-features generated from the level-1 learners and feeding them to the Neuro-Ensemble. In this paper, the meta-features represent the class predictions on the level-1 training and level-1 validation datasets.

In other terms, 50% of the dataset is used for level-2 training. To validate the level-2 meta-learner, we used another stratified holdout sample of 25% of the datasets as a level-2 validation set. Training on a large dataset can reduce the number of instances used for testing, resulting in high variability in the classifier performance [76]. Therefore, we reserved the last quarter of the dataset as the never-seen testing set of the ensemble. The stratified 25/25/25/25 split is repeated 10 times as shown in Figure 4.3.

### 4.2.2 *Experimental Result*

In this section, we will discuss three criteria of the experiments we conducted to show the efficiency of our ensemble from different perspectives. We first discuss the accuracy performance of the Neuro-Ensemble in comparison with the participating base classifiers, we then explore the relationship between time series lengths and the ensemble methodology efficiency. Finally, we compare the overall accuracy and execution times of the Neuro-Ensemble and the Elastic Ensemble method coupled with the *Proportional* voting heuristic.

The scatter plot and boxplot in Figure 4.4 shows the average test accuracy of the Neuro-Ensemble and the 11 participating classifiers on the 10-folds validation sets of each of the 43 datasets respectively. It can be observed that the Neuro-Ensemble is, in most of the cases, superior to its underlying base classifiers which makes it better than the Evaluation and Selection (ES) method. The former selects the most accurate classifier based on their performance on the validation set and use it for the testing. Figure 4.4 shows the performance distribution of all the base classifiers on the test data, which means that even with the optimal choice of the best classifier on the validation set, Neuro-Ensemble is almost always superior to Evaluation and Selection.

In order to make more sense of the improvement difference of NE compared to ES, we evaluate the average performance difference of NE and the base classifiers mean

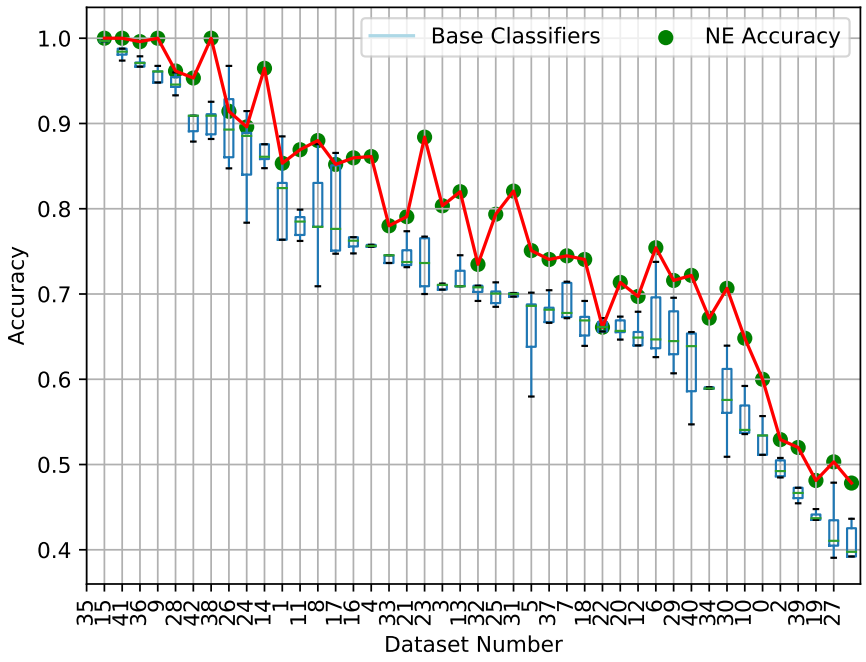



**Figure 4.4:** Average accuracy of NE method and boxplot accuracies of all the 11 participating base classifiers on the 10-fold validation set.

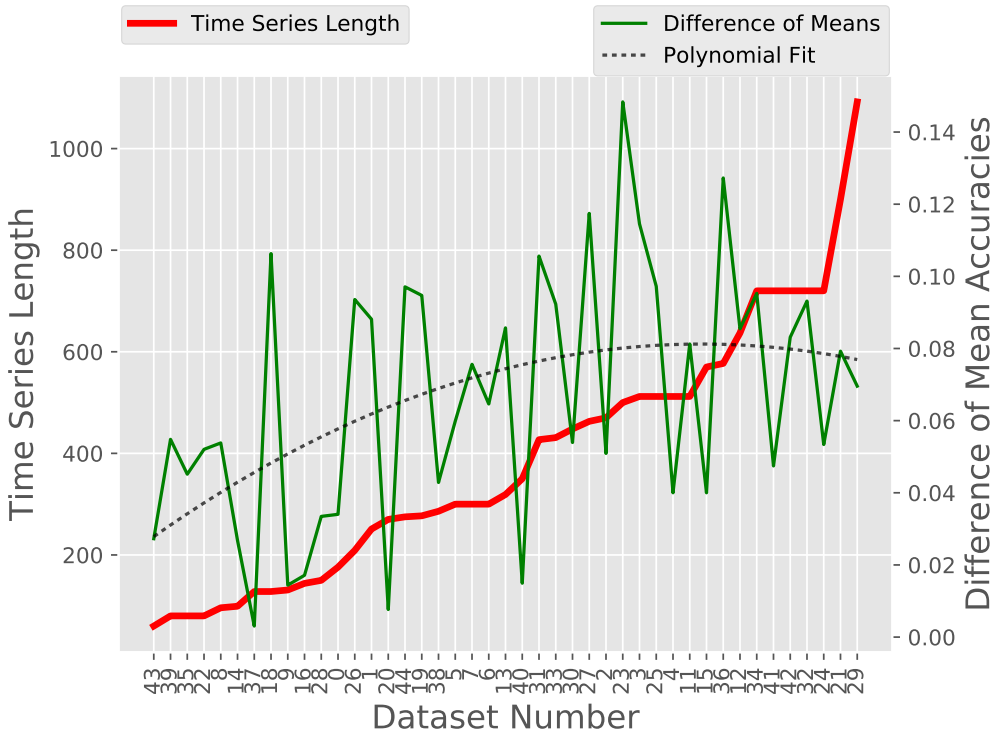



**Figure 4.5:** Average Accuracy Difference of Neuro-Ensemble (NE) method and the mean accuracy of the 11 participating base classifiers

accuracies. Figure 4.5 shows the ordered list of datasets based on their corresponding time series lengths that are shown in the (red) line plot. The second (green) line plot shows the difference of accuracies between NE and the mean accuracies of the base classifiers. The dotted line plot represents a polynomial fit of the difference of means. The first observation that can be made is that there is an upward trend on the polynomial fit that stops at around time series length ($\approx 600$). This suggests that the use of NE is useful for time series datasets whose sequences lengths fall in the range $[40, 600]$. When the time series length is high, the ensemble performance gain is comparatively limited. We explain this phenomenon by the types of base classifiers used in this study. Empirically, it has been shown that shape-based time series classifiers that rely on the use of distance measures are not particularly efficient for long time series [77]. As a result, the base classifiers are reaching their optimal Naive Bayes error quickly which does not leave any room for improvement for the ensemble. This hypothesis needs further validation by using other base classifiers in NE that are better suited for longer time series data such as interval-based and shapelet-based classifiers [78].

In the next round of experiments, we compared our method to the state-of-the-art Elastic Ensemble coupled with *Proportional* voting heuristic with a focus on a subset of datasets where the Neuro-Ensemble has shown to have a significant difference over the use of the best base classifiers. The datasets of focus fall in the upward trend of the polynomial fit line shown in Figure 4.4. We compared our method to the Elastic Ensemble based on their respective training times and test accuracies. Figure 4.8 shows the bar plots representing the training time of the Elastic Ensemble and the stacked bar plots showing the level-1 and level-2 training times of our Neuro-Ensemble. In all of the cases, the total training time of the Neuro-Ensemble is significantly less than the training time of the Elastic Ensemble. NE is at least $3x$ faster than EE and up to $86x$ faster (the plot is in log-scale). Another observation that can be made is that the level-2 training time is proportional to the number of classes of the datasets. As explained earlier, our

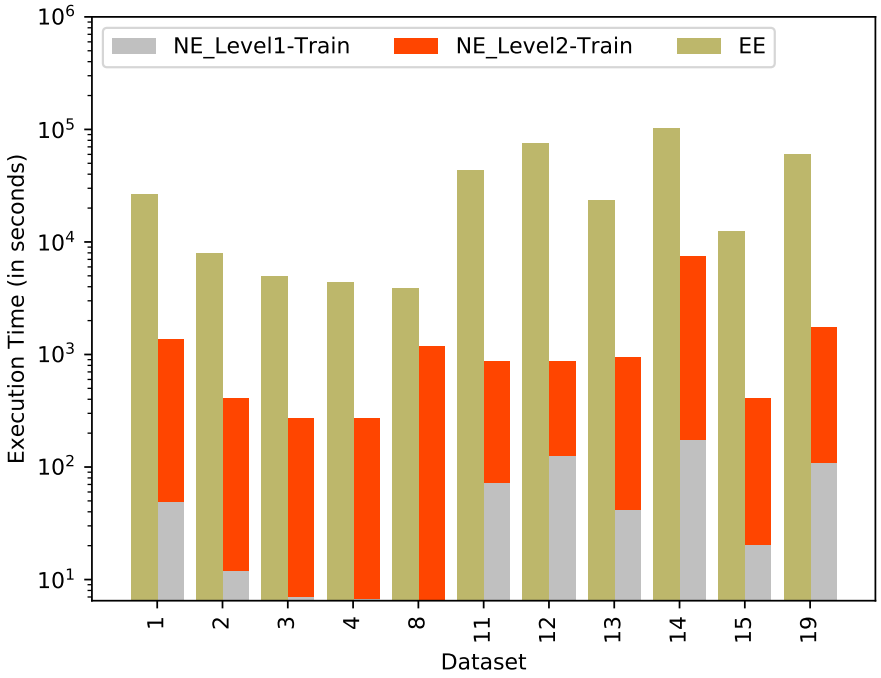



**Figure 4.6:** Training times of NE and EE

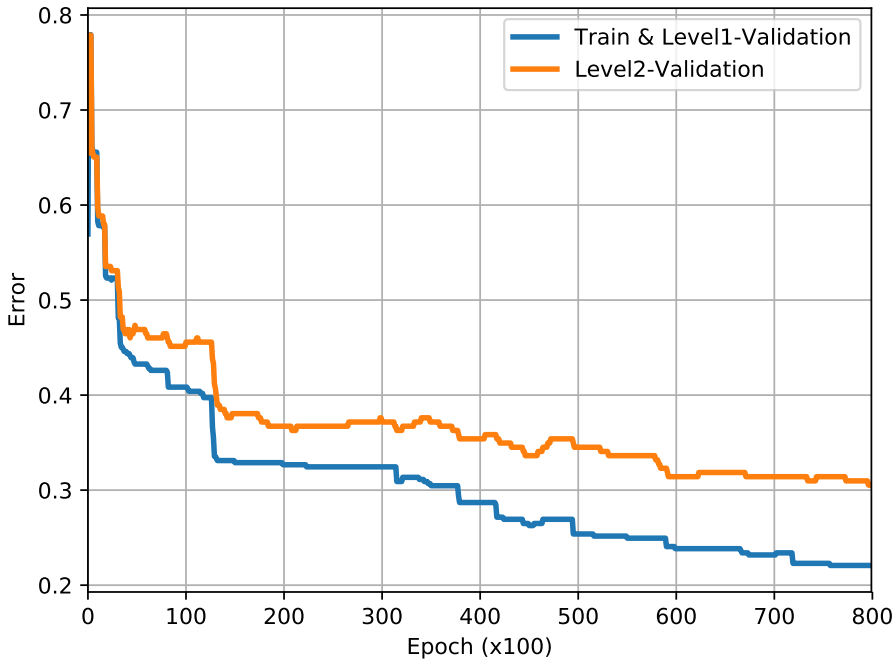



**Figure 4.7:** Learning Curve of the NE on the Beetfly dataset

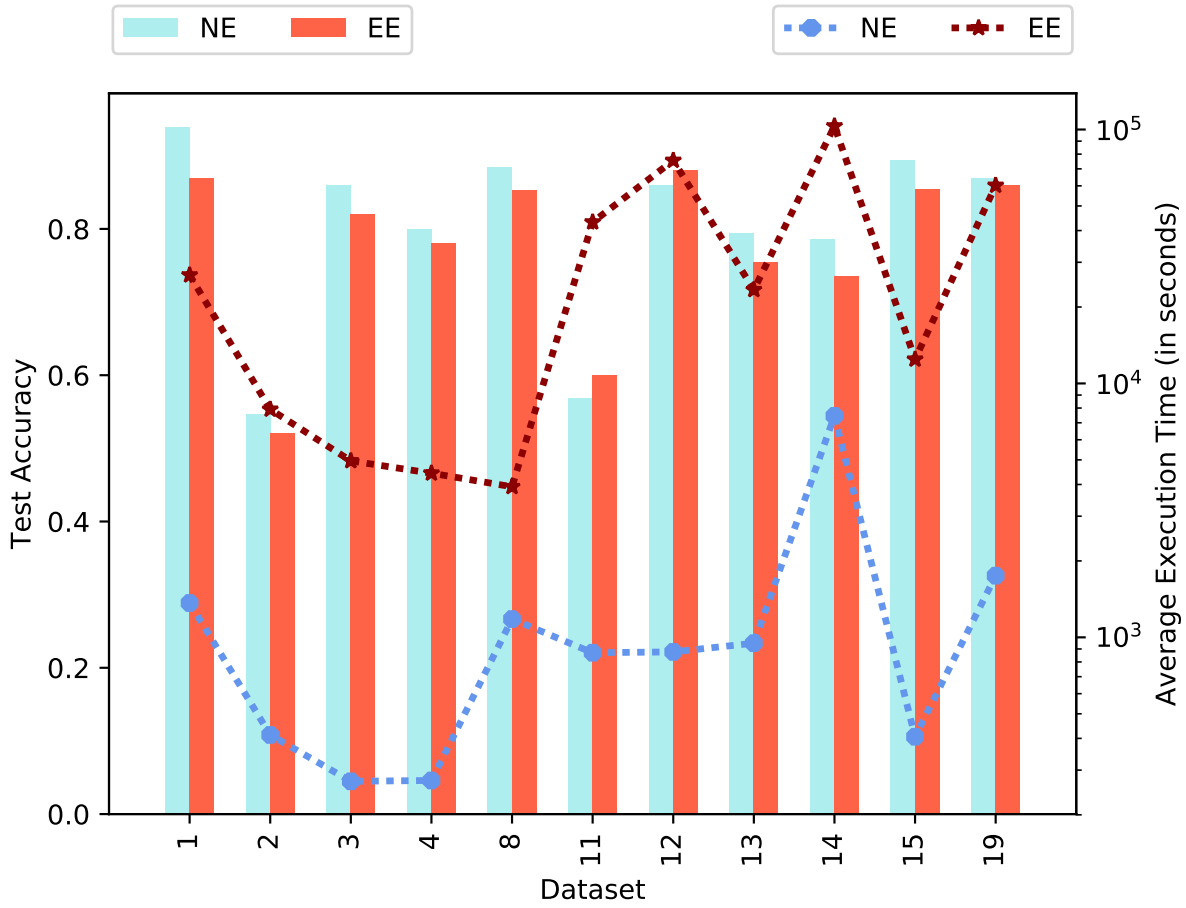



**Figure 4.8:** Test accuracy and average execution time of NE and EE

proposed ensemble can be conceptually thought of a multi-layer perceptron of $\mathcal{L}$ layers as illustrated in Figure 4.1. The more classes the problem has, the more layers exist which necessitate more training. It is also important to note that the level-2 training time of NE is always greater than the level-1 training of the base classifiers as shown in Figure 4.6. This finding is expected since we used $80,000$ epochs to train NE to make sure that we do not prematurely stop the training. An example of learning curves for the *Beetlefly* dataset is shown in Figure 4.7. The curve shows that after $\approx 70,000$ epochs, the validation error stagnates which indicates the sweet spot where training should be ended to avoid overfitting the model.

Figure 4.8 shows the mean accuracy of NE and EE methods with their corresponding training times. NE is generally more accurate than EE and takes significantly less time to

train (up to 86x faster). The highest accuracy improvement was noticed in dataset 1 and dataset 8. After mapping those datasets to their characteristics, we noted that they have a relatively high number of labels. This suggests that NE works particularly better for the case of classification problems with a high number of classes.

## 4.3 Neuro-Ensemble on Vector-Based Data:

### *4.3.1 Base Classifiers and Baseline Methods*

For further validation, we applied Neuro-Ensemble model on vector-based data. The second desired property of this study is to use a relatively small number of classifiers in the ensemble since having a large number of classifiers will result in the same asymptotic error for all the ensembles to be compared [76].

Therefore, we limited the number of base classifiers to six. The methods are used in conjunction with the scikit-learn machine learning library implementations of the following base Classifiers:

- **DT (Gini)**: The Decision Tree Algorithm of Classification and Regression Trees (CART) used with Gini as the splitting criteria [79].
- **DT (Entropy)**: The Decision Tree Algorithm of Classification and Regression Trees (CART) used with entropy as the splitting criteria [79].
- **NB**: The Gaussian Naive Bayes algorithm for classification where the likelihood of the feature is assumed to be Gaussian [80].
- **KNN**: Instance-based learning algorithm with Minkowski distance [81].
- **LR**: Regularized logistic regression liblinear [82] solver that supports both L1 and L2 regularization [83].

- **LDA**: Linear Discriminant Analysis algorithm that models the class conditional distribution of the data $P(X|y = k)$ for each class $k$ [84].

We considered six different baseline methods that we compared with our Neuro-Ensemble meta-learner. The methods are as follows: Evaluation and Selection (ES), Stacking with Decision Tree (Stk(DT)), Stacking with Logistic Regression (Stk(LR)), Weighted Voting (WV), Weighted Voting Per Class (WVC), and Majority Voting (MV).

### *4.3.2 Sampling*

For comparing the performance of different fusion algorithms, we have set some desired properties. The first requirement when evaluating an ensemble methodology is that there should still be room for improvement of every base classifier participating in the ensemble. Theoretically, the Bayes error represents the optimal level of learning of a classifier where no more improvement can be made. To make the problem challenging, the base models should be trained with few instances before they reach the Bayes optimal error [76]. To meet this requirement, we generated learning curves for all the datasets and selected the training dataset size where the learning curves were still decreasing. An example of learning curves of the base classifiers for the Image Segmentation dataset is shown in Figure 4.9 (each tick on the x-axis represents 5% of the dataset).

We can notice that the optimal Bayes error is reached starting from the point where 25% of the samples are used for the training. A good cutoff point is when 10% of the dataset is used for training. It can also be observed that the standard deviation is relatively higher than the one reached at the Bayes error. To avoid variability of train and test split proportions across the datasets, we fixed the training dataset size to be 10% of the total dataset size.

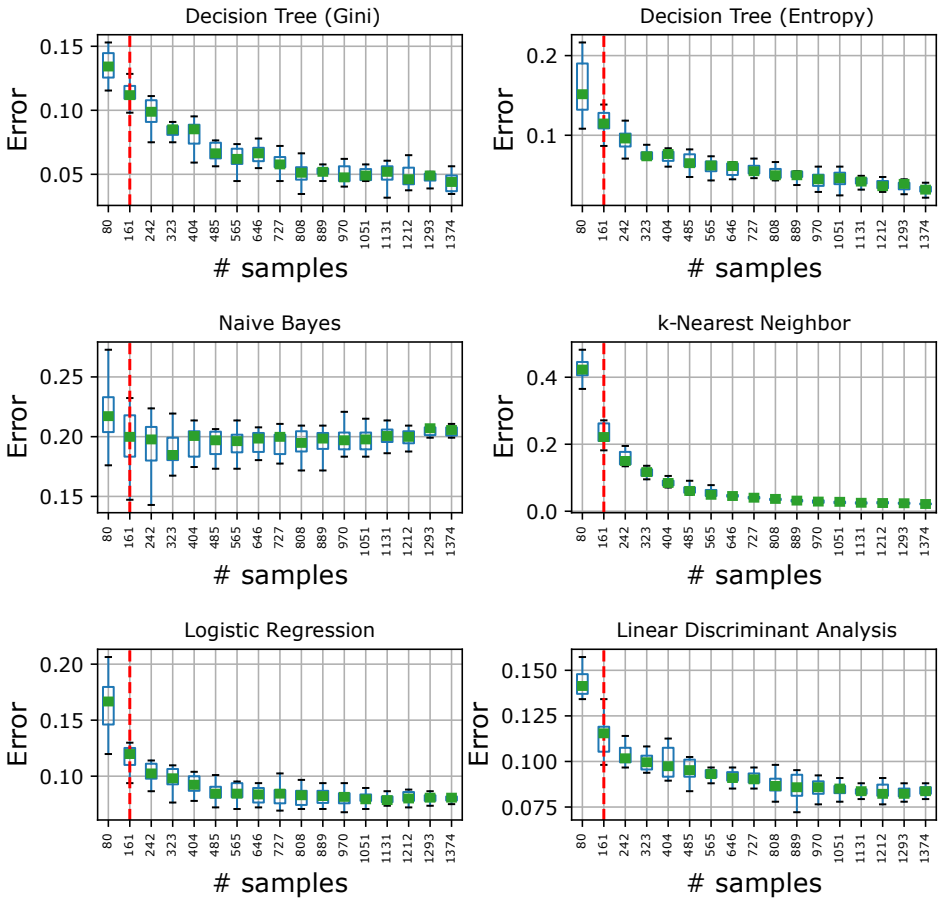



**Figure 4.9:** Learning curves for Image Segmentation dataset showing different behaviors of the base classifiers. Each point represent the mean error rate for 5 runs of experiments tested on the testing set. The vertical line marks the cutoff for the training data used in this paper that meets our desiderate. Each tick represents a 5% increase of the dataset size. It can be noted that the standard deviation is high when the models are trained on small datasets given that the size of the test data is bigger. When models are trained on a larget set of instances, the optimal Bayes error is reached.

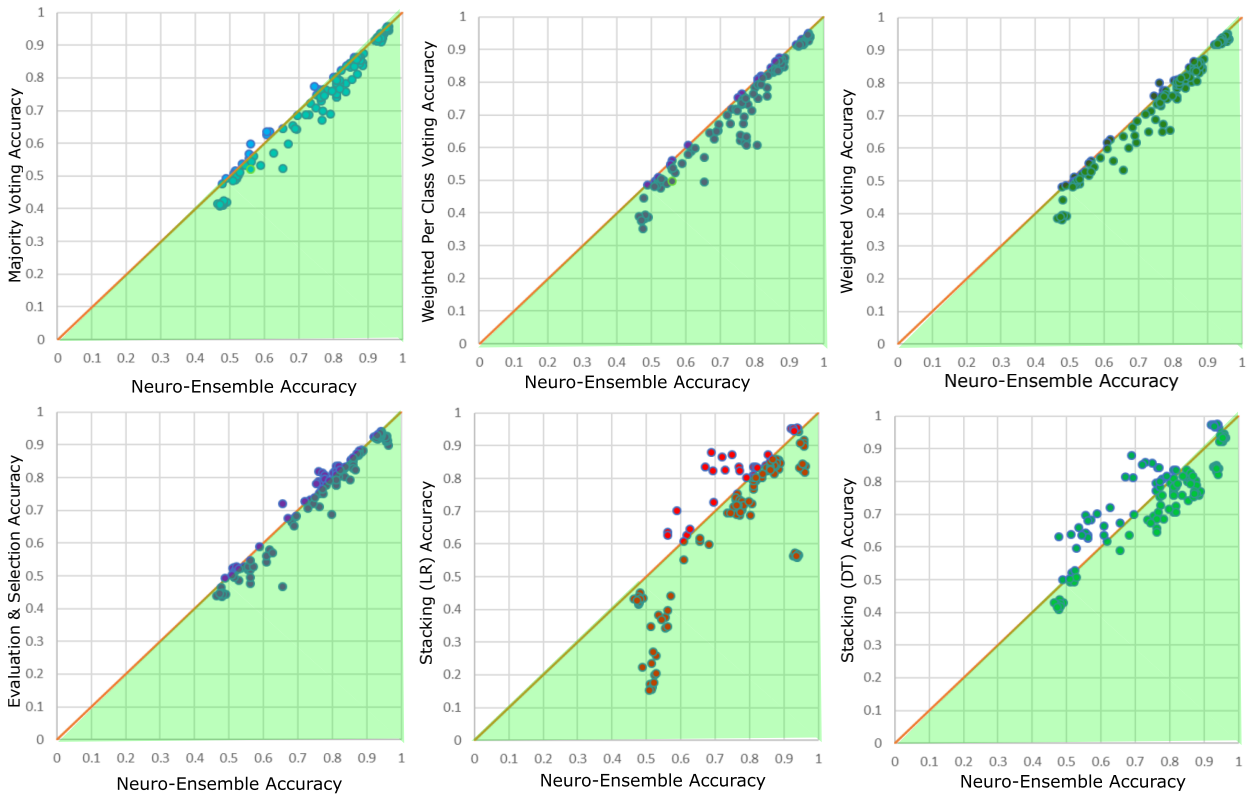



**Figure 4.10:** Neuro-Ensemble compared to the six other baselines. Each point represents one of the 10 test folds of the 14 datasets. The x coordinates represent the Neuro-Ensemble accuracy, and the y coordinates represent the baseline accuracy. The shaded area shows cases where the Neuro-Ensemble wins over the other baseline.

### *4.3.3 Results and Discussion*

To compare the Neuro-Ensemble with other baseline methodologies, we first quantify the performance gain when using the Neuro-Ensemble and then we make a statistical significance test for all the possible pairs of baseline methods using the result of the performance experiments. Figure 4.10 compares the mean accuracy of the 10 runs of experiments. The shaded area shows the cases where the Neuro-Ensemble performs better than the other baseline.

We present the accuracies of NE and other baselines and their corresponding standard deviations on the 14 datasets on our project. To ascertain whether the mean accuracy differences between baseline methods are statistically significant, we used a paired t-test

**Table 4.1:** Significant wins and losses (w:l) for each pair of methods and the total significant wins and losses for each baseline method

| | Stk(DT) | Stk(LR) | ES | WV | WVC | MV | NE | Tot. Wins | Tot. Loss |
|---|---|---|---|---|---|---|---|---|---|
| Stk(DT) | | 6:5 | 5:6 | 6:4 | 6:4 | 4:5 | 3:9 | 30 | **33** |
| Stk(LR) | 5:6 | | 2:8 | 3:6 | 5:5 | 5:6 | 2:10 | 22 | **41** |
| ES | 6:5 | 8:2 | | 8:1 | 6:1 | 8:2 | 1:7 | **37** | 18 |
| WV | 4:6 | 6:3 | 1:8 | | 5:4 | 3:5 | 0:12 | 19 | **38** |
| WVC | 4:6 | 5:5 | 1:6 | 4:5 | | 3:7 | 0:13 | 13 | **42** |
| MV | 5:4 | 6:5 | 2:8 | 5:3 | 7:3 | | 0:12 | 20 | **35** |
| NE | 9:3 | 10:2 | 7:1 | 12:0 | 13:0 | 12:0 | | **54** | 6 |

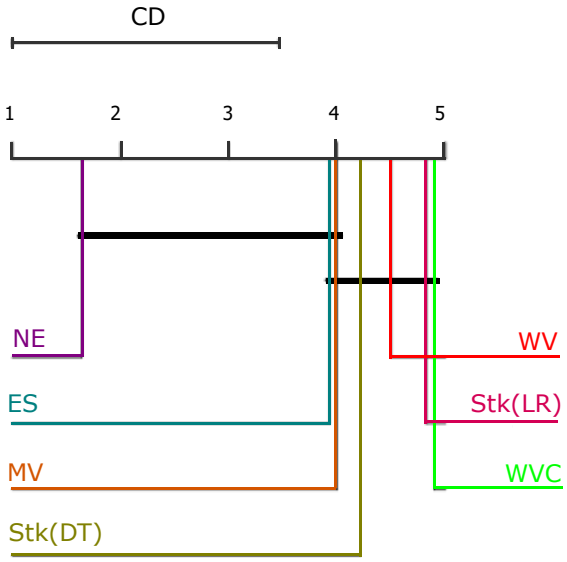



**Figure 4.11:** Critical Difference Diagram of the 7 Ensemble Methods

on the mean accuracies of the baseline methods. Table 4.1 shows the significant wins and losses of each baseline with respect to other baseline methods. The results suggest that Neuro-Ensemble and Evaluation and Selection are particularly superior over all other baseline methods since their total significant wins are greater than their significant losses. The second observation that can be made is that Neuro-Ensemble significantly wins over Evaluation and Selection method (on 50% of the datasets we used) and significantly loses only in one dataset. This suggests that in 13 out of 14 datasets we used, Neuro-Ensemble either performs significantly better than Evaluation and Selection or it performs at the same level.

In addition to the paired t-test study, we performed a post-hoc Nemenyi test relative to Neuro-Ensemble meta-learner and other state-of-the-art ensembling methodologies. The critical diagram of the Nemenyi test [85], at significance level $\alpha = 0.05$, performed on the 14 datasets is shown in Figure 4.11. The ranks in the diagram represent the average ranks of the base learners. In case some baseline classifiers cross the same bold line, it indicates that their performance is not significantly different. This means that their average ranks differ by less than the critical difference (CD) value. Therefore; the null hypothesis stating that the classifiers have the same performance levels is rejected.

The results shown in Figure 4.11 shows that the Neuro-Ensemble meta-learner achieved the best average rank in comparison with other base learners, which is compliant with our previous findings using the t-test. The proposed meta-learner achieves the same significance level of the Selective Evaluation and Majority voting approaches. However, our method achieves a better rank than all other baselines. Additionally, Neuro-Ensemble significantly outperforms other methodologies, namely stacking with decision tree, stacking with logistic regression, weighted voting and weighted voting per class. The reason behind Neuro-Ensemble superiority over the other baseline methods is that it weights the votes of the base learners based on their learned expertise on a class-level basis.

# CHAPTER 5

# Multivariate Time Series Application: SEP prediction

Solar energetic particles are a result of intense solar events such as solar flares and Coronal Mass Ejections (CMEs). These latter events all together can cause major disruptions to spacecraft that are in Earth's orbit and outside of the magnetosphere. In this work we are interested in establishing the necessary conditions for a major geo-effective solar particle storm immediately after a major flare, namely the existence of a direct magnetic connection. To our knowledge, this is the first work that explores not only the correlations of GOES X-ray and proton channels, but also the correlations that happen across all the proton channels. We found that proton channels auto-correlations and cross-correlations may also be precursors to the occurrence of an SEP event. In this paper, we tackle the problem of predicting >100 MeV SEP events from a multivariate time series perspective using easily interpretable decision tree models [27].

## 5.1 SEP Vector AutoRegresison Decision Tree

This section introduces a novel approach in predicting the occurrence of $>$ 100 MeV SEP events based on interpretable decision tree models. We considered the X-ray and proton channels as multivariate time series that entail some correlations which may be precursors to the occurrence of an event. While [32] considers the correlation between the X-ray and proton channels only, we extended the correlation study into all the channels, including correlations that happen across different proton channels. We approached

the problem from a multivariate time series classification perspective. The classification task being whether the observed time series windows will lead to an SEP event or not. There are two ways of performing a time series classification. The first approach, which first appeared in [86], is to use the raw time series data and find the K-nearest-neighbor with a similarity measures such as Euclidean distance, and dynamic time warping. This approach is effective when the time series of the same label shows a distinguishable similar shape pattern. In this problem, the time series that we are working with are direct instruments readings that show a jitter effect, which is common in electromechanical device readings [87]. An example of the jitter effect is shown in P10, and P11 in Figure. 2.2-b and Figure. 2.2-c. Time series jitter makes it hard for distance measures, including elastic measures, to capture similar shape patterns. Therefore, we explored the second time series classification approach that relies on extracting features from the raw time series before feeding it to a model. In the next subsections, we will talk about the time series data extraction, the feature generation and data pre-processing.

### *5.1.1 Data Extraction*

Our approach starts from the assumption that a >100 MeV impulsive event may occur if the parent X-ray event peak is at least M3.5 as was suggested in [32]. Therefore we carefully picked the negative class an X-ray event whose peak intensity is at least M3.5 but did not lead to any SEP event (refer column 3 in Table 2.4). We extracted different observation windows of data that we call a span. A span is defined as the number of hours that constitute the observation period prior to an X-ray event. A total of 94 (47*2) X-ray events (shown in column 3 and column 2 of Table .2.3 and Table .2.4 respectively) were extracted with different span windows. The span concept is illustrated in the yellow shaded area in Figure. 2.2. The span window, in this case is 10 hours and stops exactly at the start time of the X-ray event. As we considered the five minutes as the cadence

between reports, a 10-hour span window represents a 120-length multivariate X-ray and proton time series.

### 5.1.2 *Feature Generation*

To express the X-ray and proton cross-channel correlations we used a Vector Autoregression Model (VAR) which is a stochastic process model used to capture the linear interdependencies among multiple time series. VAR is the extension of the univariate autoregressive model to multivariate time series. The VAR model is useful for describing the behavior of economic, financial time series and for forecasting [88]. The VAR model permits us to express each time series window as a linear function of past lags (values in the past) of itself and of past lags of the other time series. The lag $l$ signifies the factor by which we multiply a value of a time series to produce its previous value in time. Theoretically, if there exists a magnetic connection between the Sun and Earth through the Parker spiral, the X-ray fluctuation precedes its corresponding proton fluctuation. Therefore, we do not express the X-ray channels in terms of the other time series, but, we focus on expressing the proton channels with respect to the past lags of themselves and with past lags of the X-ray channels (xs and xl). The VAR model of order one, denoted as VAR(1) in our setting can be expressed by Equations (5.1)-(5.4).

There is a total of eight time series that represent the proton channels. Every equation highlights the relationship between the dependent variable and the other protons and X-ray variables, which are independent variables. The higher the dependence of a proton channel on an independent variable, the higher is the magnitude of the coefficient $\|\phi_{dependent\_independent}\|$. We used the coefficients of the proton equations as a feature vector representing a data sample. The feature vector representing a data point using the VAR(n) model is expressed in Equation.5.1.2.

$$P6_{t,1} = \phi_{P6\_xs,1} * P6_{t-1,1} + \phi_{P6\_xl,1} * P6_{t-1,1} + \ldots + \phi_{P6\_P11,1} * P6_{t-1,1} + \alpha_{P6_{t,1}} \quad (5.1)$$

$$P7_{t,1} = \phi_{P7\_xs,1} * P7_{t-1,1} + \phi_{P7\_xl,1} * P7_{t-1,1} + \ldots + \phi_{P7\_P11,1} * P7_{t-1,1} + \alpha_{P7_{t,1}} \quad (5.2)$$

$$P8_{t,1} = \phi_{P8\_xs,1} * P8_{t-1,1} + \phi_{P8\_xl,1} * P8_{t-1,1} + \ldots + \phi_{P8\_P11,1} * P8_{t-1,1} + \alpha_{P8_{t,1}} \quad (5.3)$$

$$\vdots$$

$$P11_{t,1} = \phi_{P11\_xs,1} * P11_{t-1,1} + \phi_{P11\_xl,1} * P11_{t-1,1} + \ldots + \phi_{P11\_P11,1} * P11_{t-1,1} + \alpha_{P11_{t,1}} \quad (5.4)$$

Since the lag parameter $l$ determines the number of coefficients involved in the equation, the number of features in the feature vector varies. More specifically, the total number of features are 8 (independent variables) * 6 (dependent variables).

$$x = \begin{bmatrix} \phi_{P6\_xs,1} \\ \phi_{P6\_xl,1} \\ \phi_{P6\_P6,1} \\ \phi_{P6\_P7,1} \\ \vdots \\ \phi_{P11\_P8,n} \\ \phi_{P11\_P9,n} \\ \phi_{P11\_P10,n} \\ \phi_{P11\_P11,n} \end{bmatrix}$$

### 5.1.3 *Data Preprocessing*

Before feeding the data to a classifier we cleaned the data from empty values that appear in the generated features. To do so, we used the 3-nearest neighbors class-level imputation technique. The method finds the 3 nearest neighbors that have the same label of the sample with the missing feature. Nearest neighbors imputation weights the samples using the mean squared difference on features based on the other non-missing features. Then it imputes the missing value with the nearest neighbor sample. The reason why the imputation is done on a class level basis is that features may behave differently across the two classes (SEP and non-SEP), therefore; it is important to impute the missing data with the same class values.

## 5.2 Experimental Evaluation

In this section we explain the decision tree model that we will be using as well as the sampling methodology. We will also provide a rationale for the choice of parameters ($l$ and $span$). Finally we will zoom in the best model with the most promising performance levels.

### 5.2.1 *Decision Tree Model*

A decision tree is a hierarchical tree structure used to determine classes based on a series of rules/questions about the attribute values of the data points [89]. Every non-leaf node represents an attribute split (question) and all the leaf nodes represent the classification result. In short, given a set of features with their corresponding classes a decision tree produces a sequence of questions that can be used to recognize the class of a data sample.

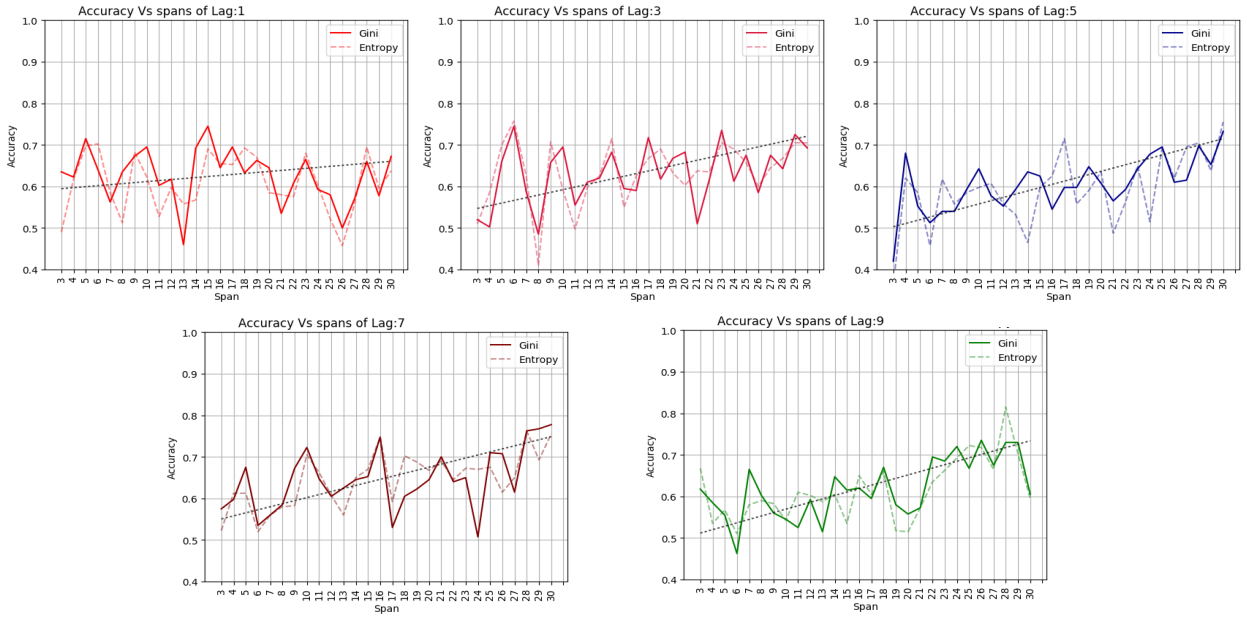



**Figure 5.1:** Decision tree accuracy with respect to the span window and the lag parameters using Gini and information gain splitting criteria. The dotted line shows a linear fit to the accuracy curve.

In this paper, the data attributes are the VAR(l) coefficients $[\phi_{p6\_xs,1}, \phi_{p6\_xl,1}, ..., \phi_{p6\_xs,l}]$ and the classes are binary: SEP and non-SEP.

The decision tree classification model first starts by finding the variable that maximizes the separation between classes. Different algorithms use different metrics, also called purity measures, for measuring the feature that maximizes the split. Some splitting criteria include Gini impurity, information gain, and variance reduction. The commonality between these metrics is that they all measure the homogeneity of a given feature with respect to the classes. The metric is applied to each candidate feature to measure the quality of the split, and the best feature is used. In this paper we used the CART decision tree algorithm, as appeared in [90] and [91], with Gini and information gain as the splitting criteria.

### 5.2.2 *Parameter Choice*

Our approach relies heavily on the choice of parameters, namely, the span window and the VAR model lag parameter. The span is the number of observation hours that precede the occurrence of an X-ray event. The latter determines the length of the multivariate time series to be extracted. On the other hand, the lag (l) determines the size of the feature space that will be used as well as the length of the dependence of the time series with each other in the past. As mentioned previously, with a one-step increment of the lag parameter the size of the feature space almost doubles $features\_number$ = 8*(independent variables)*6 (equations)*l+6*(equations). In order to determine the optimal parameters to be used, we run a decision tree model on a set of values for both the span and lag parameters. More specifically, we used the range [3-30] for the span window and the set {1,3,5,7,9} for l. Since we have a balanced dataset we used a stratified Ten-fold cross validation as the sampling methodology. A stratified sampling always ensures a balance in the number of positive and negative samples for both the training and testing data samples. Ten-fold cross-validation randomly splits the data into 10 subsets, models are trained with nine of the folds (90% of the dataset), and test it with one fold (10% of the dataset). Every fold is used once for testing and nine times for training. In our experiments, we report the average accuracy on the 10 folds. Figure 5.1 illustrates the accuracy curves with respect to the span windows for the five lags that we considered. We reported the accuracies of the decision tree model using both gini and information gain splitting criteria. In order to better capture the model behavior with the increasing span we plotted a linear fit to the accuracy curves of each lag. The first observation that can be made is that the slopes of the linear fit for l=1 and l=3 are relatively small in comparison to the other lags (l >3). This signifies that the model does not show any increasing or decreasing accuracy trend with the increase of the span window. Therefore we conclude that l=1 and l=3 are too small to discover any relationship between the

proton and X-ray channels. Having the lag parameter set to l=1 and l=3 corresponds to expressing the time series (independent variable) going back in time up to five minute and 15 minutes respectively. These latter times are small, especially for l=1 (5 minutes), which theoretically is not possible since the protons can at most reach the speed of light that corresponds to a lag of at least 8.5 minutes. For the other lags ($l > 3$) there is noticeable increase in steepness in the accuracy linear fit which suggests that the accuracy increases with the increasing span window. The second observation is that for all the $l > 3$ datasets the best accuracy was achieved in the last four span window (i.e span $\in$ {27,28,29,30}). Therefore, we filtered the initial range of parameter values to {5,7,9} for l and {27,28,29,30} for the span. In the next subsection we will zoom in into every classifier within the parameter grid.

### 5.2.3 *Learning Curves*

To be able to discriminate decision tree models that show similar accuracies we use the model learning curves, also called experience curves, to have an insight in how the accuracy changes as we feed the model with more training examples. Learning curves are particularly useful for comparing different algorithms [92] and choosing optimal model parameters during the design [93]. It is also a good tool for visually inspecting the sanity of the model in case of overtraining or undertraining. Figures 5.2 and 5.3 show the learning curves of the decision tree model using gini and information gain as the splitting criteria respectively. The red line represents the training accuracy which evaluates the model on the newly trained data. The green line shows the testing accuracy which evaluates the model on the the never-seen testing data. The shaded area represents the standard deviation of the accuracies after running the model multiple times with the same number of training data. It is noticeable that the standard deviation becomes higher as the lag is increased. Also, it can be seen that the best average accuracies, that

appeared in Figure 5.1, are not always the ones that have the best learning curves. For example from Figure 5.1, the best accuracy that has been reached appears to be in l=7 and $span = 27, 29$; however, the learning curves corresponding to that span and lag show that the standard deviation is not very smooth as compared to l=5. Therefore the experiments show that using l=5 results in relatively stable models with lower variance. Therefore, we will zoom in l=5 for all the spans $\in \{27, 28, 29, 30\}$ that we previously filtered.

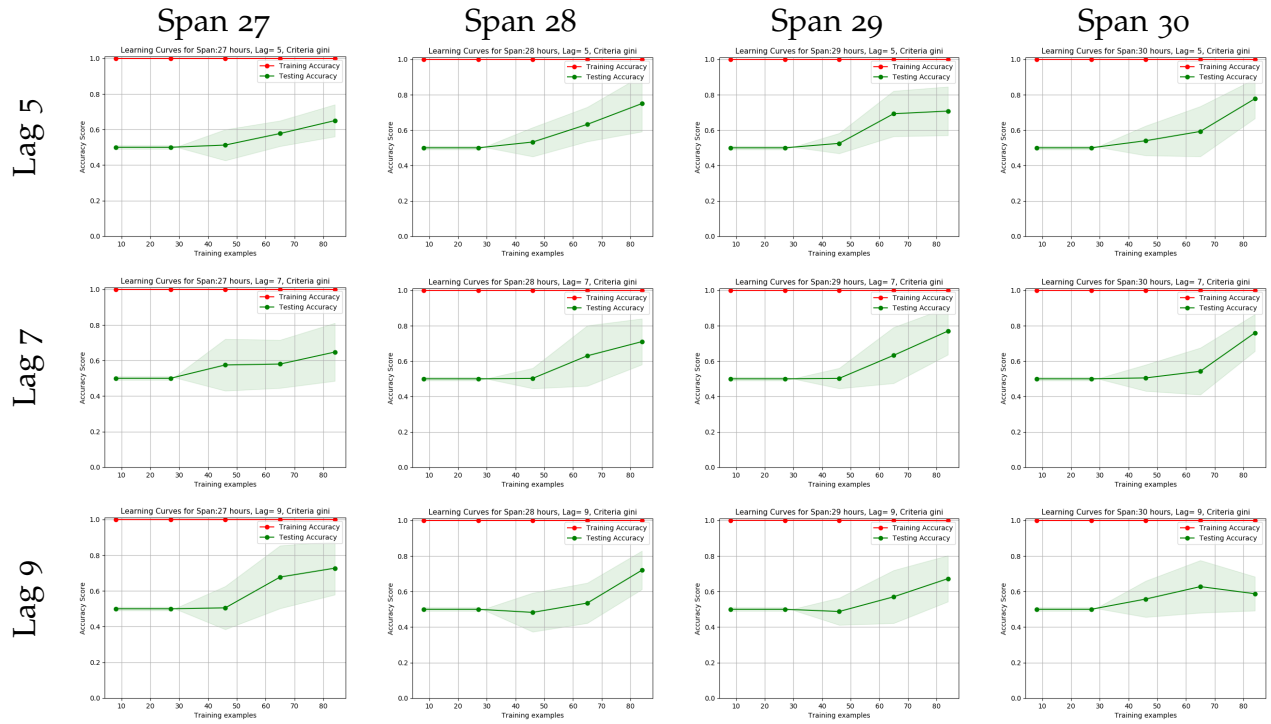



**Figure 5.2:** Learning curve of CART Decision Tree Models with Gini splitting criterion,spans $\in$ {27,28,29,30} and lag $\in$ {5,7,9}

To determine the best behaving model we choose six evaluation metrics that will assess the models' performance from different aspects. Accuracy is the most standard evaluation measure used to assess the quality of a classifier by counting the ratio of correct classification over all the classifications. In this context the accuracy measure is particularly useful because our training and testing data is balanced. The data balance ensures that if the classifier is highly biased toward a given class it will be reflected on the accuracy measure. Recall is the second evaluation measure we considered, also known as

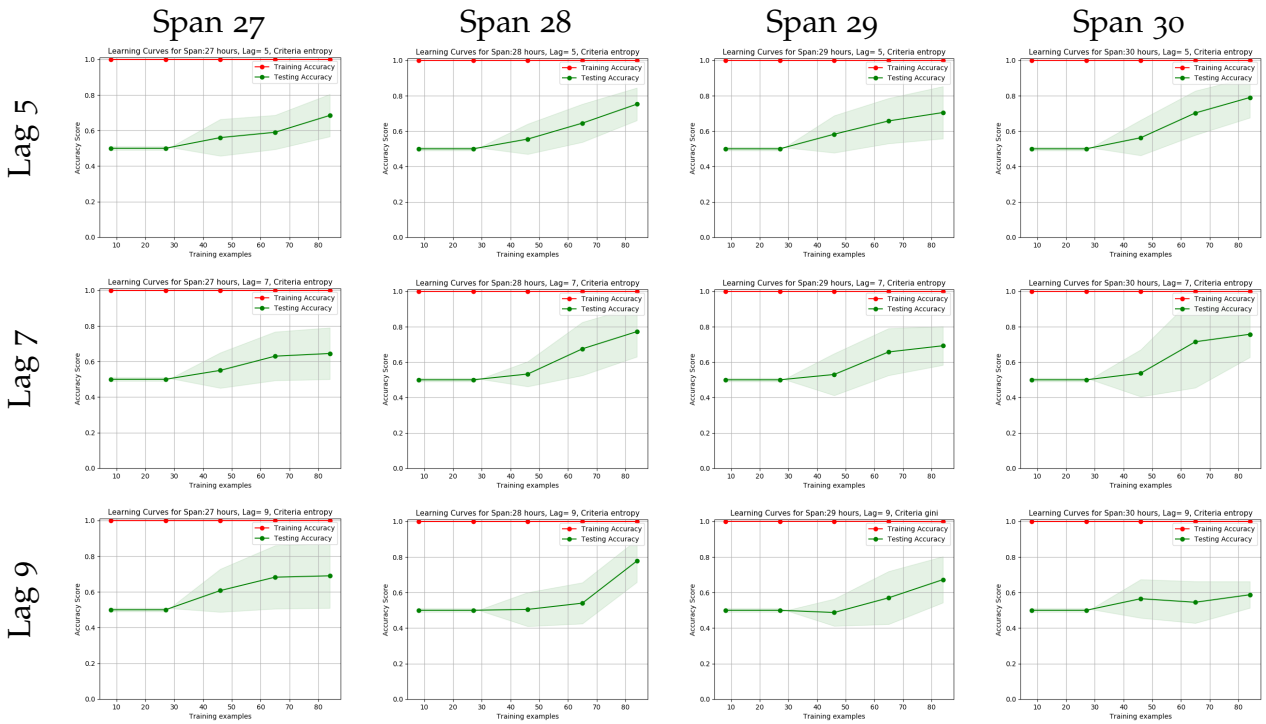



**Figure 5.3**: Learning curve of CART Decision Tree Models with information gain splitting criterion,spans $\in$ {27,28,29,30} and lag $\in$ {5,7,9}

the probability of detection, which characterizes the ability of the classifier to find all of the positive cases. Precision is used to evaluate the model with respect to the false alarms. In fact, precision is 1 - false alarm ratio. Precision and recall are usually anti-correlated; therefore, a useful quantity to compute is their harmonic mean, the F1 score. The last evaluation measure that we consider in the Area Under Curve (AUC) of the Receiver Operating Characteristic curve (ROC) curve. The intuition behind this measure is that AUC equals the probability that a randomly chosen positive example ranks above (is deemed to have a higher probability of being positive than) a randomly chosen negative example. It has been claimed in [94] that the AUC is statistically consistent and more discriminating than accuracy.

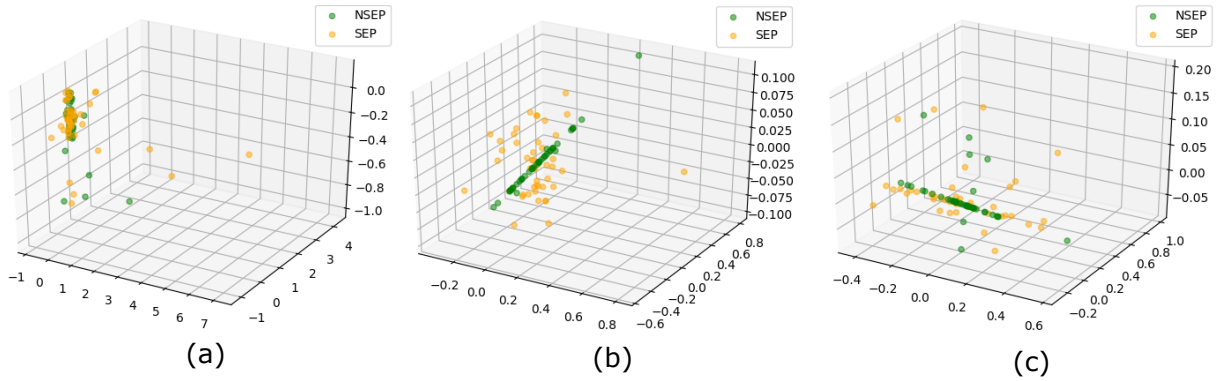



**Figure 5.4:** First 3 PCA components derived from (a) all the original 254 features, (b) the data sub-space containing only 4 parameters selected as the most relevant by the Gini index, and (c) another data sub-space containing 4 different parameters (with 1 repetition) selected as the most relevant by the Entropy measure. The PCA-based visualizations represent (sub-)spaces of the same data set, with lag=5, and span=30.

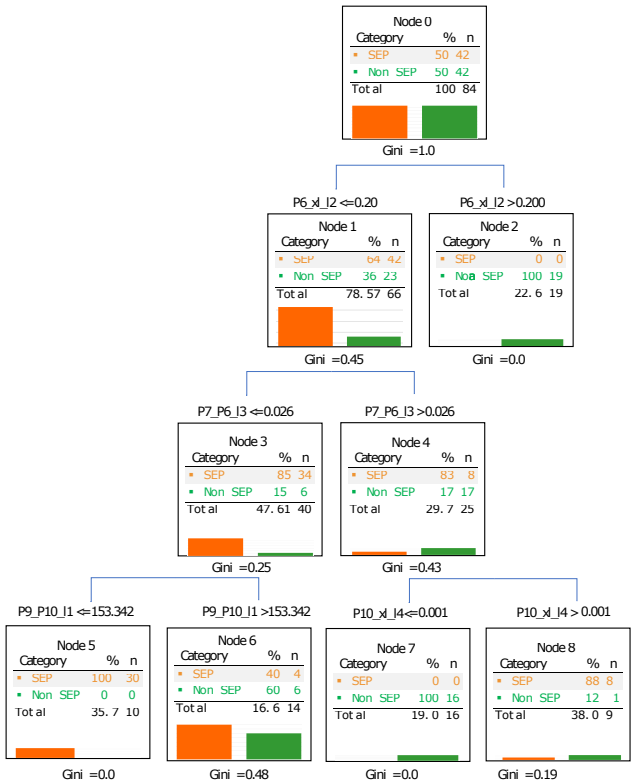



**Figure 5.5:** Decision Tree with Gini splitting criteria (span=30, l=5)

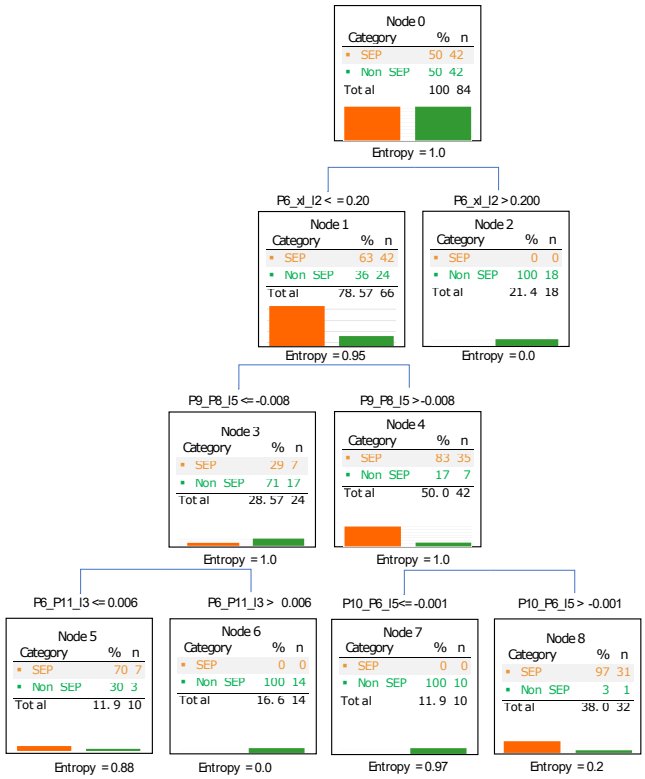



**Figure 5.6:** Decision Tree with information gain splitting criteria (span=30, l=5)

**Table 5.1:** Decision Tree model evaluation for gini and information gain splitting criteria

| | Gini | | | | Information Gain | | | |
|---|---|---|---|---|---|---|---|---|
| Span | 27 | 28 | 29 | 30 | 27 | 28 | 29 | 30 |
| Accuracy | 0.64 | 0.74 | 0.73 | **0.74** | 0.77 | 0.70 | 0.67 | **0.78** |
| Recall | 0.69 | 0.70 | 0.74 | **0.70** | 0.76 | 0.70 | 0.70 | **0.73** |
| Precision | 0.62 | 0.75 | 0.75 | **0.76** | 0.78 | 0.72 | 0.72 | **0.86** |
| F1 | 0.65 | 0.75 | 0.75 | **0.74** | 0.79 | 0.71 | 0.69 | **0.82** |
| AUC | 0.65 | 0.72 | 0.74 | **0.73** | 0.76 | 0.70 | 0.69 | **0.77** |

Table 5.1 shows the aforementioned evaluations on the l=5 datasets. It is noticeable that span=30 achieves the best performance levels for both splitting criteria. The decision

tree models corresponding to those settings using gini and information gain are shown in Figure 5.5 and Figure 5.6 respectively. Although, we are well aware of a number of common dimensionality reduction methods [95–97], we used PCA as the most common one as it served only for purpose of our data visualization. Since the major focus of this chapter is to show that the data transformation using VAR model serves the purpose of separating the feature space, the classical dimensionality reduction methods were not investigated any further. We used PCA dimensionality reduction technique to plot the full feature space with the 254 dimensions of the lag 5 and span 30 in Figure 5.4-a, as well as the reduced feature space with only the selected features from the gini measure in Figure 5.4-b and entropy measure in Figure 5.4-c [98]. It is clearly visible that the SEP and non-SEP classes are almost indistinguishable when all the dimensions are used. When the decision tree feature selection is applied, the data points become more scattered in space and therefore easier for the classifier to distinguish. We also note that both decision tree classifiers have as a root a proton x-ray correlation parameter (P6_xl_l2). Some of the intermediate and leaf nodes have features that show correlations between proton channels is their conditions. This suggests that cross-correlations in proton channels are equally important to X-ray and proton channels correlations that appeared in [32]. Our best model shows a descent accuracy that is comparable (3% better) to the UMASEP system that uses the same catalog. We also made sure that our model is not biased towards the missing data of the lower energy channels P6 and P7 of GOES-12 by choosing the same number of samples of positive and negative class that happened during the GOES-12 coverage period.

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

## 6.1 Conclusion

This thesis presents our work on one of the most prominent tasks of time series mining which is supervised learning on time series data. Our contributions include a new heterogeneous ensemble based on a modified neural network meta-learner, *Neuro-Ensemble* [25, 26], that improves the predictive performance of the participating base classifiers based on ensemble stacking/meta-learning paradigm. Our second contribution tackles the problem of lack of interpretability of black-box models, namely neural network, by proposing a new one-dimensional neural network model that produces visual latent features. The latent features are various lengths one-dimensional subsequences that have high discriminatory power, also known as shapelet. Although our initial 1DCNN model outperformed all other baselines, it suffered from a high number of shapelets and therefore required an important number of Floating Number of Operations per second (FLoPS) to train the model. We then proposed network pruning as a strategy to solely mine the most prominent various lengths shapelets. The new extension model allowed the definition of the number of shapelets to be mined (model sparsity level) in a data-driven way following the learning curves of the model. Finally, we show time series and sequence mining applications through real-life astrophysics problems of solar energetic particle events prediction. We demonstrate the use of Vector Auto-Regression (VAR) model for high dimensional time series data representation prior to the prediction task [27].

## 6.2 Future Work

We plan to extend our research on interpretable feature learning of time series data. In the following list, we specify these ideas.

1. One of the main assumptions about shapelet based classifiers is that the pattern length is a user-defined hyper-parameter usually selected at cross-validation time. In our work, we showed that the same dataset can have discriminative patterns of different lengths. One desired future work direction is to remove the parameter search step and automatically learn the optimal pattern lengths in a data-driven fashion.

2. We aim to work on extending 1DCNN to the multivariate case and conduct a detailed case study with application in solar flare forecasting on the SWAN-SF dataset [99]. We also want to use our model on other types of data (such as spatial data, graphs) and using a deeper CNN topology.

3. We would like to investigate the use of additional second-order derivatives network pruning methods such as Optimal Brain Surgeon [100] and Optimal Brain Damage [101], to be able to compare the network performance and interpretability quality based on the automatically mined minimal set of necessary interpretable shapelets.

4. We aim to work on improving the mined shapelets' visual quality using warm initialization of the kernels rather than Glorot Xavier initializer [70].

   Although we are at the end of this thesis, the possibilities for future data exploratory endeavors are endless.

# CHAPTER 7

# Appendix

**Table 7.1**: UCR Datasets Metadata and optimal K% sparsity

| ID | Dataset Name | $\mathcal{L}$ | $\mathcal{K}$% | DS Size | TS Length |
|---|---|---|---|---|---|
| 1 | Adiac | 37 | 35 | 781 | 176 |
| 2 | ArrowHead | 3 | 40 | 211 | 251 |
| 3 | Beef | 5 | 75 | 60 | 470 |
| 4 | BeetleFly | 2 | 80 | 40 | 512 |
| 5 | BirdChicken | 2 | 30 | 40 | 512 |
| 6 | Car | 4 | 75 | 120 | 577 |
| 7 | ChlorineConcentration | 3 | 70 | 4307 | 166 |
| 8 | Coffee | 2 | 95 | 56 | 286 |
| 9 | Computers | 2 | 55 | 500 | 720 |
| 10 | Cricket_X | 12 | 60 | 780 | 300 |
| 11 | Cricket_Y | 12 | 55 | 780 | 300 |
| 12 | Cricket_Z | 12 | 40 | 780 | 300 |
| 13 | DiatomSizeReduction | 4 | 30 | 322 | 345 |
| 14 | DPOutlineAgeGroup | 3 | 95 | 539 | 80 |
| 15 | DPOutlineCorrect | 2 | 50 | 876 | 80 |
| 16 | ECG200 | 2 | 70 | 200 | 96 |
| 17 | ECGFiveDays | 2 | 80 | 884 | 136 |
| 18 | FaceAll | 14 | 85 | 2250 | 131 |
| 19 | FaceFour | 4 | 75 | 112 | 350 |
| 20 | FacesUCR | 14 | 65 | 2250 | 131 |
| 21 | 50words | 50 | 65 | 905 | 270 |
| 22 | FISH | 7 | 75 | 350 | 463 |
| 23 | Gun_Point | 2 | 60 | 200 | 150 |
| 24 | Haptics | 5 | 25 | 463 | 1092 |
| 25 | Herring | 2 | 60 | 128 | 512 |
| 26 | InlineSkate | 7 | 90 | 650 | 1882 |
| 27 | InsectWingbeatSound | 11 | 65 | 2200 | 256 |
| 28 | Lighting2 | 2 | 65 | 121 | 637 |
| 29 | Lighting7 | 7 | 45 | 143 | 319 |
| 30 | MedicalImages | 10 | 80 | 1141 | 99 |
| 31 | MPOutlineAgeGroup | 3 | 90 | 554 | 80 |
| 32 | MoteStrain | 2 | 40 | 1272 | 84 |
| 33 | NIFatalECG_Thorax1 | 42 | 65 | 3765 | 750 |
| 34 | NIFatalECG_Thorax2 | 42 | 85 | 3765 | 750 |
| 35 | OliveOil | 4 | 45 | 60 | 570 |
| 36 | OSULeaf | 6 | 85 | 442 | 427 |
| 37 | Plane | 7 | 80 | 210 | 144 |
| 38 | PPOutlineCorrect | 2 | 85 | 891 | 80 |
| 39 | PPTW | 6 | 85 | 605 | 80 |
| 40 | ScreenType | 3 | 90 | 750 | 720 |
| 41 | ShapesAll | 60 | 50 | 1200 | 512 |
| 42 | SonyAIBORSurface | 2 | 75 | 621 | 70 |
| 43 | SonyAIBORSurfaceII | 2 | 55 | 980 | 65 |
| 44 | SwedishLeaf | 15 | 80 | 1125 | 128 |
| 45 | Symbols | 6 | 85 | 1020 | 398 |
| 46 | synthetic_control | 6 | 75 | 600 | 60 |
| 47 | ToeSegmentation1 | 2 | 45 | 268 | 277 |
| 48 | ToeSegmentation2 | 2 | 40 | 166 | 343 |
| 49 | Trace | 4 | 95 | 200 | 275 |
| 50 | TwoLeadECG | 2 | 95 | 1162 | 82 |
| 51 | Two_Patterns | 4 | 70 | 5000 | 128 |
| 52 | WGLibraryAll | 8 | 70 | 4478 | 945 |
| 53 | WGLibrary_X | 8 | 65 | 4478 | 315 |
| 54 | WGLibrary_Y | 8 | 40 | 4478 | 315 |
| 55 | WGLibrary_Z | 8 | 80 | 4478 | 315 |
| 56 | wafer | 2 | 95 | 7164 | 152 |
| 57 | WordsSynonyms | 25 | 50 | 905 | 270 |
| 58 | Worms | 5 | 35 | 258 | 900 |
| 59 | yoga | 2 | 70 | 3300 | 426 |

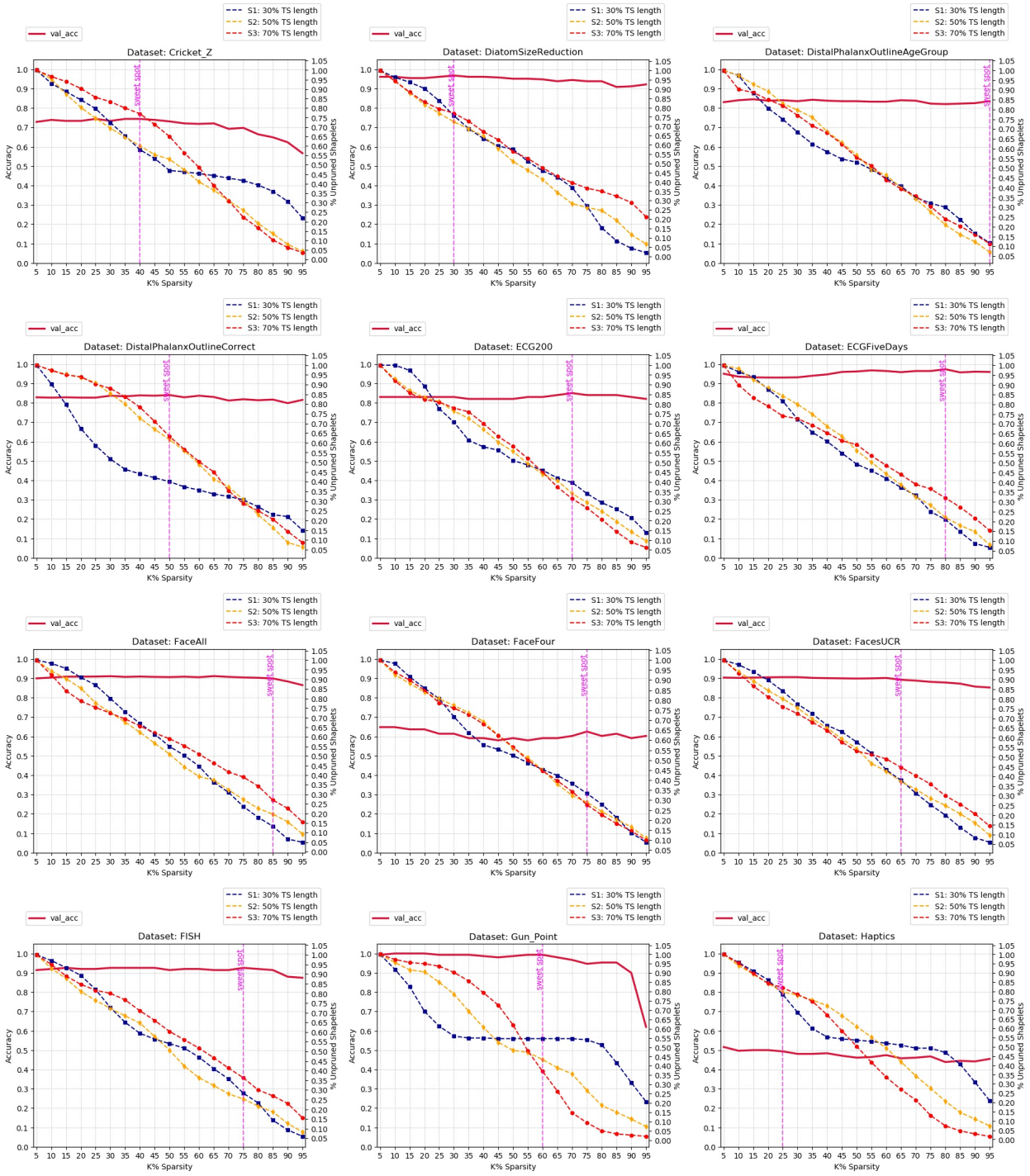



**Figure 7.1:** Validation Accuracy with respect to K% Sparsity and sparsity curves for the three convolutional blocks corresponding to the three shapelets lengths for : Cricket_Z, DSReduction, DPOAgeGroup, DPCorrect, ECG200, ECGFiveDays, FaceAll, FaceFour, FacesUCR, FISH, Gun_Point, Haptics.

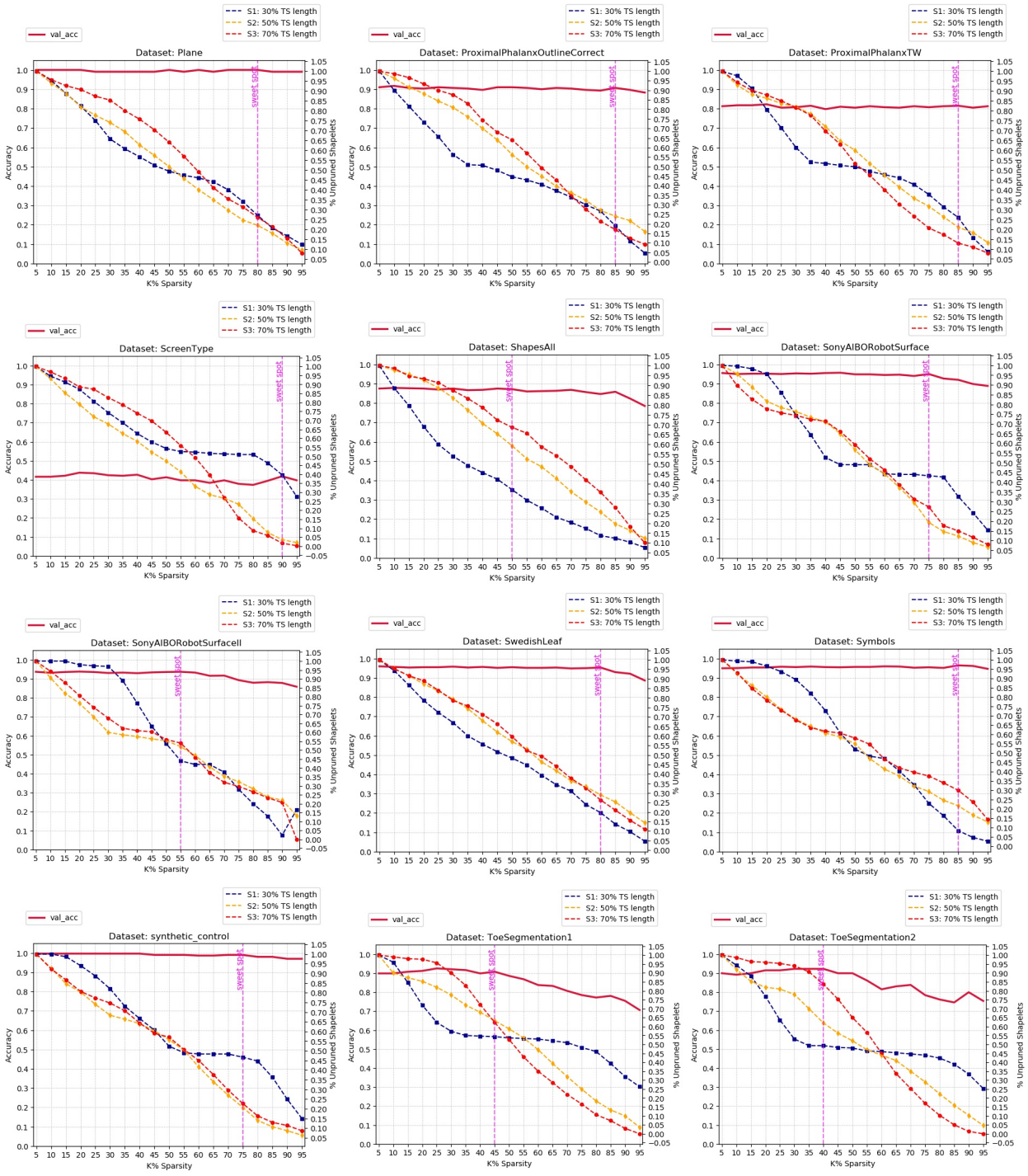



**Figure 7.2:** Validation Accuracy with respect to K% Sparsity and sparsity curves for the three convolutional blocks corresponding to the three shapelets lengths for : Plane, ProximalPhalanxOutlineCorrect, ProximalPhalanxTW, ScreenType, ShapesAll, SonyAIBORobotSurface, SonyAIBORobotSurfaceII, SwedishLeaf, Symbols, synthetic_control, ToeSegmentation1, ToeSegmentation2.

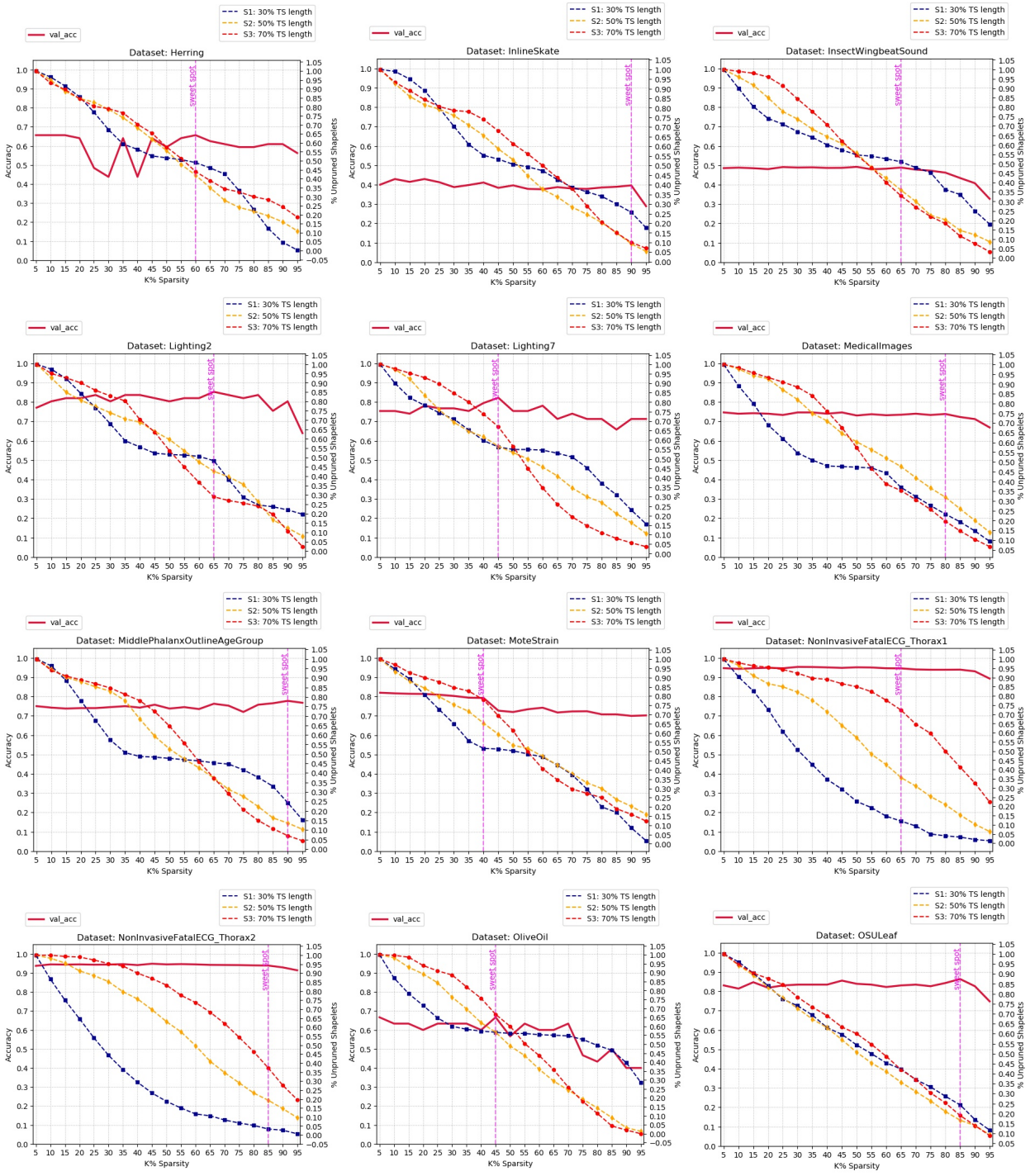



**Figure 7.3**: Validation Accuracy with respect to K% Sparsity and sparsity curves for the three convolutional blocks corresponding to the three shapelets lengths for : Herring, InlineSkate, InsectWingbeatSound, Lighting2, Lighting7, MedicalImages, MiddlePhalanxOutlineAgeGroup, MoteStrain, NonInvasiveFatalECG_Thorax1, NonInvasiveFatalECG_Thorax2, OliveOil, OSULeaf.

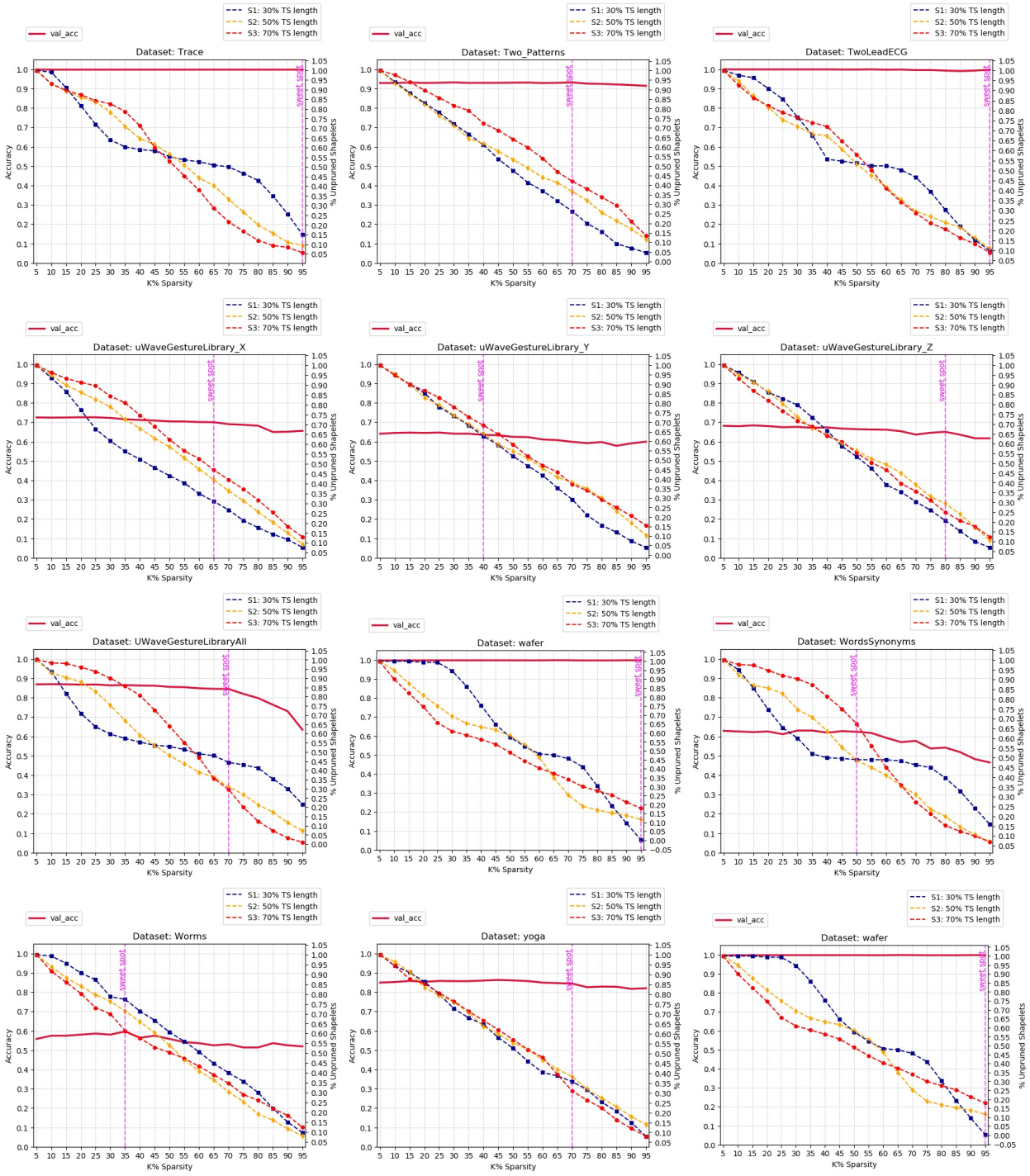



**Figure 7.4**: Validation Accuracy with respect to K% Sparsity and sparsity curves for the three convolutional blocks corresponding to the three shapelets lengths for : Trace, Two_Patterns, TwoLeadECG, WGLibrary_X, WGLibrary_Y, WGLibrary_Z, WGLibraryAll, wafer, WordsSynonyms, Worms, yoga, and wafer.

# Bibliography


[1] D Gunning. Explainable artificial intelligence (xai) darpa-baa-16-53. *Defense Advanced Research Projects Agency*, 2016.

[2] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[3] Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. Why a right to explanation of automated decision-making does not exist in the general data protection regulation. *International Data Privacy Law*, 7(2):76–99, 2017.

[4] Ruizhe Ma and Rafal Angryk. Distance and density clustering for time series data. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pages 25–32. IEEE, 2017.

[5] Vit Niennattrakul and Chotirat Ann Ratanamahatana. On clustering multimedia time series data using k-means and dynamic time warping. In *null*, pages 733–738. IEEE, 2007.

[6] Carlo Drago and Germana Scepi. Time series clustering from high dimensional data. In *International Workshop on Clustering High-Dimensional Data*, pages 72–86. Springer, 2012.

[7] Ruizhe Ma, Azim Ahmadzadeh, Soukaina Filali Boubrahimi, Manolis K Georgoulis, and Rafal A Angryk. Solar pre-flare classification with time series profiling. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4967–4976. IEEE, 2019.

[8] Petrus C Martens, S Filali Boubrahimi, Berkay Aydin, and Rafal Angryk. Forecasting space weather hazards for astronauts in deep space. *AGUFM*, 2019:SH34B–08W, 2019.

[9] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[10] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[11] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297, 2012.

[12] Shah Muhammad Hamdi, Berkay Aydin, Soukaina Filali Boubrahimi, Rafal Angryk, Lisa Crystal Krishnamurthy, and Robin Morris. Biomarker detection from fmri-based complete functional connectivity networks. In *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 17–24. IEEE, 2018.

[13] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *KDD*, volume 98, pages 16–22, 1998.

[14] Mohammad Shokoohi-Yekta, Yanping Chen, Bilson Campana, Bing Hu, Jesin Zakaria, and Eamonn Keogh. Discovery of meaningful rules in time series. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1085–1094, 2015.

[15] Jyoti Bansal. Time series anomaly detection using multiple statistical models, December 18 2007. US Patent 7,310,590.

[16] Li Wei, Nitin Kumar, Venkata Nishanth Lolla, Eamonn J Keogh, Stefano Lonardi, and Chotirat (Ann) Ratanamahatana. Assumption-free anomaly detection in time series. In *SSDBM*, volume 5, pages 237–242, 2005.

[17] Dominique T Shipmon, Jason M Gurevitch, Paolo M Piselli, and Stephen T Edwards. Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data. *arXiv preprint arXiv:1708.03665*, 2017.

[18] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *Acm Sigmod Record*, 23(2):419–429, 1994.

[19] Dimitrios Gunopulos, Gautam Das, and Gautam Das. Time series similarity measures and time series indexing. *Acm Sigmod Record*, 30(2):624, 2001.

[20] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 151–162, 2001.

[21] Eamonn J Keogh and Michael J Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Kdd*, volume 98, pages 239–243, 1998.

[22] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[23] David H Wolpert, William G Macready, et al. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[24] Gelu Nita, Manolis Georgoulis, Irina Kitiashvili, Viacheslav Sadykov, Enrico Camporeale, Alexander Kosovichev, Haimin Wang, Vincent Oria, Jason Wang, Rafal Angryk, et al. Machine learning in heliophysics and space weather forecasting: A white paper of findings and recommendations. *arXiv preprint arXiv:2006.12224*, 2020.

[25] Soukaina Filali Boubrahimi, Ruizhe Ma, Berkay Aydin, and Rafal Angryk. Neuro-ensemble. In *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 54–61. IEEE, 2018.

[26] Soukaïna Filali Boubrahimi and Rafal Angryk. Heuristics significance of neuro-ensemble-based time series classification. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 6–15. IEEE, 2018.

[27] Soukaina Filali Boubrahimi, Berkay Aydin, Petrus Martens, and Rafal Angryk. On the prediction of >100 MeV solar energetic particle events using GOES satellite data. In *Big Data*. IEEE, 2017.

[28] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[29] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[30] John S Neal and Lawrence W Townsend. Predicting dose-time profiles of solar energetic particle events using bayesian forecasting methods. *IEEE transactions on nuclear science*, 48(6):2004–2009, 2001.

[31] Marlon Núñez. Real-time prediction of the occurrence and intensity of the first hours of >100 MeV solar energetic proton events. *Space Weather*, 13(11):807–819, 2015.

[32] Marlon Nunez. Predicting solar energetic proton events (E >10 MeV). *Space Weather*, 9(7), 2011.

[33] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *23rd ICML*, pages 1033–1040. ACM, 2006.

[34] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *KIS*, 7(3):358–386, 2005.

[35] Soukaina Filali Boubrahimi, Berkay Aydin, Michael A Schuh, Dustin Kempton, Rafal A Angryk, and Ruizhe Ma. Spatiotemporal interpolation methods for solar event trajectories. *APJs*, 236(1):23, 2018.

[36] Soukaina Filali Boubrahimi, Berkay Aydin, Dustin Kempton, and Rafal Angryk. Spatio-temporal interpolation methods for solar events metadata. In *IEEE Big Data 2016*, pages 3149–3157. IEEE, 2016.

[37] Soukaina Filali Boubrahimi, Berkay Aydin, Dustin Kempton, Sushant S Mahajan, and Rafal Angryk. Filling the gaps in solar big data: Interpolation of solar filament event instances. In *BDCloud 2016*, pages 97–104. IEEE, 2016.

[38] Soukaina Filali Boubrahimi, Berkay Aydin, Dustin Kempton, and Rafal A Angryk. Solev: a video generation framework for solar events from mixed data sources (demo paper). In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 90. ACM, 2016.

[39] Soukaïna Filali Boubrahimi, Ruizhe Ma, Berkay Aydin, Shah Muhammad Hamdi, and Rafal Angryk. Scalable knn search approximation for time series data. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 970–975. IEEE, 2018.

[40] Paolo Tormene, Toni Giorgino, Silvana Quaglini, and Mario Stefanelli. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial intelligence in medicine*, 45(1):11–34, 2009.

[41] Eamonn J Keogh and Michael J Pazzani. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–11. SIAM, 2001.

[42] Eamonn J Keogh and Michael J Pazzani. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM international conference on data mining*, pages 1–11. SIAM, 2001.

[43] Young-Seon Jeong, Myong K Jeong, and Olufemi A Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9):2231–2240, 2011.

[44] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

[45] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. *IEEE transactions on Knowledge and Data Engineering*, 25(6):1425–1438, 2013.

[46] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. *IEEE transactions on Knowledge and Data Engineering*, 25(6):1425–1438, 2012.

[47] Pierre-François Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318, 2009.

[48] Lei Chen and Raymond Ng. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 792–803. VLDB Endowment, 2004.

[49] Lexiang Ye and Eamonn Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data mining and knowledge discovery*, 22(1-2):149–182, 2011.

[50] Thanawin Rakthanmanon and Eamonn Keogh. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *proceedings of the 2013 SIAM International Conference on Data Mining*, pages 668–676. SIAM, 2013.

[51] Pavel Senin and Sergey Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *2013 IEEE 13th international conference on data mining*, pages 1175–1180. IEEE, 2013.

[52] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401. ACM, 2014.

[53] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[55] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 342–350. JMLR. org, 2017.

[56] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[57] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[58] Gavin Brown and Ludmila I Kuncheva. "good" and "bad" diversity in majority vote ensembles. In *International Workshop on Multiple Classifier Systems*, pages 124–133. Springer, 2010.

[59] Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine learning*, 54(3):255–273, 2004.

[60] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *ICML*, pages 743–750, 2000.

[61] Hilan Bensusan, Christophe G Giraud-Carrier, and Claire Julia Kennedy. A higher-order approach to meta-learning. *ILP Work-in-progress reports*, 35, 2000.

[62] Alexandros Kalousis and Theoharis Theoharis. Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis*, 3(5):319–337, 1999.

[63] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[64] Kevin Woods, W. Philip Kegelmeyer, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence*, 19(4):405–410, 1997.

[65] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[66] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[67] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

[68] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[69] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *arXiv preprint arXiv:1810.07758*, 2018.

[70] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[71] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[72] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing Ltd, 2017.

[73] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[74] Soukaina Filali Boubrahimi, Ruizhe Ma, and Rafal Angryk. Neuro-ensemble for time series data classification. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 50–59. IEEE, 2018.

[75] Jason Lines and Anthony Bagnall. Ensembles of elastic distance measures for time series classification. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 524–532. SIAM, 2014.

[76] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139, 1999.

[77] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[78] Juan José Rodríguez, Carlos J Alonso, and José A Maestro. Support vector machines of interval-based features for time series classification. *Knowledge-Based Systems*, 18(4-5):171–178, 2005.

[79] Leo Breiman, JH Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. wadsworth, 1984. *Google Scholar*, 1993.

[80] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[81] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *ACM sigmod record*, volume 24, pages 71–79. ACM, 1995.

[82] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[83] Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, 2011.

[84] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of human genetics*, 7(2):179–188, 1936.

[85] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[86] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM, 2006.

[87] Jeffrey D Scargle. Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal*, 263:835–853, 1982.

[88] Eric Zivot and Jiahui Wang. Vector autoregressive models for multivariate time series. *Modeling Financial Time Series with S-Plus®*, pages 385–429, 2006.

[89] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[90] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[91] Dan Steinberg and Phillip Colla. CART: classification and regression trees. *The top ten algorithms in data mining*, 9:179, 2009.

[92] PG Madhavan. A new recurrent neural network learning algorithm for time series prediction. *Journal of Intelligent Systems*, 7(1-2):103–116, 1997.

[93] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[94] Charles X Ling, Jin Huang, and Harry Zhang. AUC: a statistically consistent and more discriminating measure than accuracy. In *IJCAI*, volume 3, pages 519–524, 2003.

[95] Suresh Balakrishnama and Aravind Ganapathiraju. Linear discriminant analysis-a brief tutorial. In *Institute for Signal and information Processing*, volume 18, pages 1–8, 1998.

[96] Brett Williams, Andrys Onsman, and Ted Brown. Exploratory factor analysis: A five-step guide for novices. *Australasian journal of paramedicine*, 8(3), 2010.

[97] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[98] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[99] Rafal A Angryk, Petrus C Martens, Berkay Aydin, Dustin Kempton, Sushant S Mahajan, Sunitha Basodi, Azim Ahmadzadeh, Soukaina Filali Boubrahimi, Shah Muhammad Hamdi, Michael A Schuh, et al. Multivariate time series dataset for space weather data analytics. 2019.

[100] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[101] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.