PhD-FSTC-2018-01
Faculté des Sciences, de la Technologie et de la Communication

DISSERTATION

Defense held on 11/01/2018 in Luxembourg
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG
EN INFORMATIQUE

by

Daoyuan LI
Born on August 5th, 1987 in Hubei, China

TRANSFORMING TIME SERIES FOR EFFICIENT AND ACCURATE CLASSIFICATION

Dissertation defense committee

Dr. Jacques Klein, Chairman
Senior Research Scientist, Université du Luxembourg

Dr. Tegawendé F. Bissyandé, Vice Chairman
Research Scientist, Université du Luxembourg

Dr. Yves Le Traon, Dissertation Supervisor
Professor, Université du Luxembourg

Dr. Jessica Lin
Associate Professor, George Mason University

Dr. Matthieu Geist
Professor, Université de Lorraine

ABSTRACT

Time series data refer to sequences of data that are ordered either temporally, spatially or in another defined order. They can be frequently found in a variety of domains, including financial data analysis, medical and health monitoring and industrial automation applications. Due to their abundance and wide application scenarios, there has been an increasing need for efficient machine learning algorithms to extract information and build knowledge from these data. One of the major tasks in time series mining is time series classification (TSC), which consists of applying a learning algorithm on labeled data to train a model that will then be used to predict the classes of samples from an unlabeled data set. Due to the sequential characteristic of time series data, state-of-the-art classification algorithms (such as SVM and Random Forest) that performs well for generic data are usually not suitable for TSC. In order to improve the performance of TSC tasks, this dissertation proposes different methods to transform time series data for a better feature extraction process as well as novel algorithms to achieve better classification performance in terms of computation efficiency and classification accuracy.

In the first part of this dissertation, we conduct a large scale empirical study that takes advantage of discrete wavelet transform (DWT) for time series dimensionality reduction. We first transform real-valued time series data using different families of DWT. Then we apply dynamic time warping (DTW)-based 1NN classification on 39 datasets and find out that existing DWT-based lossy compression approaches can help to overcome the challenges of storage and computation time. Furthermore, we provide assurances to practitioners by empirically showing, with various datasets and with several DWT approaches, that TSC algorithms yield similar accuracy on both compressed (*i.e.*, approximated) and raw time series data. We also show that, in some datasets, wavelets may actually help in reducing noisy variations which deteriorate the performance of TSC tasks. In a few cases, we note that the residual details/noises from compression are more useful for recognizing data patterns.

In the second part, we propose a language model-based approach for TSC named Domain Series Corpus (DSCo), in order to take advantage of mature techniques from both time series mining and Natural Language Processing (NLP) communities. After transforming real-valued time series into texts using Symbolic Aggregate approXimation (SAX), we build per-class language models (unigrams and bigrams) from these symbolized text corpora. To classify unlabeled samples, we compute the fitness of each symbolized sample against all per-class models and choose the class represented by the model with the best fitness score. Through extensive experiments on an open dataset archive, we demonstrate that DSCo performs similarly to approaches working with original uncompressed numeric data. We further propose DSCo-NG to improve the computation efficiency and classifi-

cation accuracy of DSCo. In contrast to DSCo where we try to find the best way to recursively segment time series, DSCo-NG breaks time series into smaller segments of the same size, this simplification also leads to simplified language model inference in the training phase and slightly higher classification accuracy.

The third part of this dissertation presents a multiscale visibility graph representation for time series as well as feature extraction methods for TSC, so that both global and local features are fully extracted from time series data. Unlike traditional TSC approaches that seek to find global similarities in time series databases (*e.g.*, 1NN-DTW) or methods specializing in locating local patterns/subsequences (*e.g.*, shapelets), we extract solely statistical features from graphs that are generated from time series. Specifically, we augment time series by means of their multiscale approximations, which are further transformed into a set of visibility graphs. After extracting probability distributions of small motifs, density, assortativity, *etc.*, these features are used for building highly accurate classification models using generic classifiers (*e.g.*, Support Vector Machine and eXtreme Gradient Boosting). Based on extensive experiments on a large number of open datasets and comparison with five state-of-the-art TSC algorithms, our approach is shown to be both accurate and efficient: it is more accurate than Learning Shapelets and at the same time faster than Fast Shapelets.

Finally, we list a few industrial applications that relevant to our research work, including Non-Intrusive Load Monitoring as well as anomaly detection and visualization by means for hierarchical clustering for time series data.

In summary, this dissertation explores different possibilities to improve the efficiency and accuracy of TSC algorithms. To that end, we employ a range of techniques including wavelet transforms, symbolic approximations, language models and graph mining algorithms. We experiment and evaluate our approaches using publicly available time series datasets. Comparison with the state-of-the-art shows that the approaches developed in this dissertation perform well, and contribute to advance the field of TSC.

To my family.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my dissertation advisor Prof. Dr. Yves Le Traon for his continuous support during my doctoral study. He has offered not only full support but also the maximum freedom, so that I can conduct research and explorations in topics I am interested in.

My sincere gratitude also goes to my daily advisors Dr. Jacques Klein and Dr. Tegawendé F. Bissyandé for their guidance and encouragement. Their support has saved me from having an abysmal time and made the completion of this dissertation all possible (instead of dropping out in the middle).

I am grateful for the jury members for their interest in this dissertation and taking their valuable time to evaluate this dissertation. In addition, many people have provided insightful comments for my Ph.D. work. Especially, I would like to acknowledge Prof. Dr. Karl Aberer from EPFL, Prof. Dr. Eamonn Keogh from UCR and Prof. Dr. Jessica Lin from GMU for their helpful suggestions on my previous work on which this dissertation is built.

My special thanks go to Dr. Anne-Marie Solvi, Paul Schummer and colleagues from Paul Wurth Geprolux S.A.. I have had a wonderful time working on the smart buildings and PWBox project with them.

Finally, I could not have gone so far (or even started in the first place) without my beloved wife and my parents in my back. They are the sunshine in my life and their support has always been warm and unreserved. I cannot imagine surviving this journey without their support.

Daoyuan Li
Luxembourg
January 11, 2018

CONTENTS

ACRONYMS

1NN	Nearest Neighbor Classification
AdaBoost	Adaptive Boosting
ARFF	Attribute-Relation File Format
ARIMA	AutoRegressive Integrated Moving Average
aSAX	adaptive SAX
BECM	Building Energy and Comfort Management
BoP	Bag-of-Patterns
BOSS	Bag-of-SFA-Symbols
BoW	Bag-of-Words
CART	Classification and Regression Trees
CNN	Convolutional Neural Networks
COTE	Collective of Transformation-based Ensembles
CV	Cross Validation
DFT	Discrete Fourier Transform
DM	Data Mining
DSCo	Domain Series Corpus
DSCo-NG	Next Generation Domain Series Corpus
DTW	Dynamic Time Warping
DWT	Discrete Wavelet Transform
ECG	Electrocardiogram
EDR	Edit Distance on Real sequence
ED	Euclidean Distance
EEG	Electroencephalogram
FS	Fast Shapelets
FSM	Finite State Machine
HIVE-COTE	Hierarchical Vote Collective of Transformation-based Ensembles
HPC	High Performance Computing
HVAC	Heating, Ventilation and Air Conditioning
HVG	Horizontal Visibility Graph
IoT	Internet of Things
iSAX	indexable SAX
kNN	k Nearest Neighbors
LM	Language Model

LOOCV	Leave-One-Out Cross Validation
LSTM	Long Short-Term Memory
MJC	Minimum Jump Cost
MPD	Motif Probability Distribution
MVG	Multiscale Visibility Graph
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NILM	Non-Intrusive Load Monitoring
NN	Neural Networks
PAA	Piecewise Aggregate Approximation
PLR	Piecewise Linear Representation
PIP	Perceptually Important Points
PCA	Principal Component Analysis
PGD	Parallel Parameterized Graphlet Decomposition
ResNet	Residual Networks
RF	Random Forest
RNN	Recurrent Neural Networks
RPM	Representative Pattern Mining
SAX	Symbolic Aggregate approXimation
SFA	Symbolic Fourier Approximation
SGD	Stochastic Gradient Descent
ST	Shapelet Transform
SVM	Support Vector Machine
TF-IDF	Term-Frequency Inverse Document Frequency
TSBF	Time Series Bag-of-Features
TSC	Time Series Classification
TWED	Time Warp Edit Distance
UVG	Uniscale Visibility Graph
VG	Visibility Graph
WVG	Weighted Visibility Graph
XGBoost	eXtreme Gradient Boosting

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF FIGURES

Part I

OVERVIEW

1 INTRODUCTION

> There will come a time when you believe everything is finished; that will be the beginning.
>
> *Louis L'Amour*
> *Lonely on the Mountain*

1.1 MOTIVATION

Traditionally, time series data are a class of temporal objects or observations that are ordered chronologically. They can be frequently found in a variety of domains, including financial data analysis [Flanagan and Lacasa, 2016], medical and health monitoring [Wong et al., 2014; Samiee et al., 2015], industrial automation applications [Wang et al., 2016a]. In the financial domain, transactions of a stock, currency or commodity forms the fluctuation of one financial instrument; and these fluctuations can naturally be modeled as time series. Typical medical applications of time series analysis include monitoring analysis, and diagnosis of bio-signals such as electrocardiograms (ECG) and electroencephalograms (EEG). Moreover, in recent years, there has been a rapid development and adoption of the Internet of Things (IoT) concept. As a result, with a large amount of heterogeneous sensors deployed to monitor every aspect of our surrounding environments and industrial production lines, a huge amount of time series data are being generated everyday.

More recently, the research community as well as practitioners have seen a great increase of interests modeling non-temporal observations as time series. It has been shown to be feasible to conduct video retrieval, image retrieval, handwriting recognition and text mining tasks by transforming various types of data into time series [Ratanamahatana and Keogh, 2004] and approximating these tasks with specific time series mining techniques. For instance, Chen et al. [2014] have designed sensors to capture the incidental flight sound of insects and extract time series features from them in order to classify different types of insects. Besides, it seems feasible to model software systems as time series take advantage of time series classification (TSC) techniques for malware detection and classification [Kang et al., 2016; Wojnowicz et al., 2017].

Given a large corpus of time series data – either strictly temporal or simply sequential, extracting useful knowledge from such data is a popular research topic but remains a challenging one. These tasks are often referred to as time series

mining or time series analysis. Due to the abundance of time series data and the ever-increasing interests in them, a great amount of research and development attempts have been ignited. This momentum has in turn resulted in a wealth of techniques in the literature. As a sub-domain in the field of generic data mining (DM), tasks in time series mining resemble those of generic DM approaches. Generally, time series mining tasks fall into one of the following categories [Esling and Agon, 2012]:

(a) **Classification.** The classification task seeks to learn from a labeled training set and tries to assign labels to each series of a testing set. Clearly, classification consists of learning the distinctive features that distinguish one class of instances to another; then this extracted knowledge can be applied to instances without explicit labels. It is a typical data mining task and maybe one of the most frequently applied time series tasks. Applications of TSC include handwriting recognition and gesture recognition, as well as other generic pattern recognition tasks.

(b) **Clustering.** Clustering is the process of assign time series instances into groups, called clusters, in a dataset so that the intra-cluster similarity of different instances are maximized and the inter-cluster dissimilarity of groups are maximized. Clustering is a type of unsupervised learning process since time series instances do not need to be labeled beforehand. Time series clustering applications often relates to exploratory tasks that help users to reorganize time series datasets and finding potential patterns as well as anomalies [Liao, 2005].

(c) **Motif discovery.** Motif discovery tries to find from a long time series all the subsequences that appears recurrently. Unlike clustering that groups similar instances into one cluster, motif discovery first extracts possible subsequences from a single time series instance and then looks for similar or identical subsequences occurred repeatedly. Its applications are mainly related to those investing the internal characteristics of long series.

(d) **Prediction/Forecasting.** Given the historical values of a time series, it is desirable to predict the next following values. This is especially beneficial in the financial (*e.g.*, stock price prediction) and industrial (*e.g.*, trajectory analysis and prediction) domain. Time series prediction are commonly implemented via the means of curve fitting and the use of autoregressive integrated moving average (ARIMA) models [Zhang, 2003].

(e) **Outlier detection.** There can be different lines of research in outlier detection of time series data. The first focuses on the detection of abnormal subsequences in a series, which are often conducted with statistic models combined with ARIMA models. Another line involves taking advantage of clustering techniques to locate peer instances are are different from the majority of instances. Typical applications of outlier detection are fraud and intrusion detection [Zhong et al., 2007].

(f) **Query by Content.** This problem is also known as indexing time series from a database. Query by content involves locating the most similar time series in a database according to a given input of time series instance. The main challenge of this task is to define the similarity between two instances. While researchers have proposed many similarity measures, Dynamic Time Warping (DTW) [Berndt and Clifford, 1994] has proven to be one of the most effective similarity measures for time series [Xi et al., 2006].

Obviously, it is impractical to cover all the time series mining aspects in this dissertation. Instead, this dissertation focuses mainly on time series classification due to its wide application scenarios, including speech recognition, handwriting recognition, image classification, non-intrusive load monitoring and so on.

1.2 CHALLENGES

Time series data exists in numerous application domains within our daily life. Especially, as the concept of pervasive computing and Internet of Things (IoT) slowly becomes reality, time series data are generated at an industry scale. For instance, the BLUED non-intrusive load monitoring dataset [Anderson et al., 2012] has been collected from a single household for one week, recording voltage and current measurements with a sampling rate of 12 kHz, leading to a total of tens of billions of time series readings and making it difficult to learn meaningful patterns in real-time. Another power company in the US records a trillion data points every four months; and in astronomy satellites may collect one trillion data points of starlight curves every single day [Rakthanmanon et al., 2012]. Other domains where time series are prevalent include financial applications (e.g. currency exchanges and stock prices), environment monitoring (e.g. weather forecast and disaster monitoring), medical and health care (e.g. electrocardiograms a.k.a. ECGs).

Besides its abundance characteristics, time series data are known for its extremely high dimensionality, *i.e.*, when mapping a time series instance to an n-dimensional space, the number of features or the dimensionality is often very high. This makes general-purpose machine learning algorithms fail or underperform. As a result, in order to learn meaningful knowledge from time series data, efficient machine learning algorithms for time series are desired.

Finally, unlike generic data types where features are more often independent of each other, time series data are intrinsically characterized as sequential. It is thus especially important to taken into account both the local features (*i.e.*, details in specific subsequence characteristics) as well as the global features (*i.e.*, overall curve shapes) when conducting time series mining tasks. To date, how to efficiently and accurately extract meaningful features in time series remains a challenging task.

Due to the abundance of time series data, their high dimensionality and intrinsic sequential characteristics, efficiently mining useful knowledge from time series is still a challenging task. Thanks to recent advances in modern hardware architec-

tures as well as software algorithms, we seek to explore possibilities that are rarely attempted before and address these challenges.

1.3 CONTRIBUTIONS

The main contributions of this dissertation are listed as follows:

(a) We conduct a large scale empirical study that takes advantage of discrete wavelet transform (DWT) for time series dimensionality reduction. We first transform real-valued time series data using different families of DWT. Then we apply dynamic time warping (DTW)-based 1NN classification on 39 datasets and find out that existing DWT-based lossy compression approaches can help to overcome the challenges of storage and computation time. Furthermore, we provide assurances to practitioners by empirically showing, with various datasets and with several DWT approaches that TSC algorithms yield similar accuracy on both compressed (i.e., approximated) and raw time series data. We also show that, in some datasets, wavelets may actually help in reducing noisy variations which deteriorate the performance of mining tasks. In a few cases, we note that the residual details/noises from compression are more useful for recognizing data patterns.

(b) We propose a language model-based approach for TSC named Domain Series Corpus (DSCo). After transforming real-valued time series into texts using Symbolic Aggregate approXimation (SAX). Then we build per-class language models (unigrams and bigrams) from these symbolized text corpora. To classify unlabeled samples, we compute the fitness of each symbolized sample against all per-class models and choose the class represented by the model with the best fitness score. Our work innovatively takes advantage of mature techniques from both time series mining and NLP communities. Through extensive experiments on an open dataset archive, we demonstrate that DSCo performs similarly to approaches working with original uncompressed numeric data. We further propose DSCo-NG to improve the computation efficiency and classification accuracy of DSCo. Unlike in DSCo where we try to find the best way to recursively segment time series, DSCo-NG breaks time series into smaller segments of the same size, this simplification of the classification process also leads to simplified language model inference in the training phase and slightly higher classification accuracy.

(c) We present a multiscale visibility graph representation for time series as well as feature extraction methods for time series classification (TSC). Unlike traditional TSC approaches that seek to find global similarities in time series databases (*e.g.*, Nearest Neighbor with Dynamic Time Warping distance) or methods specializing in locating local patterns/subsequences (*e.g.*, shapelets), we extract solely statistical features from graphs that are generated from time series. Specifically, we augment time series by means of their multiscale approximations, which are further transformed into a set of visi-

bility graphs. After extracting probability distributions of small motifs, density, assortativity, *etc.*, these features are used for building highly accurate classification models using generic classifiers (*e.g.*, Support Vector Machine and eXtreme Gradient Boosting). Thanks to the way how we transform time series into graphs and extract features from them, we are able to capture both global and local features from time series. Based on extensive experiments on a large number of open datasets and comparison with five state-of-the-art TSC algorithms, our approach is shown to be both accurate and efficient: it is more accurate than Learning Shapelets and at the same time faster than Fast Shapelets.

Besides, we also list a few industrial applications that relevant to our research work towards the end of this dissertation, including Non-Intrusive Load Monitoring as well as anomaly detection and visualization by means of hierarchical clustering for time series data. Overall, this dissertation explores different possibilities to improve the efficiency and accuracy of TSC algorithms. To that end we employ a range of techniques including wavelet transforms, symbolic approximations, language models and graph mining algorithms. We experiment and evaluate our approaches using publicly available time series datasets. Comparison with the state-of-the-art shows that the approaches developed in this dissertation perform well, and contribute to advance the field of TSC.

The research work we have performed during the course of the PhD program has lead to the following publications:

- [Li et al., 2016c] D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. Time Series Classification with Discrete Wavelet Transformed Data: Insights from an Empirical Study. In The 28th International Conference on Software Engineering and Knowledge Engineering (SEKE 2016), pages 273–278, July 2016.
- [Li et al., 2016b] D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. Time Series Classification with Discrete Wavelet Transformed Data. In International Journal of Software Engineering and Knowledge Engineering, volume 26, pages 1361–1377. World Scientific, November & December 2016.
- [Li et al., 2016d] D. Li, T. F. Bissyandé, S. Kubler, J. Klein, and Y. Le Traon. Profiling Household Appliance Electricity Usage with N-Gram Language Modeling. In The 2016 IEEE International Conference on Industrial Technology (ICIT 2016), pages 604–609. IEEE, March 2016.
- [Li et al., 2016e] D. Li, L. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. DSCo: A Language Modeling Approach for Time Series Classification. In P. Perner, editor, Machine Learning and Data Mining in Pattern Recognition: 12th International Conference, MLDM 2016, New York, NY, USA, pages 294–310. Springer International Publishing, July 2016.
- [Li et al., 2016a] D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. DSCo-NG: A Practical Language Modeling Approach for Time Series Classification. In The 15th International Symposium on Intelligent Data Analysis (IDA 2016), October 2016.

- [Li et al., 2017] D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. Sensing by Proxy in Buildings with Agglomerative Clustering of Indoor Temperature Movements. In The 32nd ACM Symposium on Applied Computing (SAC 2017), pages 477–484, April 2017.
- [Li et al., 2018] D. Li, J. Lin, T. F. Bissyandé, J. Klein, and Y. Le Traon. Extracting Statistical Graph Features for Accurate and Efficient Time Series Classification. In The 21st International Conference on Extending Database Technology (EDBT), March (To Appear) 2018.

1.4 ORGANIZATION OF THIS DISSERTATION

The structure of this dissertation is organized as follows. We lay down the foundations of this dissertation by introducing the related background in Chapter 2 and state-of-the-art in Chapter 3. Next, we explore different approaches for improving the efficiency and accuracy of TSC tasks. Specially, we conduct studies on taking advantage of Discrete Wavelet Transform (DWT) for time series dimensionality reduction in Chapter 4. In Chapter 5, we employ symbolic representations for time series and building per-class language models for classification. And in Chapter 6, we bring up a graph-based TSC approach that extracts features from multiscale time series visibility graphs so that such unordered features can be fed into generic classifiers (such as RF, SVM and XGBoost) for accurate classification. We also introduce the applications of TSC (in Chapter 7) in the domain of Non-Intrusive Load Monitoring (NILM) and time series clustering (in Chapter 8) for anomaly and indoor activity detection. Finally, we conclude this dissertation in Chapter 9 with directions for future work. For the sake of readability, each chapter of this dissertation is structured to be as independent as possible, so that readers can directly dive into chapters that interest them most without the need of consulting other chapters.

Part II

TIME SERIES

2

BACKGROUND

> You'd better have a firm foundation when you go out into the world. There's no telling what you'll run into.
>
> *Ann B. Ross*
> *Miss Julia to the Rescue*

Outline
In this chapter, we introduce the core ideas and definitions for time series mining and especially for time series classification. Specifically, we give the commonly accepted definitions for time series and its related operations including dimensionality reduction and distance calculation. We also introduce the concept of symbolic representation including SAX and related techniques.

2.1 TIME SERIES

Traditionally, time series refer to a sequence of numbers that are chronologically ordered:

Definition 2.1 (Time series with time stamps)
A traditional time series sample or instance T *is a temporally ordered sequence of* n *real-valued variables,* i.e., $T = ((t_1, v_1), ..., (t_n, v_n)), v_i \in \mathbb{R}, \forall i, j$ *such that* $i < j,\ t_i < t_j$.

In the data mining community, however, time series have a much broader scope and do not associate strictly with timestamps:

Definition 2.2 (Generic time series discussed in this dissertation)
A time series sample or instance T *is an ordered sequence of* n *real-valued variables,* i.e., $T = (v_1, ..., v_n), v_i \in \mathbb{R}$.

If we consider each point in a time series as a feature, then time series data usually have a lot of features. When considering these features as a vector in an n-dimensional space, time series data are often high dimensional.

Definition 2.3 (Time series dimensionality)
The dimensionality of a time series sample T *is the length of* T*, denoted by* $|T|$.

Due to difficulties to conduct data mining tasks on high dimensional data, it is frequently required to reduce the dimensionality of time series in order to improve computation efficiency:

Definition 2.4 (Time series dimensionality reduction)
Given a time series $T = (v_1, ..., v_n)$*, an approximated representation of* T*,* T' *is another time series sample such that* $|T'| \ll |T|$.

The research community has proposed a number of dimensionality reduction techniques for time series, including sampling [Åström, 1969], Piecewise Linear Representation (PLR) [Keogh, 1997], Piecewise Aggregate Approximation (PAA) [Keogh and Pazzani, 2000] and so on. Among them PAA is probably one of the simplest and most widely approaches. PAA can reduce the time series $T = (v_1, ..., v_n)$ from n dimensions to s dimensions by dividing the data into s segments of equal size. The data reduction representation is then a vector of the mean values of the data readings per segment [Keogh and Pazzani, 2000; Lin et al., 2007]. Let $\bar{T} = (\bar{v}_1, ..., \bar{v}_s)$ be this vector where each $\bar{v}_i$ is computed by equation 2.1.

$$\bar{v}_i = \frac{s}{n} \sum_{k=\frac{n}{s}(i-1)+1}^{\frac{n}{s}i} v_k \tag{2.1}$$

Definition 2.5 (Multivariate time series)
A Multivariate time series is a set of more than one time series (variables) that share the same time range.

2.2 DISTANCE MEASURES

One of the core routines in many time series mining tasks – *e.g.*, in distance-based classification and clustering – involves evaluating the dissimilarity or similarity of two time series:

Definition 2.6 (Time series dissimilarity metric)
The dissimilarity measure $D(X, Y)$ *of two time series samples* X *and* Y *is the distance of two time series. This distance is nonnegative,* i.e., $D(X, Y) \geqslant 0$*. If this measure satisfies the additional symmetry property* $D(X, Y) = D(Y, X)$ *and triangle inequality* $D(X, Z) \leqslant D(X, Y) + D(Y, Z)$*, the distance measure is considered to be a metric. Time series metrics are beneficial for indexing and querying a time series database.*

There are a number of dissimilarity measures and metrics for time series, two of the most frequently used measures in the research community are Euclidean distance and DTW distance. The Euclidean distance is defined in Equation 2.2. As illustrated in Figure 2.1, the Euclidean distance maintains a one-to-one mapping of all the points in two series. The gray dotted lines indicating the point mappings are all vertical.

$$D_{Euclidean}(X, Y) = \sqrt{\sum (X_i - Y_i)^2} \quad (1 \leqslant i \leqslant |X| = |Y|) \tag{2.2}$$

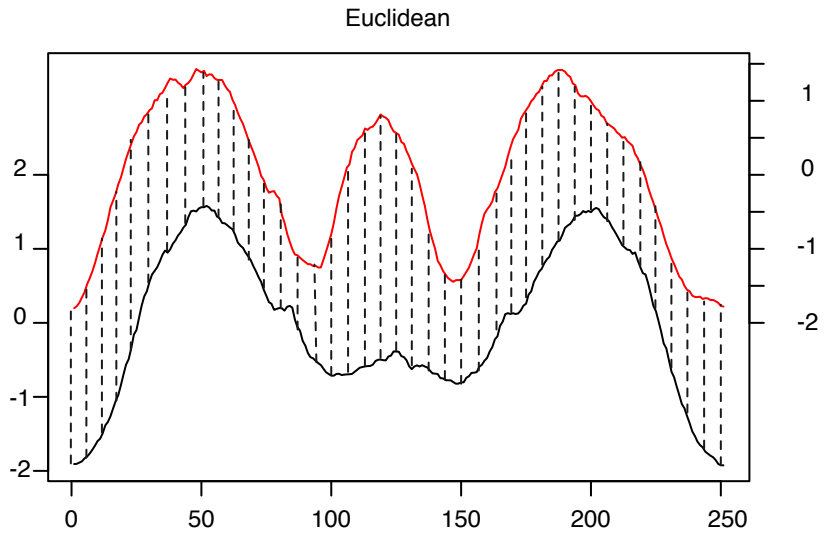

Figure 2.1: Example of time series alignment with the Euclidean distance.

On the other hand, the DTW distance tries to find the best mapping of points in two series using the dynamic programming paradigm, so that the minimum distance between these two series is achieved. The paradigm is called "time warping" since the time axis of series can be expanded or compressed in order to ensure the minimum distance. Figure 2.2 illustrates the alignment of different points. As shown, an i^{th} point in X can be mapped to a j^{th} point (it is possible that $i \neq j$), or one point in X may even be mapped to multiple points in Y.

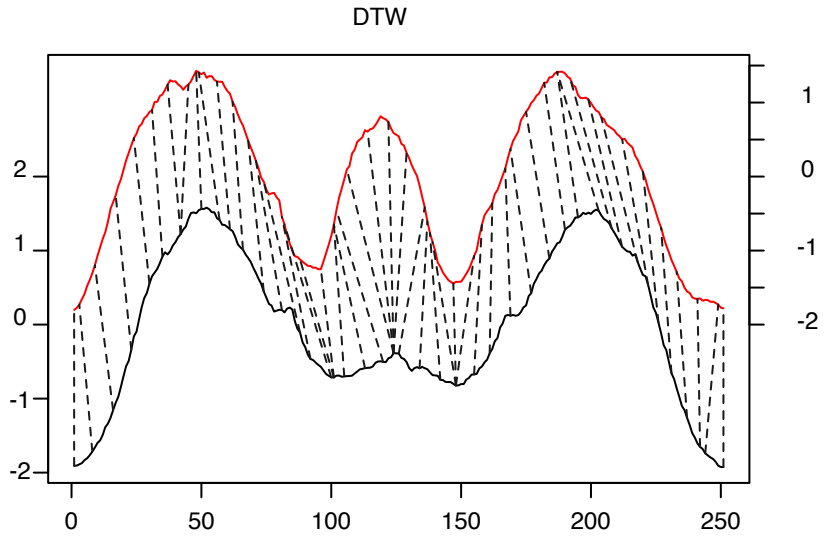

Figure 2.2: Example of time series alignment with DTW distance.

Formally, the DTW distance of two time series can be defined as in Equation 2.3, *i.e.*, the distance of X and Y is the final result of comparing till the end of each

time series, given the pre-condition specified in Equation 2.4 and the recursive computational process in Equation 2.5.

$$D_{DTW}(X, Y) = D_{DTW}(X_m, Y_n) \quad (m = |X| \text{ and } n = |Y|) \tag{2.3}$$

$$D_{DTW}(X_1, Y_1) = |X_1 - Y_1| \tag{2.4}$$

$$D_{DTW}(X_i, Y_j) = |X_i - Y_j| + min \begin{cases} D_{DTW}(X_{i-1}, Y_j) \\ D_{DTW}(X_{i-1}, Y_{j-1}) \\ D_{DTW}(X_i, Y_{j-1}) \end{cases} \quad (1 < i \leqslant |X| \text{ and } 1 < j \leqslant |Y|) \tag{2.5}$$

It is obvious that DTW has a computation complexity of $O(mn)$, where m and n are the dimensionality of two time series instances. In order to speed up the computation, lower bounding techniques have been proposed. One of the most widely used lower bounding algorithms is LB_{Keogh} [Keogh and Ratanamahatana, 2005; Rakthanmanon et al., 2012]. As defined in Equation 2.6, the LB_{Keogh} algorithm limits the scope of the best match point within the upper and lower bounds.

$$LB_{Keogh}(X, Y) = \sum_{i=1}^{|X|} \begin{cases} (X_i - U_i)^2 & \text{if} \quad X_i > U_i \\ (X_i - L_i)^2 & \text{if} \quad X_i < L_i \\ 0 & otherwise \end{cases} \tag{2.6}$$

Specifically, the upper and lower bound of time series Y with the warping window size of r are defined in Equation 2.7 and Equation 2.8. Essentially, the upper and lower bound at a specific time step is the minimal and maximal values of a series within r steps.

$$U_i(r) = max(Y_{i-r}, ..., Y_{i+r}) \quad (r < i \leqslant |Y| - r) \tag{2.7}$$

$$L_i(r) = min(Y_{i-r}, ..., Y_{i+r}) \quad (r < i \leqslant |Y| - r) \tag{2.8}$$

Research has shown that with LB_{Keogh}, the computational complexity can be approximately lowered to $O(n)$ [Smith and Craven, 2008]. One issue with DTW and its lower bounding techniques, however, is that the family of DTW algorithms focuse on finding *global* similarities, i.e., the overall curve *shape* of two time series in the time dimension. As a result, it requires applications to specify a proper warping window size or to properly align data samples. Figure 2.3 shows an example where DTW fails to identify two different subsequences cropped from the

same parent-curve due to different data alignment, i.e., a phase shift. Specifically, plots b and c are segments in a with the same length. However, the DTW distance between b and c is 8.37 (warping window size 5), larger than the DTW distance between b and d or c and d, where the distances are both 4.47. In this specific case, all test samples resembling the overall shape of b or c may be mistakenly labeled as class d, leading to poor classification accuracy.

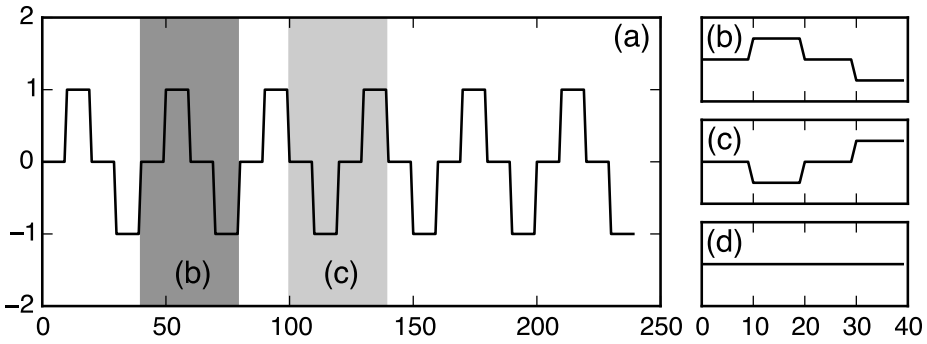

Figure 2.3: An example to illustrate DTW's phase issue.

2.3 SYMBOLIC REPRESENTATION OF TIME SERIES

In the literature of time series data mining, real valued data are sometimes transformed into symbolic representations, so as to potentially benefit from the enormous wealth of data structures and algorithms made available by the text processing and bioinformatics communities. Besides, symbolic representation approaches make it easier to solve problems in a streamed manner [Lin et al., 2007]. Finally, many algorithms target discrete data represented by strings over floating point numbers.

Consider the case of two samples from the `BirdChicken` [Chen et al., 2015] dataset illustrated in Figure 2.4, where bird/chicken images have been transformed into time series (illustrated in gray curves). By observing the readings in different segments of the time series and which segment succeeds another, one can immediately summarize the characteristics of each class. For computers, however, it can be fairly difficult to acquire such knowledge.

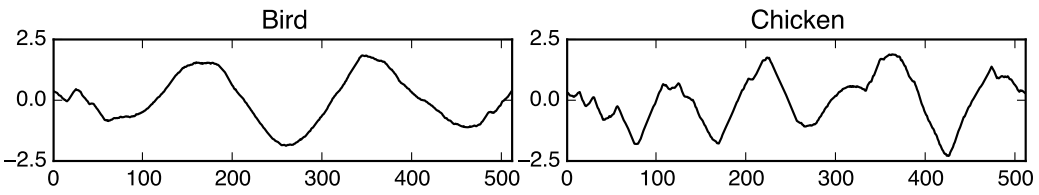

Figure 2.4: Illustrative time series samples from the `BirdChicken` dataset.

In order to improve computation efficiency and remove potential noises from time series, it is common practice to conduct dimensionality reduction. Figure 2.5

illustrates the effect of PAA – one of the most widely applied time series dimensionality reduction approaches as defined in euqation 2.1 – corresponding to the readings in Figure 2.4.

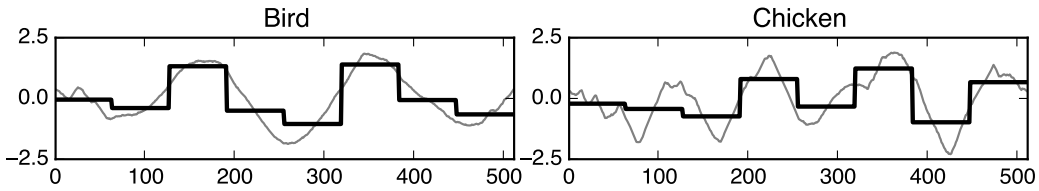

Figure 2.5: PAA representations of two time series samples where dimension is reduced from 512 to 8 ($n = 512$ and $s = 8$).

Although very simple, the PAA dimensionality reduction can greatly reduce the dimensionality of time series data. In order to benefit from the plethora of algorithms that exist in the NLP field, we can transform the PAA representation into a more symbolic representation with alphabets. To that end, the Symbolic Aggregate approXimation (SAX) [Lin et al., 2003, 2007] have been proposed. SAX was initially brought up to transform real valued time series data into a sequence of alphabets, i.e., a string. It has then been proven especially efficient for motif (repeated patterns) discovery tasks. For example, it is advantageous to use SAX in order to find variable-length motifs [Li and Lin, 2010; Senin et al., 2014]. Figure 2.6 illustrates the SAX representations of samples from the two classes in the `BirdChicken` dataset.

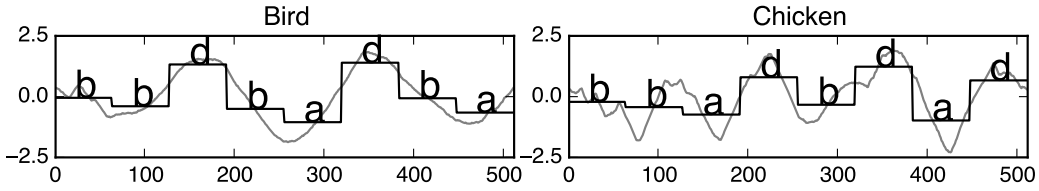

Figure 2.6: SAX representations of two time series samples with an alphabet size of four.

Essentially, SAX assumes a normal probability distribution of time series values and tries to divide the area under the probability distribution curve into smaller areas of equal size. Each of these small areas can then be mapped to a single alphabet so that each area represents an equiprobable interval. For simplicity, the boundaries of these areas can be precomputed and provided as a lookup table (*e.g.*, [Lin et al., 2007] provides a look up table for alphabet size from 3 to 10). For instance, for SAX alphabet of 4, the boundaries (denoted as β) are $\beta = (-0.67, 0, 0.67)$. As a result, transforming raw time series into SAX strings can be extremely efficient.

After transforming real-valued time series data into symbolic strings, one might wonder how to evaluate the similarity of two time series. To that end, a SAX distance measure has been proposed resembling the Euclidean distance. We denote

the PAA representation of two time series X and Y respectively as $\hat{X}$ and $\hat{Y}$, and the corresponding SAX strings as $\bar{X}$ and $\bar{Y}$, then $D(X,Y) \approx D_{PAA}(\hat{X},\hat{Y}) \approx D_{SAX}(\bar{X},\bar{Y})$:

$$D_{PAA}(\hat{X},\hat{Y}) = \sqrt{\frac{|X|}{|\hat{X}|}} \sqrt{\sum_{i=1}^{|\hat{X}|} (\hat{X}_i - \hat{Y}_i)^2} \tag{2.9}$$

$$D_{SAX}(\bar{X},\bar{Y}) = \sqrt{\frac{|X|}{|\bar{X}|}} \sqrt{\sum_{i=1}^{|\bar{X}|} dist(\bar{X}_i - \bar{Y}_i)^2} \tag{2.10}$$

The $dist$ function again can be precomputed and defined as a lookup table, where $dist(i,j)$ is defined as:

$$dist(i,j) = \begin{cases} 0 & \text{if} \quad |i-j| \leqslant 1 \\ \beta_{max(i,j)-1} - \beta_{min(i,j)} & otherwise \end{cases} \tag{2.11}$$

More recently, other symbolic representation methods and improvements have been proposed. For instance, indexable SAX (iSAX) [Shieh and Keogh, 2008] is a superset of SAX with support for time series indexing and fast similarity search. Besides, adaptive SAX (aSAX) [Pham et al., 2010] tries to combine and the k-means algorithm to optimize the performance of SAX on datasets that are not strictly Gaussian-distributed. Finally, Symbolic Fourier Approximation (SFA) [Schäfer and Högqvist, 2012] takes advantage of Discrete Fourier Transform (DFT) instead of PAA for approximation and employs a novel discretization technique named multiple coefficient binning to improve pruning of the search space during time series queries.

3

STATE-OF-THE-ART

For the man who studies to gain insight, books and studies are merely rungs of the ladder on which he climbs to the summit of knowledge. As soon as a rung has raised him up one step, he leaves it behind.

Arthur Schopenhauer
The World as Will and Representation

Outline
This chapter briefly surveys the most relevant classification algorithms as well as datasets for benchmarking TSC approaches. Specifically, we introduce both generic classification algorithms (*e.g.*, kNN, SVM, RF and XGBoost) and classifiers specific to time series (*e.g.*, SAX-VSM, BOSS and COTE). Datasets used in this dissertation are mainly from the UCR time series classification archive.

3.1 GENERIC CLASSIFICATION ALGORITHMS

Classification in data mining refers to a type of supervised machine learning task that involves extracting useful information from a set of labeled data (the training set) and apply such information to predict the labels of data that are not labeled. It is similar to regression tasks, with the exception that the labels are categorical in classification tasks. As one of the most important machine learning tasks, a plethora of algorithms have been proposed. In this section we only survey the most important or relevant classifiers to time series mining.

3.1.1 k*NN*

Given an unlabeled instance and a training set consisting of m samples, kNN tries to find k samples – nearest neighbors – from these samples that closest with the unlabeled instance based on a distance measure, *e.g.*, Euclidean. Once the nearest-neighbor set is obtained, the testing instance is assigned the label of the majority class of its nearest neighbors. kNN is a type of instance-based learning method, *i.e.*,

it compare instances to instances instead of learning an abstract models from the training set. It is considered to be an extremely popular algorithm in data mining due to its simplicity [Wu et al., 2008].

kNN's main advantage is its simplicity: it is extremely easy to understand and to implement. But it has some very obvious drawbacks including its inability to scale with large training datasets, since each sample in the test set has to compare with every sample in the training set. Besides, kNN does not explicitly model anything from the training set, and although it may find the most similar objects to the testing instance, it has difficulty providing extra insights (abstracted knowledge) about the data. As a result, kNN is a *lazy* machine learning algorithm.

3.1.2 *Support Vector Machine*

The core of Support Vector Machine (SVM) is finding a hyperplane that best separates data points from different classes. With 2-dimensional data, the separating plane can be simply a line; when generalizing to n-dimensional data, the hyperplane will be a plane that has $n-1$ dimensions. To find the hyperplane, SVM needs to define what is the best way to separate data from different classes: generally it is defined as the plane that maximizes the margin of the points that has the smallest margin to this plane, *i.e.*, the points that are closest to the plane.

SVM is known for its effectiveness with high dimensional data and especially, it can remain effective even if the dimensionality is greater than the number of samples. It is also memory efficient, since SVM uses a subset of training points in the support vectors. Finally, SVM is very easy to extend by means of choosing predefined kernel methods or customizing one. Thanks to these advantages, SVM is very frequently used in practice.

3.1.3 *Decision Trees*

Decision trees are one of the most straightforward classification approaches that resemble how human beings learn and think. They try to establish a set of rules that best separate data from different classes. Each non-leaf node in the tree is associated with a feature test also known as a split, since data are split into different subsets according to their corresponding values on the feature test. Generally, decision trees are very easy to understand and interpret since they can be visualized in a simple flowchart form. They are also capable of handling missing values and do not require prior imputation.

Depending on how decision trees are developed, there are a range of similar algorithms. For example, ID3 tries to find for each node the categorical feature that yields the largest information gain for the target classes. C4.5 builds on ID3 and converts the output of ID3into sets of if-then rules. Then the accuracy of each rule is evaluated to determine the order they will be applied. Also, CART (Classifica-

tion and Regression Trees) is very similar to C4.5 but constructs binary trees using the feature and threshold that yield the largest information gain at each node. It also adds support for regression and does not compute rule sets.

Although a effective and popular classifier, decision trees tend to overfit on data with a large number of features. As a result, it is common practice to conduct principal component analysis (PCA) on high dimensional features. A well known decision tree-based ensemble approach – rotation forest [Rodriguez et al., 2006] – develops a number of trees and to grow each tree, it takes advantage of PCA on a random subset of the input features.

3.1.4 *Ensemble Methods*

In order to improve the generalization ability and classification accuracy of classifiers, ensemble methods are brought up. Essentially, ensemble methods train a set of classifiers instead of only one and refer to the collective results during classification. In general, there are two major ways to build ensemble models, *i.e.*, through boosting or bootstrap aggregating often abbreviated as bagging.

The idea behind boosting is quite simple: a set of classifiers are built sequentially, with the later classifiers focusing more on the mistakes of earlier classifiers. This process can be considered as a evolution of the same classifier within a limited amount of cycles. Although boosting can make weak classifiers stronger, they are also more likely to overfit the training data. AdaBoost [Freund and Schapire, 1995] is a popular approach that falls in this category.

Bagging, on the other hand, works by initially assigning each model in the ensemble vote with a equal weight. Subsequently, a subset of the training set is randomly drawn and this subset of data is used to train different models independently in parallel. After repeating this process a number of times and every model in the ensemble are trained, during classification each base classifier's output is collected and undergone a voting and the winner label is produced as the final prediction. Bagging can often yield better accuracy as well as generalization capability. Another benefit of bagging is that they are inherently favorable to parallelism, and the training speed can be easily accelerated using parallel computing environments. Random Forest (RF) [Breiman, 2001] is a popular representative of bagging ensemble.

Fernández-Delgado et al. [2014] have conducted an extensive empirical evaluation of 179 classifiers from 17 different families, and found out that RF achieved the best overall accuracy based on the UCI machine learning classification database. Following RF, SVM with Gaussian kernel also performs reasonably well. Overall, RF and SVM are the bests classification families.

Very recently, eXtreme Gradient Boosting (XGBoost) [Chen and Guestrin, 2016] has been proposed and gained a great momentum in adoption in the DM com-

munity and especially in Kaggle[1] competitions: a majority of winning solutions in Kaggle have used XGBoost. XGBoost can achieve extremely high classification performance thanks to a few improvements on the state-of-the-art approaches. First of all, it utilizes a regularized objective function that leads to better parallelism as well as selection of models with simple and predictive functions. Besides, it employs a shrinkage technique [Friedman, 2002] in order to reduce the influence of each individual tree and leave space for future trees to improve the model. Finally, XGBoost borrows the column (feature) subsampling technique from RF to combat overfitting issues.

3.1.5 *Neural Networks*

Neural Networks (NNs) are inspired by the biological structure of neural networks in our brains. The basis construction unit of NNs is called a cell, which takes a number of inputs from other cells and outputs a single signal that may be attached to another cell. Even if a cell may only be able to model simple signals like linear regression, when many cells are combined into a single structure, they can be used for modeling complex signals.

As hardware platforms develop rapidly, researchers are able to create extremely complex models consisting of many cells organized in different layers. Depending on how NNs are constructed, there can be many types of them. Promising ones include recurrent neural networks (RNNs) and convolutional neural networks (CNNs). RNNs allows loops in their cells, so that information may persist in such cells. One prominent type of RNN is called Long Short-Term Memory (LSTM), whose recurrent components do not use activation functions, so that it can remembers values for either long or short time periods. LSTM networks are inherently favorable to sequential and it performs extremely well for speech recognition [Graves et al., 2013] tasks. Compared to RNNs that are suitable for mining of sequential data, CNNs are generally a good fit for image recognition tasks. For instance, deep residual networks (ResNets) [He et al., 2016] have become the state-of-the-art approach for image recognition.

Although proven to produce highly accurate classification results, NNs and especially deep NNs replies heavily on the underlying hardware platform: using GPUs can be magnitudes faster than using CPUs but they can still be time-consuming to train. Furthermore, they are known for overfitting issues and are in general best suitable for extremely large datasets.

3.2 TIME SERIES CLASSIFIERS

Time series classification (TSC) is a subclass of generic classification in machine learning that involves learning from existing labelled time series instances (training set) and applying the learned knowledge to assign labels to instances from

1 https://www.kaggle.com/

a testing dataset, where instance classes or labels are often unknown (either this information does not exist or has been intentionally hidden from the classification process).

Definition 3.1 (Labelled time series dataset)
A labelled time series dataset $\mathbb{D}$ *is an unordered collection of time series along with their corresponding labels or classes,* i.e., $\mathbb{D} = \{(T_1, l_{T_1}), ..., (T_n, l_{T_n})\}$. *The size of* $\mathbb{D}$ *is the number of time series samples in* $\mathbb{D}$, i.e., $|\mathbb{D}| = n$.

TSC tasks are especially common in application domains such as image and speech recognition (e.g., for recognizing spoken words), medical diagnosis (e.g., for detecting the type of a heart disease in an ECG signal), gesture detection, and so on. Due to its numerous application scenarios, many techniques have been proposed for TSC, including k-Nearest Neighbors, shapelets [Ye and Keogh, 2009], and bag-of-features [Baydogan et al., 2013]. Among them, the Nearest Neighbor (1NN) approach has been proven to work exceptionally well, especially when using DTW [Batista et al., 2011] for computing the distance metrics between a pair of time series samples.

3.2.1 *Similarity-based Nearest Neighbor*

The Nearest Neighbor (1NN) classification is a special case of the more general k-Nearest Neighbors algorithm (kNN) where k is set to one. Underlying, kNN needs a distance measure to find the nearest neighbors. Traditionally, Euclidean and DTW are the preferred measure. For instance, Batista et al. [2011] claim that DTW-based 1NN is extremely difficult to beat. Other distance measures such as Time Warp Edit Distance (TWED) [Marteau, 2009], Edit Distance on Real sequence (EDR) [Chen et al., 2005] and Minimum Jump Cost (MJC) [Serra and Arcos, 2012] can also lead to accurate 1NN classification. For example, Serra and Arcos [2014] have empirically shown with 45 publicly-available datasets that DTW, EDR and MJC can yield accuracy scores that are not statistically significant.

3.2.2 *Bag-of-Patterns*

There can be a few drawbacks for instance-based classification algorithms, one of the most important issue probably lies in scalability. That is, when the size of training datasets grows, it becomes increasingly inefficient to find similar instances as the algorithms needs to scan every instance within the training set. Furthermore, they are often not adaptive to very long time series. As a result, it is more appropriate to consider the overall statistical features. The bag-of-patterns (BoP) approach [Lin et al., 2012] borrows the bag-of-words methods from the NLP community and adopts a histogram-based representation for time series data. Specifically, BoP first transforms real-valued time series into SAX strings, and then uses a sliding window to scan the strings and curate a set of unique *words* . Word frequencies are counted and saved in a matrix, which is used to compute the

similarity between different time series. BOP can produce very good results even if time series in a dataset are not properly aligned. Time series Bag-of-Features (TSBF) [Baydogan et al., 2013] is similar approach to BoP, but it considers both fixed- and variable-length words as well as shape-based features such as the slope and variance.

3.2.3 *SAX-VSM*

Aiming for providing an interpretable time series classification algorithm that typical 1NN classifier lacks, Senin and Malinchik [2013] propose SAX-VSM as an alternative to 1NN that provides a superior interpretability and a low computational complexity in classification. Similar to BoP, SAX-VSM also takes advantage of SAX to transform time series data into symbolic strings and use the sliding window technique to convert the strings into a set of words, which is represented by the bag-of-words model. Unlike BoP that assigns each word in the model the same weight, SAX-VSM instead adopts a term-frequency inverse document frequency (TF-IDF) paradigm to reduce the impact of too frequent patterns in time series. During the training phase, SAX-VSM builds TF-IDF bags for all classes; and during classification, cosine similarity is used for evaluating the similarity to the class vector and assigning labels.

Thanks to the different weights assigned to words in SAX-VSM, it is able to find out the weight of any arbitrary selected subsequence. SAX-VSM also provides a visualization tool resembling a heatmap that can provide users with an immediate insight into the layout of the more defining characteristics of subsequences in each class.

3.2.4 *Representative Pattern Mining*

Another recent TSC approach based on SAX is Representative Pattern Mining (RPM) [Wang et al., 2016b]. After discretization of time series into symbolic representations, RPM takes advantage of grammatical inference techniques to automatically finds recurrent and correlated patterns of variable lengths. This pool of patterns shared by many instances in a class is further refined so that the most representative patterns that capture the properties of a specific class are selected. Thanks to the SAX discretization method, RPM demonstrates excellent performance on real-world medical time series that are inherently noisy. Furthermore, RPM can be robust on shifted data, since its feature extraction process is rotation invariant.

3.2.5 *BOSS*

It is obvious that several of the aforementioned classifiers transform time series into symbolic strings using SAX. After inventing SFA for symbolic representation of time series, Schäfer [2015] proposes a classification method named Bag-of-SFA-Symbols (BOSS) that is similar to the bag-of-words approach. Since SFA is noise tolerant and invariant to phase shifts, offsets, amplitudes and occlusions, BOSS can be applied to noisy data while achieving good classification results. It is also computationally efficient due to the use of hashing techniques to evaluate the similarity of SFA words. Intuitively, two time series are considered similar in the BOSS distance if they share the same set of SFA words. Using the BOSS distance measure, 1NN can be used to classify unlabeled datasets.

One challenge with BOSS is to choose the appropriate parameters, *i.e.*, the alphabet size and sliding window length. To achieve optimal classification accuracy, it is advised to use the BOSS ensemble method, which consists of a number of basic BOSS classifiers with different parameter combinations. During the training phase, different weights are calculated and assigned to the set of classifiers. When these weights are optimized during training, they are used in to accurately predict the labels of test dataset.

3.2.6 *Shapelets*

Besides instance-based and symbolic transform-based TSC approaches, another line of research tries to find out the most discriminative features called shapelets [Ye and Keogh, 2009] per time series class and take advantage of this feature for classification. Specifically, shapelets are subsequences of time series that are representative of a class. Technically, the retrieval of shapelets is done by optimizing the information gain (the increase of entropy) of dividing time series data into subsequences and choosing the best candidates extracted from the series segments.

Grabocka et al. [2014] propose a new mathematical formalization of the shapelets discovery algorithm by means of a classification objective function and a tailored stochastic gradient learning algorithm. This approach enables learning near-optimal shapelets without trying out a large number of candidates.

Similar to SAX-VSM, shapelets can also produce interpretable classification results by means of matching a shapelet with the testing instance. One drawback with this approach lies in its computation complexity. Although subsequence distance early abandoning and admissible entropy pruning techniques have been proposed, the running time for shapelet discovery algorithm is $O(n^2m^3)$ [Rakthanmanon and Keogh, 2013], where n is the size of the training dataset, and m is the length of the time series instances.

3.2.7 *Logical Shapelets*

Due to the computation complexity of shapelet discovery, finding shapelets are in practice done offline. Logical Shapelets [Mueen et al., 2011] improves the efficiency of shapelets discovery based on intelligent caching and reuse of computations as well as admissible pruning of the search space.

3.2.8 *Learning Shapelets*

Contrary to the procedure of the original shaplet discovery where a shapelet is found through exhaustive or optimized search, Learning Shapelets (LS) [Grabocka et al., 2014] proposes a new mathematical formalization of the task and defines an objective function, so that optimization methods such as stochastic gradient descent (SGD) maybe applied to minimize the loss function and find near-to-optimal shapelets without the need to evaluate a large pool of shapelet candidates.

In addition, since LS defines a distance between a shaplet and a time series, it can find the score of each shapelet independently and then sort them, so that the k-most discriminative shapelets can be found during the optimization process.

3.2.9 *Shapelet Transform*

The original shapelets algorithm tries to find the most representative time series subsequences for all classes in the training dataset and use these shapelets to match unlabeled time series data, forming a decision tree-like classification paradigm. The classification accuracy can be further improved by extracting top-k shapelets per class in the training set to avoid overfitting. To that end, Shapelet Transform (ST) [Lines et al., 2012; Hills et al., 2014] is proposed to find k shapelets from each class and use them to transform datasets by calculating the distances from a series to each shapelet.

After shapelet transforming time series datasets, generic classification algorithms can be used utilized. The authors have experimented with different classifiers and found out that SVM can produce best classification accuracy. Since finding shaplets of a single length costs $O(n^2m^3)$ computation, finding k shapelets is even more time consuming. As a result, ST's high computation complexity can be a real obstacle for practical use.

3.2.10 *Fast Shapelets*

In order to improve the computation efficiency of the shaplets algorithm, Rakthanmanon and Keogh [2013] propose a heuristic-based approach named Fast Shapelets (FS). Although it is an approximate algorithm and is not guaranteed

to return the same shapelets as the original algorithm, it can decrease the complexity of the shaplet discovery algorithm down to $O(nm^2)$. In order to achieve this, FS carries out a range of optimizations including: 1) mapping the real-valued high dimensional time series data into discrete and low dimensional symbolic representations using SAX; 2) using random masking to project SAX words from a high dimension to a lower dimension to reduce false dismissals, which are caused by the fact that two time series that differ a tiny margin may produce different SAX words; and 3) taking advantage of similarity hashing to find and count similar words.

3.2.11 *Collective of Transformation-based Ensembles*

In order to further improve the classification accuracy of TSC algorithms, Bagnall et al. [2015] introduce an ensemble algorithm named Collective of Transformation-based Ensembles (COTE) – an ensemble of 35 different classifiers. It is based on two concepts that 1) transforming time series data into an alternative data space can ease the process of finding discriminatory features and 2) improved accuracy can be achieved through simple ensemble methods even with a single data representation. COTE has proven to be more accurate than all others thanks to the different underlying time series representations as well as the combination effect of various classifiers. This approach, however, has a very high computation complexity, which limits its applications in practice.

The authors of COTE also claim that COTE can be more accurate than deep learning methods such as convolutional neural networks (CNN) [Lines et al., 2016]. Besides, they continue to propose Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE), which incorporates three new classifiers, *i.e.*, an interval based ensemble named Time Series Forest (TSF) [Deng et al., 2013]), a dictionary ensemble (BOSS) [Schäfer, 2015] and a spectral component named Random Interval Features (RIF). Besides, HIVE-COTE proposes a modular hierarchical structure with a probabilistic voting scheme that can be tuned during cross-validation. HIVE-COTE appears to be significantly more accurate than its predecessor, making it on of the most accurate TSC algorithms. However, it also suffers from the same complexity issue similar to its predecessor.

3.3 DATASETS

Although time series data are widely available, clearly labelled time series datasets that are perfect for classification tasks are difficult to come by. The UCI Machine Learning Repository[2] provides various datasets donated by the research community and industry. Time series datasets are also amiable in the UCI repository, but different datasets come with different formats and it takes a lot of efforts to preprocess these datasets into formats that can be consumed by TSC algorithms.

2 https://archive.ics.uci.edu/ml/datasets.html

The UCR time series classification archive[3] [Chen et al., 2015], on the other hand, offers a relatively uniform file format as well as class labels. Datasets contained in this archive are popular within the TSC community and has become the *de facto* dataset archive for TSC, thus allowing for a reliable comparison baseline. Besides, the archive includes error rates for DTW- and Euclidean-based 1NN classification as a performance benchmark for TSC. The UCR archive is composed of two sub-archives: `Pre_Summer_2015_Datasets` and `Newly Added Datasets` which include datasets from various fields, ranging from electrocardiograms (ECG) to intra-species image recognition data. We tested on the latter (which contains 39 different datasets) because its file format and internal data structures are consistent, making it possible to conduct batch processing in a content-agnostic manner. Furthermore, both sub-archives have similar dataset diversity and some datasets for specific domains (for example, ECG data) appear in both sub-archives. Generally, datasets in the `Newly Added Datasets` contain more time series samples than datasets in the `Pre_Summer_2015_Datasets`. Finally, the UCR archive provides a predefined train/test split, so that different TSC algorithms can be benchmarked with exactly the same data. As shown in 3.1, these 39 datasets have various number of classes from 2 to 60 and different number of training and testing instances from 20 to 8,926, with time series lengths varying from 80 to 2,079.

Finally, the UEA and UCR Time Series Classification Repository[4] [Bagnall et al., 2016] contains similar datasets to the UCR archive, with the exception that all the datasets are in Attribute-Relation File Format (ARFF), which is the default input file format for the well-known Java machine learning framework Weka [Frank et al., 2009]. Note that although names of the datasets used in this repository may be exactly the same with the UCR archive, the similarity of their contents is not guaranteed. In fact, the training and testing datasets may have been swapped for a number of datasets. An obvious example is the FordA dataset, where in the UCR archive the training dataset and testing set are of size 1320 and 3601 respectively, while in the UEA & UCR repository the training set has 3601 samples and the testing test has 1320.

To increase the reproducibility of our work, we detail which datasets have been used for each of our experiments. Furthermore, parameters of different algorithms can have a big impact on the final performance, as a result, we discuss about the parameters and hyper-parameters in the next section.

3.4 PARAMETERS AND HYPER-PARAMETERS

Although it is desirable that DM algorithms should have as few parameters as possible [Keogh et al., 2004] in order to improve the generalization and reproducibility of these algorithms, in reality very few algorithms are parameter-free. Take SAX for example, at least two algorithms must be set, *i.e.*, the alphabet size and the PAA

3 http://www.cs.ucr.edu/~eamonn/time_series_data/
4 http://timeseriesclassification.com/dataset.php

Table 3.1: Characteristics of datasets in `Newly Added Datasets` of the UCR archive.

#	Dataset Name	#Classes	#Training	#Testing	Length
1	ArrowHead	3	36	175	251
2	BeetleFly	2	20	20	512
3	BirdChicken	2	20	20	512
4	Computers	2	250	250	720
5	DistalPhalanxOutlineAgeGroup	3	139	400	80
6	DistalPhalanxOutlineCorrect	2	276	600	80
7	DistalPhalanxTW	6	139	400	80
8	ECG5000	5	500	4500	140
9	Earthquakes	2	139	322	512
10	ElectricDevices	7	8926	7711	96
11	FordA	2	1320	3601	500
12	FordB	2	810	3636	500
13	Ham	2	109	105	431
14	HandOutlines	2	370	1000	2709
15	Herring	2	64	64	512
16	InsectWingbeatSound	11	220	1980	256
17	LargeKitchenAppliances	3	375	375	720
18	Meat	3	60	60	448
19	MiddlePhalanxOutlineAgeGroup	3	154	400	80
20	MiddlePhalanxOutlineCorrect	2	291	600	80
21	MiddlePhalanxTW	6	154	399	80
22	PhalangesOutlinesCorrect	2	1800	858	80
23	Phoneme	39	214	1896	1024
24	ProximalPhalanxOutlineAgeGroup	3	400	205	80
25	ProximalPhalanxOutlineCorrect	2	600	291	80
26	ProximalPhalanxTW	6	205	400	80
27	RefrigerationDevices	3	375	375	720
28	ScreenType	3	375	375	720
29	ShapeletSim	2	20	180	500
30	ShapesAll	60	600	600	512
31	SmallKitchenAppliances	3	375	375	720
32	Strawberry	2	370	613	235
33	ToeSegmentation1	2	40	228	277
34	ToeSegmentation2	2	36	130	343
35	UWaveGestureLibraryAll	8	896	3582	945
36	Wine	2	57	54	234
37	WordSynonyms	25	267	638	270
38	Worms	5	77	181	900
39	WormsTwoClass	2	77	181	900

window size. It is possible that assigning a different set of parameters can lead to very different final classification results for algorithms built on top of SAX.

ML and DM algorithms can have not only parameters, but also hyper-parameters. Generally, the former refers to the configuration variables that are internal to a model, and such variables can be obtained or estimated from the data. A simple example is the number of classes parameter in many classifiers. One user may explicitly set such a parameter, although it is not common practice – the classifier should be able to infer such information from the training data. Other common parameters include weights in NNs and support vectors in SVMs. On the other hand, hyper-parameters are external to the model and these values cannot be estimated from the data. As a result, they are often set using heuristics. For instance, the k in kNN classifier can be considered as a hyper-parameter, since "there is no analytical formula available to calculate an appropriate value" [Kuhn and Johnson, 2013]. Another example of hyper-parameters is the learning rate parameter persisting in many classifiers such as NNs and XGBoost.

3.4.1 *Cross Validation*

Learning the parameters for a classifier and finding its best hyper-parameters often involves validating such a classifier to reduce overfitting and improve generalization. This process is often known as model validation, where a model in "tested" with validation data, which are selected from the training set but have not been fitted to the model. To best take advantage of the training dataset, it is common practice to conduct cross validation (CV).

One round of CV partitions the training dataset into a subset for training the classifier and other subset for validation. CV can be either exhaustive or non-exhaustive. In an exhaustive CV, for example, the leave-one-out CV (LOOCV), a single instance is used for validation in each CV round. Since every instance in the training dataset is validated in LOOCV, there has to be n rounds of LOOCV, where n is the size of the training dataset. Obviously, LOOCV (and exhaustive CV in general) can be time-consuming. In an non-exhaustive CV, *e.g.*, the k-fold CV, one k-th of training dataset is used for validation while the rest is used for training. A k-fold CV always go through k rounds and the validation results are then aggregated for evaluation.

3.4.2 *Tuning Hyper-Parameters*

One traditional way of tuning hyper-parameters is grid search [Hsu et al., 2003], which is an exhaustive searching through a subset of the hyper-parameter space for a model. This subset is often manually designated. A grid search process is often optimized following a performance metric – *e.g.*, accuracy score or entropy – that can be obtained through CV. Being exhaustive, grid search can be time-consuming. As a result, non-exhaustive search methods such as random search [Schrack and Choit, 1976] can be used to speed up the process.

Grid search can be helpful in cases where hyper-parameters are discrete. But when hyper-parameters are continuous and their spaces become high dimensional, exhaustive search may be infeasible. In this case other optimization methods Bayesian optimization [Martinez-Cantin, 2014] can be more practical. For instance, Auto-WEKA [Kotthoff et al., 2016] leverages Bayesian optimization for model selection as well as hyper-parameter tuning for classifiers in Weka.

Part III

TRANSFORMING TIME SERIES FOR TSC

4

DISCRETE WAVELET TRANSFORM FOR DIMENSIONALITY REDUCTION

Less is more.

Robert Browning
The Faultless Painter

Outline
We take advantage of DWT for time series dimensionality reduction, so that distance-based TSC approaches can be tuned more efficiently. We have experimented a large number of DWT approaches from different families and found out the most suitable approach for DTW distance-based 1NN. After conducting TSC on compression residuals from wavelet transform, we have also found out that such residuals can be helpful for classification tasks.

4.1 INTRODUCTION

Time series data exists in numerous application domains within our daily life. Especially, as the concept of pervasive computing and Internet of Things (IoT) slowly becomes reality, time series data are generated at industry scale. For instance, the BLUED non-intrusive load monitoring dataset [Anderson et al., 2012] has been collected from a single household for one week, recording voltage and current measurements with a sampling rate of 12 kHz, leading to a total of tens of billions of time series readings and making it difficult to learn meaningful patterns in real-time. Another power company in the US records a trillion data points every four months; and in astronomy satellites may collect one trillion data points of starlight curves every single day [Rakthanmanon et al., 2012]. Other domains where time series are prevalent include financial applications (e.g. currency exchanges and stock prices), environment monitoring (e.g. weather forecast and disaster monitoring), medical and health care (e.g. electrocardiograms a.k.a. ECGs). Besides its abundance characteristics, time series data are known for its extremely high dimensionality, often making general-purpose machine learning algorithms fail or underperform. As a result, in order to learn meaningful knowledge from time series data, efficient machine learning algorithms for time series are desired.

Extracting useful knowledge from time series data – a.k.a. time series mining [Fu, 2011] – is now popular but remains a challenging research topic. In particular, time series classification (TSC) attracts significant interest among the re-

searchers and industry practitioners. TSC approaches often implement supervised learning techniques to classify unknown time series instances based on knowledge gained from existing labeled ones. Due to the abundance and intrinsic high dimensionality of time series data, computing resources such as storage, CPU and memory have become critical bottlenecks in the exploitation of time series. To address these challenges, the research community has proposed efficient dimensionality reduction [Wang and Megalooikonomou, 2008] and representation mechanisms such as SAX [Lin et al., 2007] and Discrete Wavelet Transforms (DWT). DWT is very popular among both researchers and industry practitioners. In general, wavelets are mathematical functions that process raw data to only keep meaningful oscillations in data values. The earliest wavelet was brought up by Haar in 1909, but has undergone great development especially since Ingrid Daubechies [Daubechies, 1988] proved the existence of wavelet families with compact support over an interval [Cohen et al., 1993] and orthogonal translates [Daubechies, 1993]. Wavelets compression techniques have since then been extensively used in image compression and are part of the JPEG 2000 standard [Taubman and Marcellin, 2002]. Wavelet transforms have also been extensively used in medical data analysis including ECG diagnosis [Addison, 2005] and ultrasound image processing [Pizurica et al., 2006]. More recently, researchers have applied wavelet transforms in the field of Non-Intrusive Load Monitoring (NILM) [Duarte et al., 2012; Gray and Morsi, 2015].

Although wavelets are popular in the research community, the literature lacks a large scale empirical study on the impact of wavelet transformation on the performance of time series mining approaches. In this chapter, which extends our previous work [Li et al., 2016c], we seek to investigate this impact on a baseline state-of-the-art time series classification approach. To that end, we compare the classification accuracy of raw uncompressed time series data against transformed data using DWT, including both wavelet approximations and the details, i.e., the residuals or *noises* that are often thrown away during the lossy compression processes. In this way, we are able to separate time series' global features from local defining subsequences. Furthermore, through extensive significance tests, we prove that wavelets can perform better than explicit smoothing techniques in TSC tasks.

In this study, we extensively test how discrete wavelet transforms impact TSC accuracy and computational efficiency using 39 openly accessible datasets. Our study suggests that DWT can indeed be useful in time series classification tasks:

- Wavelet transforms can be used to reduce dimensionality of time series data, while at the same time achieving similar classification accuracy compared to using the original uncompressed data. In fact, we demonstrate that time series dimensionality may be reduced by around 90% while still achieving good classification accuracy.
- Wavelets can be used to reduce noises in time series data, so that better classification performance can be achieved after conducting wavelet transform on the original uncompressed data.

- Wavelets implicitly smoothens time series data, and they can be slightly more effective for time series classification compared to explicit smoothing techniques.
- We have further found that, surprisingly, for a few datasets, classification using the compression residual details can be even more effective than using either the original data or the wavelet approximation. This finding suggests that there are specific datasets which are more distinguishable using local features instead of global ones.

The remainder of this chapter is organized as follows. We introduce the necessary background information in Section 4.2 and related work in Section 4.3. We present experimental details and results in Section 4.4 and study the implicit smoothing effect of wavelets in Section 4.5, before concluding the chapter and outlining future work in Section 4.6.

4.2 DISCRETE WAVELET TRANSFORM

In this section, we present the necessary background to facilitate understanding of this chapter. Specifically, we show how DWT works and especially how DWT can be applied to time series dimensionality reduction.

Wavelets are mathematical functions that resemble the shape of wave oscillations, with the constraint for waves to start at 0 and then oscillate to 0 in the end. Mathematically, wavelets are described using two types of functions: the wavelet function (the mother function denoted with $\psi(t)$) and the scaling function (the father function denoted with $\phi(t)$) [Amolins et al., 2007]. These functions have to satisfy the following conditions:

$$||\psi(t)||^2 = \int |\psi(t)|^2 dt < \infty \tag{4.1}$$

$$\int |\psi(t)| dt < \infty \tag{4.2}$$

$$\int \psi(t) dt = 0 \tag{4.3}$$

$$\int \phi(t) dt = 1 \tag{4.4}$$

The earliest and simplest wavelet function is Haar, whose wavelet function and scaling function are defined as follows:

$$\psi_{Haar}(t) = \begin{cases} -1 & 1/2 \leqslant t < 1, \\ 1 & 0 \leqslant t < 1/2, \\ 0 & \text{otherwise.} \end{cases}$$

$$\phi_{Haar}(t) = \begin{cases} 1 & 0 < t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

As illustrated in Figure 4.1, the Haar wavelet is not a very smooth one, that is, it has a low regularity. Other more sophisticated wavelets – for instance, Daubechies 20 and Symlets 20 – have a higher regularity and thus are able to represent signals more accurately.

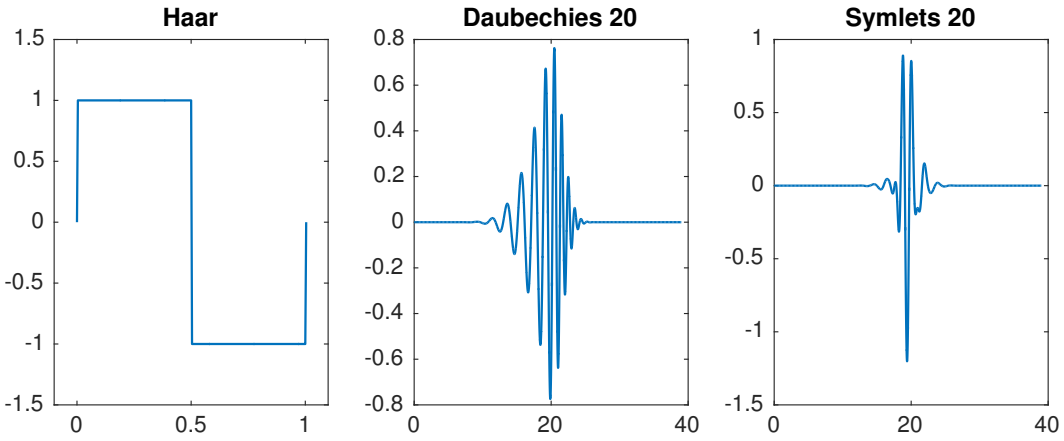

Figure 4.1: Wavelet functions of Haar, Daubechies 20 and Symlets 20.

In practice, the Haar transform is performed as follows. Given a series $T = t_1, ..., t_n$, the Haar transform outputs two series: the approximation A and the details D, where for $1 \leqslant i \leqslant \frac{n}{2}$,

$$A_i = \frac{t_{2i-1} + t_{2i}}{\sqrt{2}} \tag{4.5}$$

$$D_i = \frac{t_{2i-1} - t_{2i}}{\sqrt{2}} \tag{4.6}$$

It is clear that the approximations capture the overall *shape* – the global features – of the original series, while the details are the variances – local features – in time series. Figure 4.2 demonstrates an example of single level, one dimensional Haar transform. As shown, the original signal on the top has been compressed to half of the original size (figure in the middle, note the x-axis label) and the amplitude has been scaled up despite that the overall signal shape is not changed. Note also the grayed area in the signal, it is clear that the approximation becomes smoother after transformation, and that the details shown in the bottom have been dropped. Note also that the approximation becomes smoother, i.e., after residual details are removed, the data points in the time series approximation do not tend to change drastically over time when comparing with the original time series. We will study the smoothing effect of Wavelets later in Section 4.5.

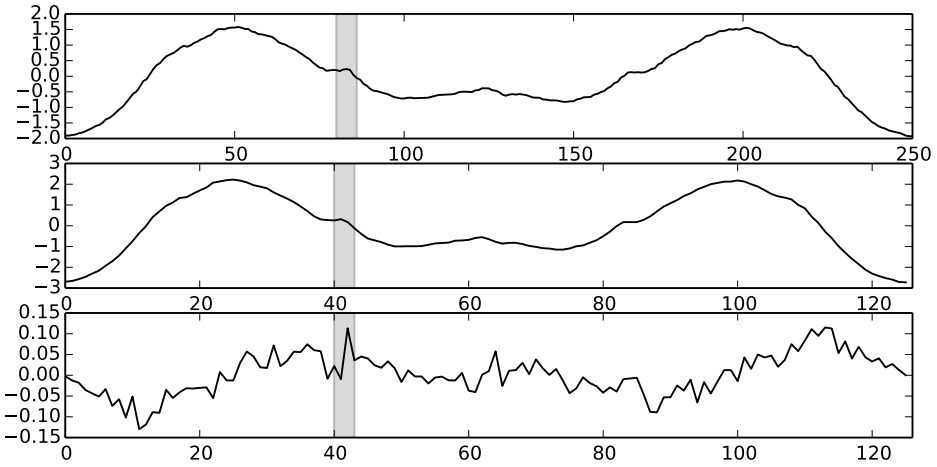

Figure 4.2: Example of Haar transform: the original signal, the Haar approximation and the residual details.

4.3 RELATED WORK

Thanks to wavelets' wide application domains [Leung et al., 1998], there has been research investigating their performance in various domains. However, to the best of our knowledge, impact of different DWT techniques on TSC has not been done in a generic manner. As a result, in this section we enumerate relevant work to ours in the domains of medical applications, image compression, NILM, and so on.

Addison [Addison, 2005] has conducted a review of both continuous wavelet transform (CWT) and discrete wavelet transform (DWT) on ECG data, concluding that DWT is practically easy to use, while CWT – being more complex and difficult to tune parameters – is able to keep a high resolution in the time-frequency plane, which can result in more accurate identification of components. Another study in wavelet applications in the medical field was done by Pizurica et. al. [Pizurica et al., 2006], where the authors reviewed the performance of wavelet denoising specifically in MRI and brain imaging. Amolins et. al. [Amolins et al., 2007] has compared wavelet transforms' performance in image fusion against standard fusion techniques including Intensity-Hue-Saturation (IHS) and principal component analysis (PCA), and found out that even the simplest wavelet-based scheme outperforms IHS and PCA.

Zhu et. al. [Zhu et al., 2009] claim that wavelets are superior to Fourier methods in the field of tool condition monitoring in terms of signal denoising and feature extraction. To disaggregate electric signals, Duarte et. al. [Duarte et al., 2012] take advantage of CWT in order to extract features from voltage transients and use the extracted features for classification. Gray et. al. [Gray and Morsi, 2015] use wavelet-based classification for NILM and claim that "symlets behave in an identical manner as their less symmetric Daubechies representation". And as our em-

pirical study shall demonstrate in the next section, Symlets can actually perform significantly better than Daubechies.

Chan et. al. [Chan et al., 2003] have proposed using Haar for more efficient similarity search, but have not considered other wavelet families. Finally, this work was partly inspired by our previous work, where we have taken advantage of SAX to transform/compress time series data and then build per-class language models to profile household electric appliances [Li et al., 2016d] and conduct general-purpose time series classification [Li et al., 2016e,a]. When classifying, we compare time series against models instead of known samples. Especially, we have used SAX to conduct dimensionality reduction before converting real-valued time series into strings of alphabets. And our empirical experiments have suggested that dimensionality reduction can be pushed more using DWT, thus we provide such results in this study.

4.4 EXPERIMENTAL STUDY

In this section, we present our experimental setup and the collected results. In order to facilitate reproducibility, we opt to experiment on publicly available datasets, and further open source our own implementation[1].

4.4.1 *Setup and Datasets*

The goal of our study is to investigate how wavelet transformed data may impact TSC performance. Therefore, we do not intend to compare the performance of different classifiers and similarity metrics (which have been empirically studied in [Serra and Arcos, 2014]). We rely in our study on the most frequently used classification method: Nearest Neighbor Classification (1NN) with DTW distance. In this study we choose to calculate the classification performance first in Section 4.4.2, Section 4.4.3 and Section 4.4.4 using FastDTW [Salvador and Chan, 2007], which is a DTW approximation with linear time complexity, and then extend the experiments using an exact DTW implementation named UCRSuite [Rakthanmanon et al., 2012] in Section 4.4.5.

The datasets that we have experimented on are from the UCR Time Series Classification Archive [Chen et al., 2015], which contains datasets from various domains ranging from electricity readings and medical signals such as Electrocardiographs (ECGs) to image recognition data. We have specifically chosen to use the *Newly Added Datasets*, which contains 39 separate datasets from various domains, with all these datasets sharing a unified file format and internal representation structure, which is convenient for batch processing. The UCR archive provides both the datasets and the ground-truth, i.e., the correct label of each testing instance. Table 3.1 summarizes the characteristics of these 39 datasets. These characteristics include the number of classes in the training and testing sets, how many instances

1 https://github.com/serval-snt-uni-lu/wavelets-tsc

are there in training and testing sets respectively, and finally the lengths (dimensionality) of time series samples. As shown, this archive comes with predefined training and testing sets, which makes it easier for researchers to compare results in an uniformed manner. However, we are not aware of more detailed information about each dataset such as the sampling frequency and original amplitude, making it difficult for us to draw conclusions about domain-specific tasks.

Besides, it is obvious that several datasets are large in size, for instance, to classify all $7,711$ testing instances using 1NN in *ElectricDevices* (#10), there will be $7,711 * 8,926 = 68,828,386$ pairwise comparisons, making the classification process extremely time consuming. Due to the large amount of computation tasks, we have split the test datasets in order to parallelize computation. All the classification tasks are conducted on an HPC platform [Varrette et al., 2014].

4.4.2 *TSC with Wavelet Transformed Data*

As a first step, we seek to investigate how DWT compressed data will impact classification accuracy compared with using raw uncompressed data. Here we transform all the 39 datasets using wavelets from seven well-known families, choosing the single wavelet with the highest regularity from each family. Concretely, the wavelets are: Haar, Daubechies 20, Symlets 20, Coiflets 5, Biorthogonal 6.8, Reverse biorthogonal 6.8 and Discrete Meyer with finite impulse response (FIR) approximation. In this step, all time series – both training instances and testing instances – from each dataset are processed using single level, one dimensional DWT. After transformation, the size of *compressed* data is reduced by half from the original series. Then, we use FastDTW-based 1NN to classify all the testing instances in each dataset. Thanks to the $O(n)$ time complexity of FastDTW, reducing time series sizes by half means reducing classification time by half. And for DTW with $O(n^2)$ complexity, classification time can be reduced by 75%.

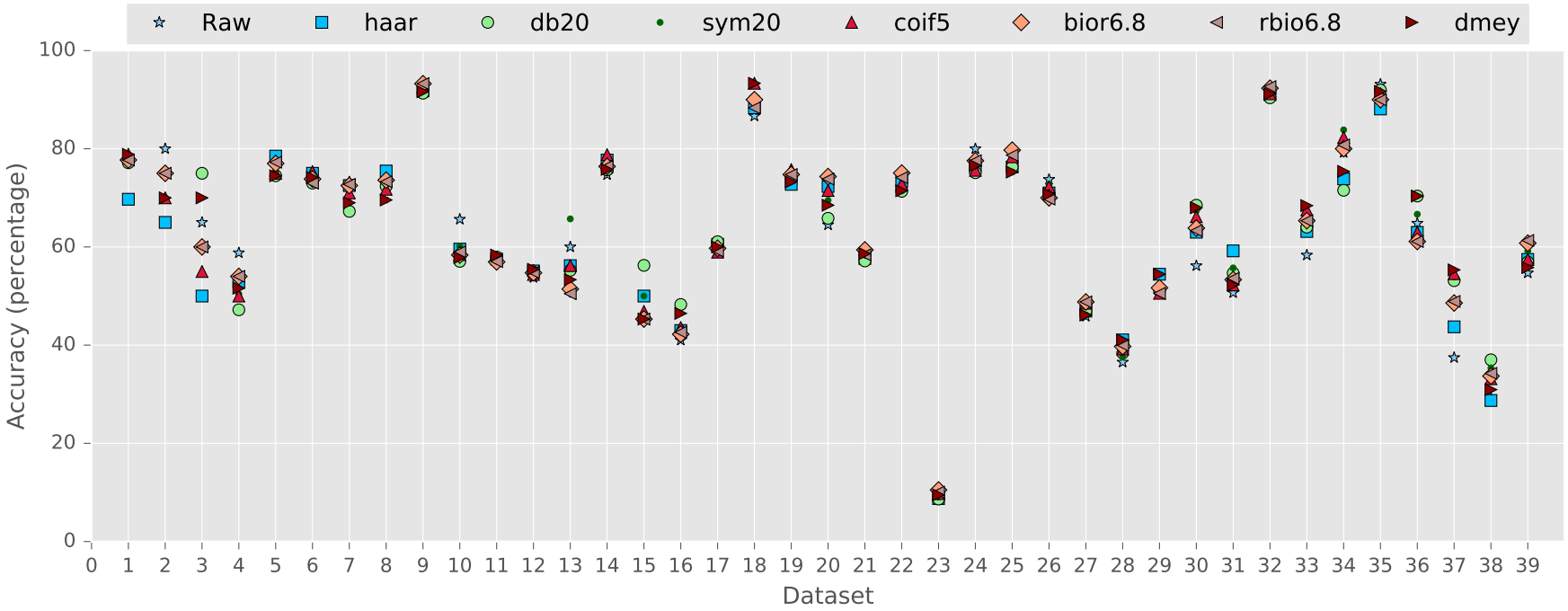

Figure 4.3: Classification accuracy with FastDTW based 1NN, using original and DWT transformed/compressed data.

Figure 4.3 presents the classification accuracy of each wavelet transformed dataset together with that of the original data. We note that, while the compression yields smaller datasets and leads to faster classification, it does not impact the classification accuracy in most cases. Furthermore, in the case of several datasets, the classification accuracy has actually been improved on compressed data. These observations suggest that wavelet transformations are indeed relevant means for noise reduction in TSC tasks.

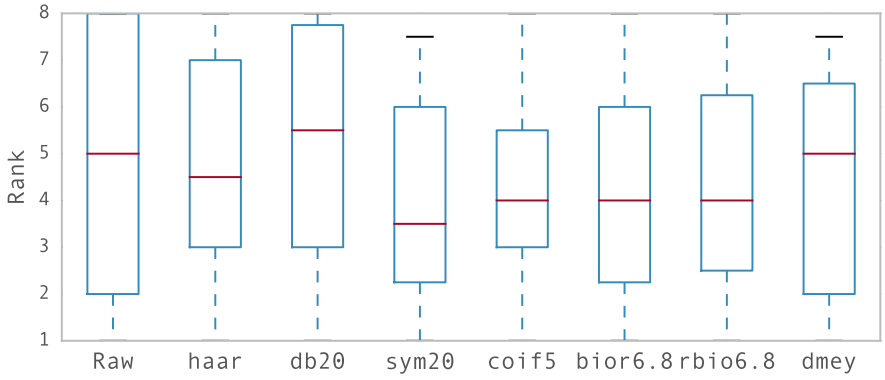

Figure 4.4: Rank of classification accuracy by approximation of different wavelet transformation.

To further investigate which wavelet family performs better globally, we rank each wavelet family's classification performance per dataset and draw a boxplot chart of these rankings. As shown in Figure 4.4, in general wavelet transform data performs better compared with the original data, thanks to wavelets' noise reduction functionality. Regarding the performance of individual wavelets, **Symlets 20** generally outperforms the rest, including classification using original data. And much to our surprise, *Daubechies 20* in general performs the worst among all tested wavelets, indicating that the most smooth wavelet may not be the most suitable wavelet for TSC.

4.4.3 *TSC with Residual Details*

It is intuitive that when using wavelet transformation, the approximation of original data keeps more relevant information than the residual details. However, in the next experiments, we demonstrate that the residual details may be also useful for TSC and that in some scenarios the *noises* are more discriminative features than the approximations. To prove this seemingly counter-intuitive point, we follow the same procedures as Section 4.4.2 to compress all datasets, while instead of keeping the approximations, here we drop all of them and consider the residual details, i.e., the noises. Again, to be fair, we compare the classification accuracies using the noises against that using the original raw data.

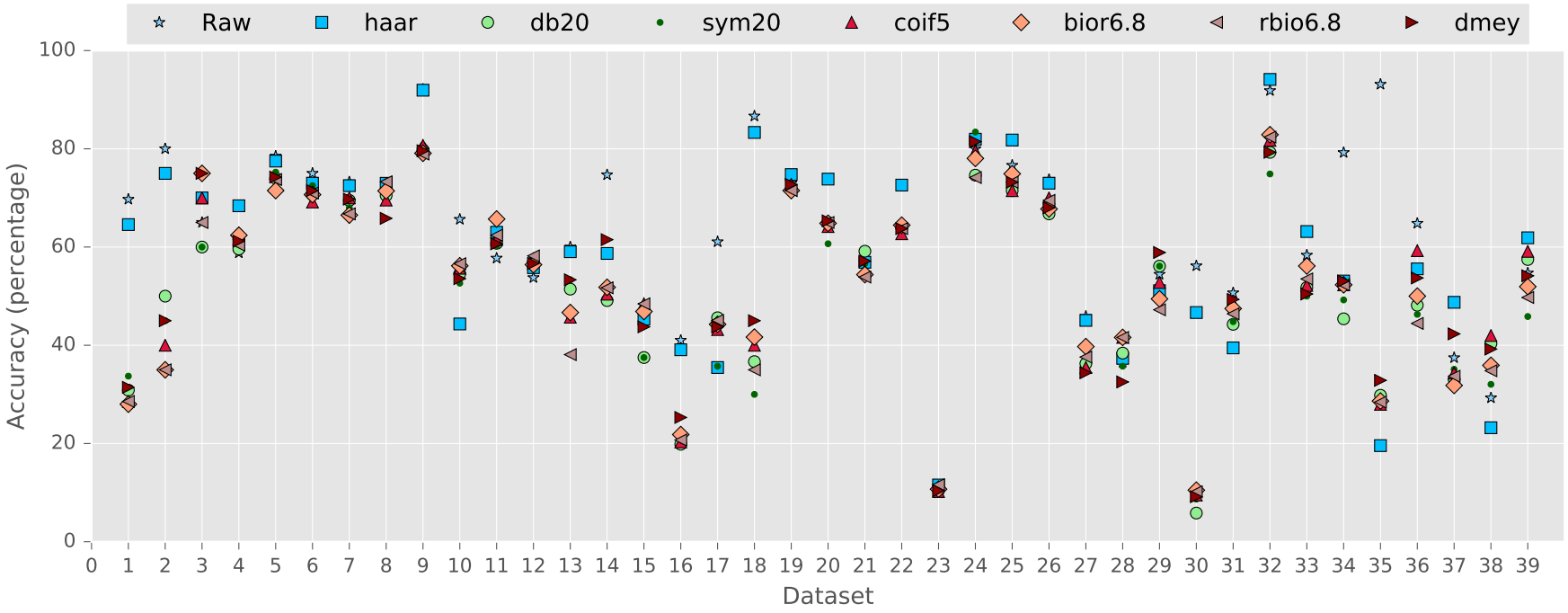

Figure 4.5: Classification accuracy with FastDTW based 1NN, using original data and residual details from DWT transform.

Figure 4.5 presents the classification accuracies using only the residual details. As shown, although the classification accuracies are generally lower when classifying using only the residuals, they are still surprisingly high in many cases. Especially, for several datasets, classification accuracy using only residuals are similar or higher than using the raw data. Next, we try to rank how discriminative the residuals from each wavelet family are. The boxplot shown in Figure 4.6 suggests that residuals produced by *Haar* are most discriminative, being almost as good as the original data. *Symlets 20*, on the other hand, falls on the other extreme. These two observations are in accordance with the finding in Section 4.4.2, suggesting that *Symlets 20* is good at keeping globally relevant information during transformation. This observation suggests that residuals from DWT compression can also be useful, since these details contain local features that are potentially discriminative.

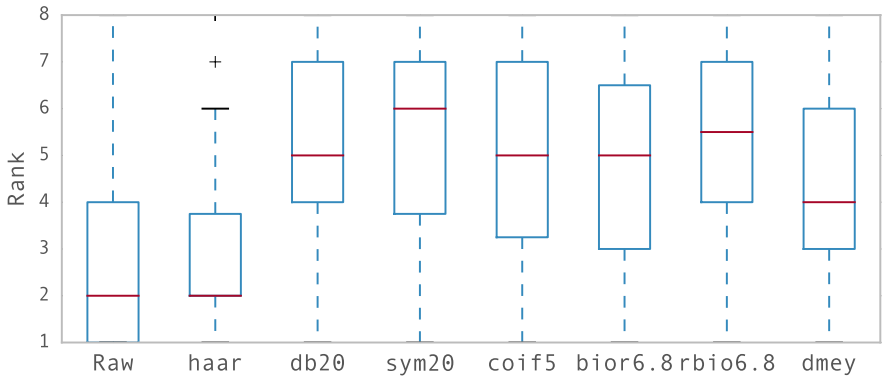

Figure 4.6: Rank of classification accuracy by residual details from different wavelet transformation.

As a result, we believe that both the approximation and details transformed from the original data are important, due to the fact that the transforms extract two independent discriminative features. While some datasets are more distinguishable using global features, others are more so using local defining features.

4.4.4 *Multi-Level Wavelet Transformation*

So far we have only tested the performance of single level wavelet transformation. Since it is possible to conduct wavelet transformation in multiple levels, we seek to investigate how TSC performance can be affected when transforming data through multiple levels. To be specific, based on the good results in previous sections, we transform all datasets with *Symlets 20* from level 2 till the maximum level as permitted by the wavelet transform library[2]. That is, due to different time series lengths, even level 2 transformation is not possible for short time series, while longer time series may be processed using level 6 Symlets 20 decomposition. In Table 4.1 we present both the percentage of dimension/size reduction (**R**)

2 PyWavelets – Discrete Wavelet Transform in Python (http://www.pybytes.com/pywavelets/)

and the corresponding classification accuracy (**A**) in each level. Note that not all datasets support at least level 2 decomposition due to short lengths, as a result, these datasets are omitted in this table.

Table 4.1: Classification accuracy with FastDTW based 1NN, using Symlets 20 multi-level decomposition.

	Raw	**Level 2**		**Level 3**		**Level 4**		**Level 5**		**Level 6**	
#		**R**	**A**	**R**	**A**	**R**	**A**	**R**	**A**	**R**	**A**
1	69.7	63.3	**78.3**								
2	80.0	69.3	65.0	80.9	65.0						
3	65.0	69.3	**70.0**	80.9	**80.0**						
4	58.8	71.0	58.4	82.8	55.2	88.8	50.0				
8	71.7	69.3	**78.6**	80.9	69.9						
11	57.7	69.2	56.5	80.8	**59.7**						
12	53.8	69.2	**55.9**	80.8	**62.0**						
13	60.0	68.2	**65.7**	79.6	53.3						
14	74.7	73.9	**80.0**	86.3	**79.9**	92.4	**79.0**	95.5	71.1	97.0	70.7
15	45.3	69.3	**53.1**	80.9	**50.0**						
16	41.0	63.7	**44.1**								
17	61.1	71.0	**64.0**	82.8	**63.2**	88.8	**64.3**				
18	86.7	68.5	**93.3**	79.9	**93.3**						
23	9.9	72.2	**12.3**	84.2	**12.4**	90.2	**11.0**				
27	45.9	71.0	45.3	82.8	41.9	88.8	39.2				
28	36.5	71.0	**36.8**	82.8	**38.4**	88.8	**40.0**				
29	54.4	69.2	**55.0**	80.8	51.7						
30	56.2	69.3	**72.7**	80.9	**72.2**						
31	50.7	71.0	**61.9**	82.8	**65.9**	88.8	**67.5**				
32	91.8	62.6	91.7								
33	58.3	64.6	**68.0**								
34	79.2	66.5	**81.5**	77.6	77.7						
35	93.1	72.0	92.4	83.9	**93.2**	89.9	91.5				
36	64.8	62.8	55.6								
37	37.5	64.4	**61.3**								
38	29.3	71.8	**35.9**	83.8	**45.3**	89.8	**47.0**				
39	54.7	71.8	**60.2**	83.8	**64.6**	89.8	**65.7**				

As shown in Table 4.1, although many datasets are compressed by 80% to 90% in size, the classification accuracy using these reduced data can still outperform those using original compressed data. As a result, we think it is safe to claim that multi level wavelet transformation are indeed helpful for TSC tasks when it comes to classifying long time series. Note especially the *HandOutlines* (#14) dataset, due to its extremely high dimensionality, we are able to compress them by up to 97% of the original size, while still obtaining remarkably high classification accuracy. Since FastDWT normally has a time complexity of $O(n)$, this indicates huge time savings in the classification process.

4.4.5 *Using the UCR suite for TSC and Significance Test*

Since the distance measure we have used – FastDTW – is approximate [Li et al., 2016c], we extend our evaluation using an exact distance measure named UCR-Suite [Rakthanmanon et al., 2012], in order to get a more accurate view of DWT's performance in terms of noise reduction when it comes to classification. Note that since UCRSuite requires a DTW warping window size to be configured when calculating time series distances, we have tried all possible warping window size specified as a percentage (0%, 1%, 2%, ..., 99%) of lengths of corresponding time series samples and select the best classification accuracy.

Besides using a more accurate distance measure, we also seek to find out if one wavelet family indeed significantly outperforms another. To that end we conduct a Nemenyi test [Nemenyi, 1962], which is a post-hoc test that takes pairwise tests of performance and decides if these groups are statitically similar. Figure 4.7 shows the test result in the form of a critical difference diagram, where average ranks of all examined approaches are presented and bold lines (insignificance lines) indicate groups of approaches which are not significantly different. As shown, using a more accurate DTW distance implementation shows similar results, indeed indicating that wavelets can be very useful for dimensionality reduction in TSC tasks. Especially, we note that performances of *Haar*, *Reverse biorthogonal 6.8*, *Symlets 20*, *Biorthogonal 6.8* and *Coiflets 5* are not significantly different than using uncompressed raw data. Furthermore, it is consistant with our previous experiments that *Daubechies 20* and *Discrete Meyer with finite impulse response (FIR) approximation* do not perform as well as the other wavelets.

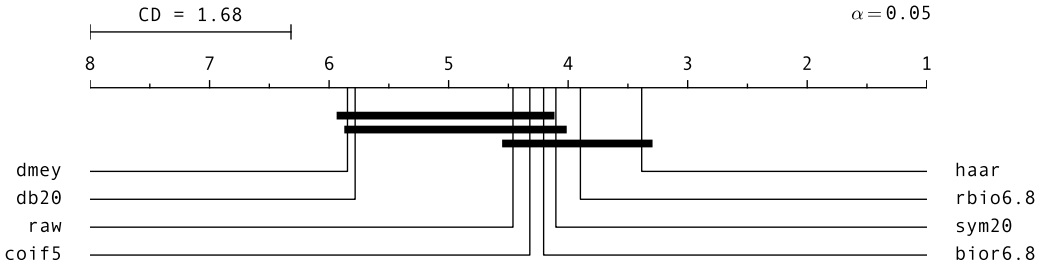

Figure 4.7: Critical difference diagram for classification using raw and transformed data using different families of DWT.

4.5 THE SMOOTHING EFFECT OF WAVELETS

We could see clearly from Figure 4.2 that the Haar Wavelet approximation compression smoothens the overall curve while removing the details. In this section, we try to understand if the good performances of DWT on time series data is due to the implicit smoothing effect. One of the simplest explicit smoothing tech-

niques is probably simple moving average, which is calculated by computing the unweighted mean over a specific number of time periods. Similar to Haar as specified in Eq. 4.5, simple moving average can be defined mathematically in Eq. 4.7:

$$SMA_i = \frac{\sum_{j=1}^{m} t_{m*i+j}}{m} \tag{4.7}$$

where $2 \leqslant m < \frac{n}{m}$ is the number of data points to average on.

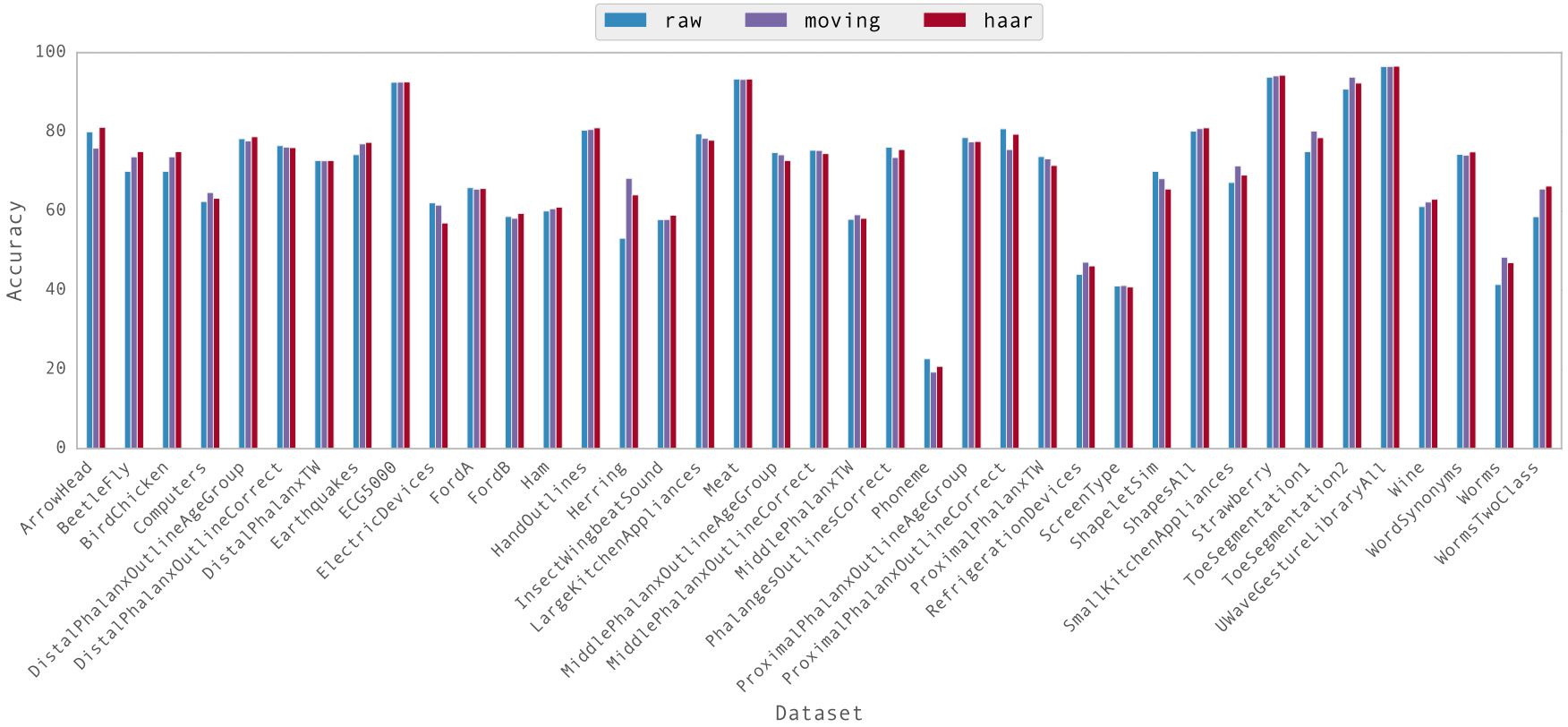

Figure 4.8: Classification accuracy with UCRSuite DTW-based 1NN, using raw, Haar transformed and moving average smoothened data.

Due to their similarity, we first compare the performance of Haar against raw and moving average smoothened data. Figure 4.8 illustrates the classification across 39 datasets used in the previous section. As shown, in some datasets Haar outperforms moving average (e.g., *ArrowHead* and *BeetleFly*), while in some others Haar underperforms (e.g., *ProximalPhalanxTW* and *ShapeletSim*). To answer the question whether wavelet transforms statistically outperform smoothing techniques, we need to run another significance test. To that end we compare DWT with explicit smoothing techniques, including moving average smoothing, local regression with 1st degree polynomial model (*lowess*) and second degree polynomial model (*loess*) as well as their robust versions (*rlowess* and *rloess* respectively), and finally the Savitzky-Golay filter (*sgolay*). These techniques are frequently used in real-world scenarios and are available from MATLAB®. Figure 4.9 illustrates the significance test. We can see that overall wavelets perform better than explicit smoothing techniques. Especially, simple moving average smoothing helps but not in significant terms. Besides, *Haar*, *Reverse biorthogonal 6.8*, *Symlets 20*, *Biorthogonal 6.8* and *Coiflets 5* slightly outperforms moving average, indicating wavelets' superiority in general purpose time series data smoothing and noise reduction.

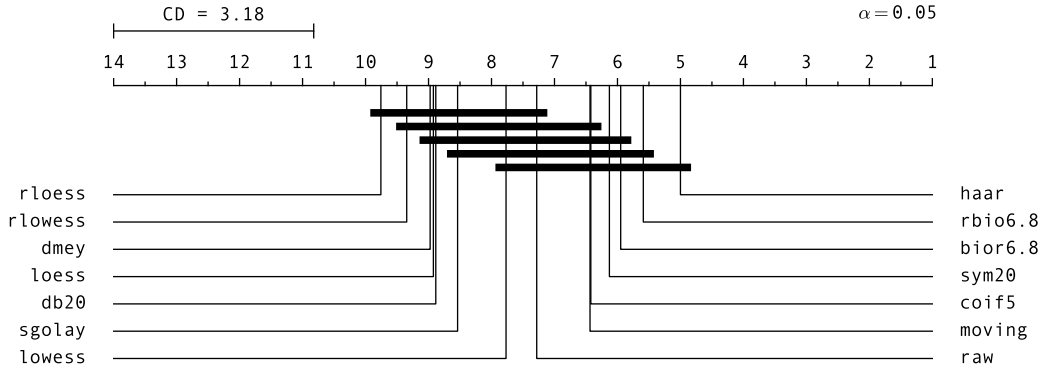

Figure 4.9: Critical difference diagram for classification using raw and transformed data using different families of DWT and explicit smoothing techniques.

4.6 CONCLUSIONS AND FUTURE WORK

Discrete Wavelet Transform techniques have matured in the past decades to deliver high data compression rates. Applied to time series data, existing DWT-based lossy compression approaches help to overcome the challenges of storage and computation time. In this chapter, we provide assurances to practitioners by empirically showing with various datasets and with several DWT approaches that time series classification yields similar accuracy on both compressed (i.e., approximated) and raw time series data. We also show that, in some datasets, wavelets may actually help in reducing noisy variations which deteriorate the performance of mining tasks. In a few cases, we note that the residual details/*noises* from compression are more useful for recognizing data patterns.

In future work, we plan to extensively investigate the characteristics of time series datasets, in order to empirically correlate successful wavelets techniques per application domains. Dataset characterizations will also help identify which types of time series data can benefit from the use of residual details instead of the approximation data.

5

DOMAIN SERIES CORPORA

> For last year's words belong to last year's language.
>
> *T. S. Eliot*
> *Four Quartets*

Outline

We propose a novel TSC approach named Domain Series Corpus (DSCo), which firstly transforms numeric values into texts and then builds per-class language models from these texts. To classify unlabeled samples, we compute the fitness of each symbolized sample against all per-class models and choose the class represented by the model with the best fitness score. In addition, we propose improvements to this approach and bring up DSCo-NG, which can be more efficient and yield more accurate classification results.

5.1 INTRODUCTION

Time series data refers to a sequence of data that is ordered either temporally, spatially or in other defined order. Such data are abundant in various domains including health-care, finance, energy and industry applications. Furthermore, time series data are often characterized as large in size and high in dimensionality [Fu, 2011]. These characteristics of time series data – together with its abundance – has led to various challenges in both storage and analytics. For example, the BLUED non-intrusive load monitoring dataset [Anderson et al., 2012] records voltage and current measurements in a single household for one week with a sampling rate of 12 kHz, leading to a total of tens of billions of numeric readings and making it extremely difficult to mine meaningful patterns in real-time using these raw numeric data. Researchers from EPFL also argue that while smart meter technologies make it possible for utility companies to analyze household energy consumption data in real-time, data acquired by smart meters are often so large that analytic tasks become extremely expensive [Wijaya et al., 2013].

Traditionally, researchers have proposed various methodologies to represent time series more efficiently, including dimensionality reduction [Keogh et al., 2001] and numerosity reduction [Xi et al., 2006] techniques. Another line of research on time series representation focuses on converting numeric values into symbolic

form [Fu, 2011], while one of the most prominent approaches is Symbolic Aggregate approXimation (SAX) [Lin et al., 2007]. Although SAX comes with a distance measure that can be used for nearest neighbor classification, it is still unclear how classification performance will be affected when classifying SAX's symbolic representations of time series.

In this chapter, we set to investigate how symbolic representations of time series can tackle time series classification (TSC) challenges. Specifically, we propose a novel TSC approach named Domain Series Corpus (DSCo, pronounced as `disco`), which firstly transforms numeric values into texts and then builds per-class language models from these texts. To classify unlabeled samples, we compute the fitness of each symbolized sample against all per-class models and choose the class represented by the model with the best fitness score. Our work innovatively takes advantage of mature techniques from both time series mining and Natural Language Processing (NLP) communities. Through extensive experiments on an open dataset archive, we demonstrate that our approach not only performs similarly or better than state-of-the-art approaches (which works with original data that possesses much more information), but can also work on reduced data, an essential property to ensure scalability in TSC.

Overall, the contributions of this chapter are summarized as follows:

- We bring up a novel method for TSC by leveraging mature techniques from the NLP community. By taking advantage of language modeling techniques we are able to consider both *local* and *global* similarities among time series.
- We have tested our approach extensively on an open archive which contains datasets from various domains, demonstrating by comparison with state-of-the-art approaches that DSCo is performant, efficient and can be generalized.
- We prove that although our approach works with approximated data, DSCo can perform similarly to approaches that work with original uncompressed numeric data.
- We propose a new perspective for TSC: we view time series data as *sentences* (as in natural languages), where some *words* and their *combinations* will define different classes. In this way, we approximate a TSC task to a pseudo language detection problem.

The remainder of this chapter is organized as follows. Section 5.2 provides our intuition and the necessary background information on time series classification as well as preliminaries on language modeling. Section 5.3 presents the details of our approach, while experiments and evaluation results compared with related work are described in Section 5.4. We further propose DSCo-NG to improve the performance of DSCo in Section 5.5 and evaluate its performance in Section 5.6. Section 5.7 briefly surveys related research work to ours. Section 5.8 concludes the chapter with directions for future work.

5.2 BACKGROUND AND KEY INTUITION

In this section, we briefly introduce the mechanism behind SAX and how SAX is traditionally used for TSC tasks. Then we present language modeling and how it can be used in combination with *SAX-ified* strings. In the meanwhile, we introduce our intuitions on tackling TSC with symbolic representation and language modeling.

5.2.1 *Language Modeling*

Given a string representation of time series data, we can apply language modeling to assess whether it fits the model of a class. Language models are used to answer questions such as *"How likely a string of words from a language vocabulary is good phrasing in this language?"*. A statistical language model is a probability distribution over strings of a corpus [Ponte and Croft, 1998]. Thus, any sequence of words W has a probability score $P(W) = P(w_1, ..., w_n)$ in the language model, indicative of its relative validity within a language.

N-gram language models are common means of language modeling. In the simplest case of *unigram* models (1-gram models), the probability score of the sequence of words W is approximated to the product of the probabilities of each word. Equation 5.1 provides the formula for computing this score.

$$P(W) = P(w_1, ..., w_n) \approx \prod_{i=1}^{n} P(w_i) \tag{5.1}$$

Bigram models put conditions on the previous word to account for the likelihood of co-occurrence between two words (for instance, *beer drinkers* appears more often than *beer eaters*). The probability score of the sequence W is then approximately the product of conditional probabilities of words with their previous peer. It is computed by the formula in Equation 5.2.

$$P(W) = P(w_1, ..., w_n) \approx \prod_{i=1}^{n} P(w_i|w_{i-1}) \tag{5.2}$$

Since in a given corpus it is possible that certain bigrams are never observed beforehand, it is reasonable to use a back-off mechanism to take into account only their unigram probabilities. Although N-grams models can be theoretically insufficient because language has long-distance dependencies (for instance, some words may co-occur in a sentence but not directly following each other in the sequence: *"the computer which I just bought and setup in my room crashed"*), these models have been shown to be efficient in practice and used in various fields including speech recognition, author attribution and malware detection.

5.3 DOMAIN SERIES CORPORA FOR TSC

Since symbolic representation of time series data is a promising mechanism to tackle the numeriosity and high dimensionality issue in the era of big data, we investigate how symbolic representation and language modeling can be used for TSC. Recall that DSCo builds on the simple intuition that time series patterns in a specific class of a given domain can be differentiated from other class patterns, which is similar to NLP methods that distinguish texts from different languages or dialects. The assumption is thus that the language model extracted from the samples of a specific class will be descriptive and discriminative enough to differentiate it from another language model within the same domain. DSCo therefore consists of building a corpus of *words* representative of time series subsequences (or *segments*) for a given domain and the associated language models for its classes.

Figure 5.1 illustrates the steps for building per-class Domain Series Corpora for a specific domain. First of all, data readings of time series samples from each class are transformed into texts. Next, language modeling is applied on these texts to extract the corresponding language models, so that afterwards these models can be used to test and classify unlabeled samples.

5.3.1 *Data Representation as Texts*

As described in Section 5.2, we create symbolic representations for real-valued samples. In DSCo we have leveraged SAX for this task. It is nonetheless possible to leverage another symbolic representation algorithms for time series. The output of this step is a string representation for each time series sample.

5.3.2 *Language Model Inference*

DSCo explores a training set of time series to extract meaningful patterns of segments by studying their occurrence frequencies. Once time series are represented as texts, a language model can be built to summarize each time series class. To build a language model for a time series class, DSCo generates its corresponding dictionary by collecting *words* that appear in the training set. A large body of work have been proposed in the NLP literature on how to obtain such dictionaries.However, since the symbolic representations generated by SAX have no word boundaries, we need to break them into smaller pieces first by employing a corpus acquisition mechanism. This is common procedure for some natural languages such as Chinese, which has no obvious word boundaries.

One approach to dictionary acquisition is to break the sequences using an annealing algorithm, which is a probabilistic and non-deterministic algorithm that randomly permutes the possible segmentations and searches the whole solution space with the best segmentation until thresholds are met according to an ob-

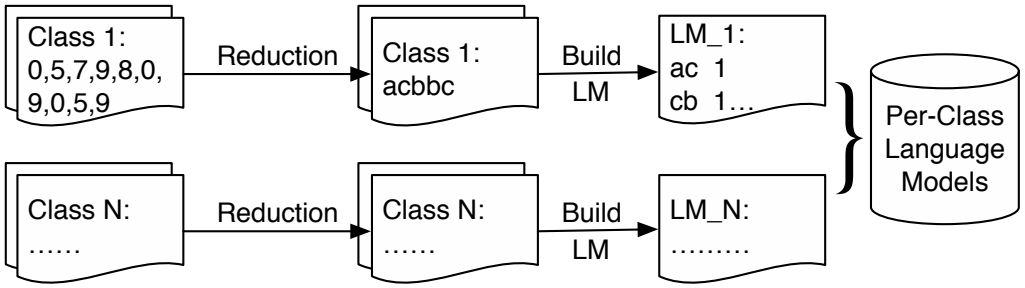

Figure 5.1: Process for building language models in DSCo.

jective function. Such an approach is able to find word boundaries with reasonable accuracy given a large training set. Unfortunately, this algorithm is highly time-consuming. Besides, time series training sets are seldom sufficiently large to accommodate this algorithm. As a result, we extract *words* from the symbolic representations using a naïve sliding window method described in Algorithm 5.1. This algorithm collects all possible sub-strings of length w within a string, so that no descriptive segment is left uncaptured from the original time series. For example, we can break string `abccc` into the following 2-alphabet segments: `ab`, `bc`, `cc` and `cc`.

Algorithm 5.1 Extract *words* from a string (S) using a sliding window (of length l).

```
1: procedure ExtractWords(S, l)
2:     words ← ∅
3:     for i ← 0, GetLength(S) − l + 1 do
4:         word ← SubString(S, i, l)                ▷ Sub-string of size l
5:         words ← words ∪ {word}
6:     return words
```

Next, in order to better preserve the descriptive information that time series generally come together in a sequence, we also compute the frequencies of n-grams. We build n-gram language models for each time series class in our training set. This process is illustrated in Algorithm 5.2. In order to be generic, we define a minimum (`minWL`) word length and a maximum word length (`maxWL`): The intuition behind this is that 2-alphabet segments may be generic but not descriptive, while segments with larger length may be descriptive but not generic enough. These extreme values can be easily determined with enough domain knowledge. For instance, if we assume that electrocardiogram (ECG) patterns generally have similar lengths, the `minWL` and `maxWL` values can be predefined to avoid producing noisy segments.

When all n-grams in every per-class language model are counted, we convert their frequencies into probabilities within each language model. Note that frequencies may need normalization when there are different number of instances in each class. Otherwise n-gram probabilities will be biased and lead to classification errors in the next step.

Algorithm 5.2 Build language models (*LMs*) from a list (SL) of $(string, label)$ pairs.

```
1: procedure BuildLM(SL, minWL, maxWL)
2:     LMs ← ∅
3:     for all (string, label) ∈ SL do
4:         if NGrams_label ∉ LMs then
5:             NGrams_label ← ∅
6:         for wl ← minWL, maxWL do
7:             words ← ExtractWords(string, wl))
8:             for all ngram ∈ GetNGrams(words) do
9:                 InsertOrIncreaseFreq(NGrams_label, ngram)
10:        LMs ← LMs ∪ NGrams_label
11:    ConvertFreqToProbability(LMs)
12:    return LMs
```

5.3.3 *Classification*

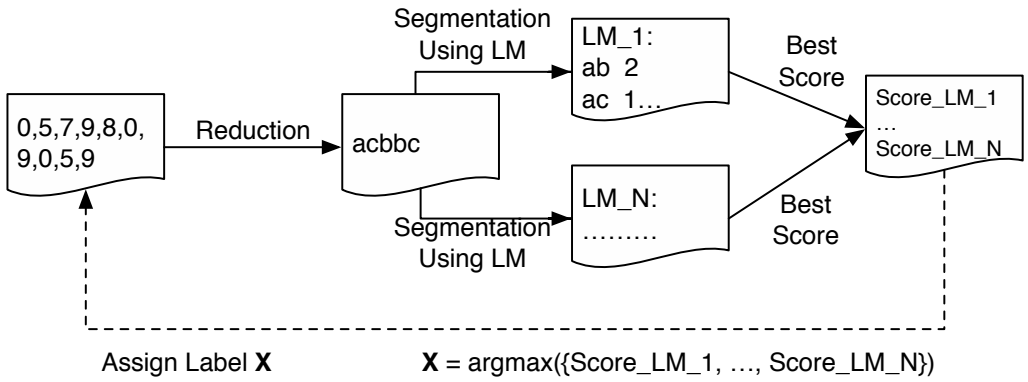

Figure 5.2: Illustration of DSCo's classification process.

In DSCo, classification is performed by checking which language model is the best fit for the tested sample, as shown in Figure 5.2. First, similar to the training phase, each test sample is reduced to a string using the same algorithms and parameters. Then, in order to test model fitness, each language model – which summarizes the characteristics of all samples from a given class – is used to segment the time series' text representation. To be computationally efficient, we consider a probabilistic language modeling approach with bigrams as introduced in Section 5.2. We compute the segmentation score as the product of conditional probability of all segmented words following Equation 5.2. If a bigram is not known in the language model, we just back-off to the unigram probability values for this specific segment. Since there are different ways to segment the text according to a specific language model, we only consider the best segmentation score that can be obtained with each language model. That is, each language model segments the sample and keeps its best segmentation score, then all language models compare

their scores and the winning language model's class label will be applied to the sample.

Algorithm 5.3 Given language models, find the best way (with the maximum probability) to segment a string (S).

Require: global $minWL$ and $maxWL$

```
1: procedure SplitString(S)
2:     P ← ∅
3:     sl ← GetLength(S)
4:     for l ← minWL, min(sl, maxWL) do
5:         if sl − l ⩾ minWL then
6:             P ← P ∪ {(Slice(S, 0, l), Slice(S, l, sl))}
7:     return P
8: procedure Segment(S, prev)
9:     UpdateViterbiTable()          ▷ Dynamic programming to avoid repetitive computing
10:    if GetLength(S) < minWL then
11:        return 0
12:    Sgs ← ∅
13:    for all (h, t) ∈ SplitString(S) do
14:        Sgs ← Sgs ∪ {P(S|prev) ∗ Segment(t, h)}
15:    return max(Sgs)
```

5.4 EVALUATION

In order to evaluate the feasibility and performance of our approach, we have implemented DSCo and tested on an open dataset archive. We first reduce time series data using SAX and show that SAX distance-based 1NN under-performs DTW. Then we investigate the value added by DSCo on top of SAX. Finally, through comparison with Euclidean- and DTW-based 1NN classification, we show that DSCo is indeed performant. We have open sourced our implementation[1] in order to increase reproducibility.

To explore the performance of the DSCo approach and investigate the extent of its applicability, we consider the Time Series Classification Archive [Chen et al., 2015] from University of California, Riverside. The datasets contained in this archive are popular within the TSC community, allowing for a reliable comparison baseline. Besides, the archive includes error rates for DTW- and Euclidean-based nearest neighbor classification as a performance benchmark for TSC. The UCR archive is composed of two sub-archives: `Pre_Summer_2015_Datasets` and `Newly Added Datasets` which include datasets from various fields, ranging from electrocardiograms (ECG) to intra-species image recognition data. We tested on the latter (which contains 39 different datasets) because its file format and internal

1 https://github.com/serval-snt-uni-lu/dsco

data structures are consistent, making it possible to conduct batch processing in a content-agnostic manner. Furthermore, both sub-archives have similar dataset diversity and some datasets for specific domains (for example, ECG data) appear in both sub-archives. Specifically, these 39 datasets have various number of classes from 2 to 60 and different number of training and testing instances from 20 to 8,926, with time series lengths varying from 80 to 2,079.

5.4.1 *Reducing Data using SAX*

In order to validate SAX's data reduction performance, we take those `Newly Added Datasets` from UCR and transform the numeric data records into symbols. We have set the maximum time series length to 100 and varied SAX's alphabet size from 3 to 20, so that we can take advantage of SAX's dimensionality and numeriosity reduction mechanism. Since SAX can be viewed as a lossy compression function, we evaluate how well the original information are kept for TSC using 1NN classification and compare the classification performance between DTW distance and SAX's internal distance measure as defined in [Lin et al., 2007]. SAX's distance measure is essentially a variance of Euclidean distance except that the distance between two alphabets are predefined in SAX's look-up table. For instance, for an alphabet size of 4, $\mathtt{dist}(a, b) = 0$ and $\mathtt{dist}(a, c) = 0.67$. Figure 5.3 presents the 1NN classification comparison, where the solid lines indicate SAX distance-based 1NN classification accuracy for each of the 39 datasets, and the dashed lines shows DTW distance-based 1NN classification performance.

Here we show the classification accuracy of DTW-based 1NN because it is the most mature and widely used distance measure among the research community. Furthermore, classification performance using DTW is readily available from the UCR archive. Finally, using DTW requires one to explicitly setting its warping size parameter. In Figure 5.3 we show DTW's performance with the best warping size (parameter space is 100: warping size varies from 0 to 100 percent of original time series length) that is found in [Chen et al., 2015]. When comparing SAX distance to DTW's best performance, we believe it is fair to present SAX's best accuracy results as well (parameter space is 18: alphabet size varies from 3 to 20). In this case DTW-based 1NN outperforms SAX in $69.2\%(27/39)$ datasets indicating that SAX distance-based 1NN classification indeed under-performs DTW, possibly due to two major facts, namely SAX's lossy reduction of the original numeric data and SAX distance's inability to consider time series' global similarities. As a result, we take advantage of DSCo and take into account both global and local similarities of time series to counter SAX's lossy reduction. And as we shall demonstrate later, DSCo indeed classifies time series more accurately than SAX distance-based 1NN classification.

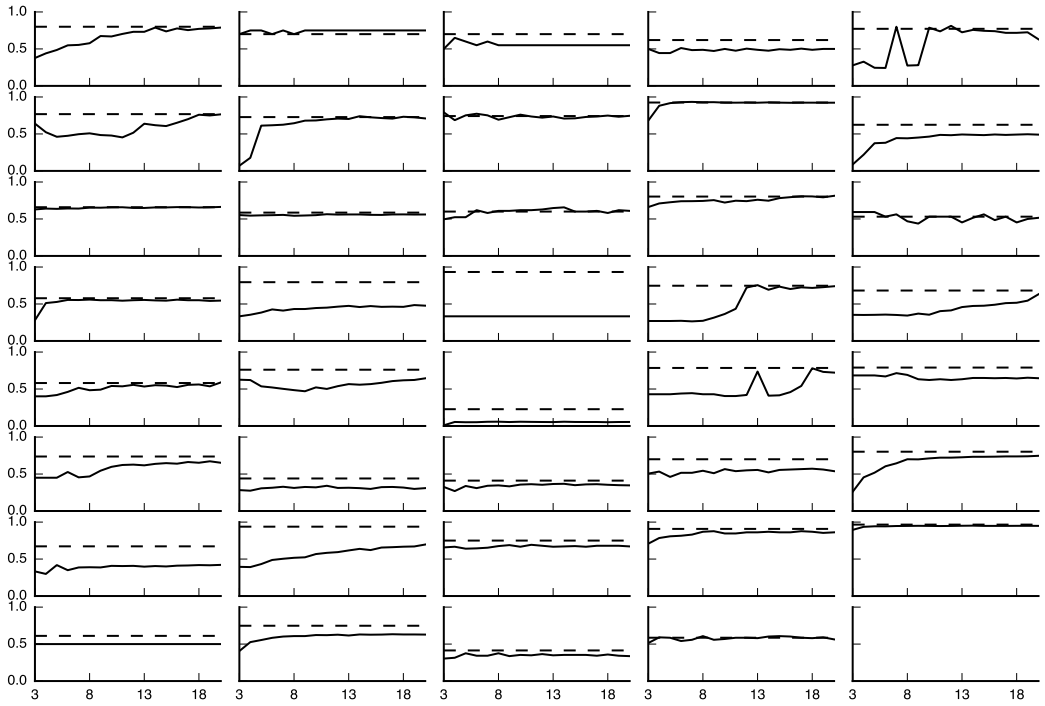

Figure 5.3: 1NN classification accuracy comparison between DTW (dashed) and SAX (solid) distance.

5.4.2 *Implementation and Setup*

Normally, DSCo's classification process can be extremely expensive due to the need of recursively dividing strings in to smaller pieces and segmenting these sub-strings. However, a good implementation may take advantage of the Viterbi algorithm [Viterbi, 1967] – which is in essence a dynamic programming approach – to avoid redundant computation. In addition, the classification process calculates best segmentation scores based on the text segmentation algorithm provided in [Norvig, 2009], which works exceptionally well for NLP text segmentation tasks and has a computational complexity of $O(nL^2)$, where n is the length of a testing string, and L is the maximum word length. Finally, DSCo computes segmentation scores using mainly bigram probabilities, only falling back to unigram probabilities when a specific bigram is not found.

Recall that time series longer than 100 have been arbitrarily reduced to 100-alphabet strings during dimensionality reduction, in order to speed up the classification process; and we have varied SAX's alphabet size from 3 to 20 to search for the best text representation. Since DSCo requires two parameters: a $(minWL, maxWL)$ tuple, we experiment with three sets of parameter settings: short segments $(minWL = 2, maxWL = 10)$, long segments $(minWL = 11, maxWL = 20)$ and short-long combined $(minWL = 2, maxWL = 20)$. Results illustrated in Figure 5.4 show that DSCo's performance are relatively consistent

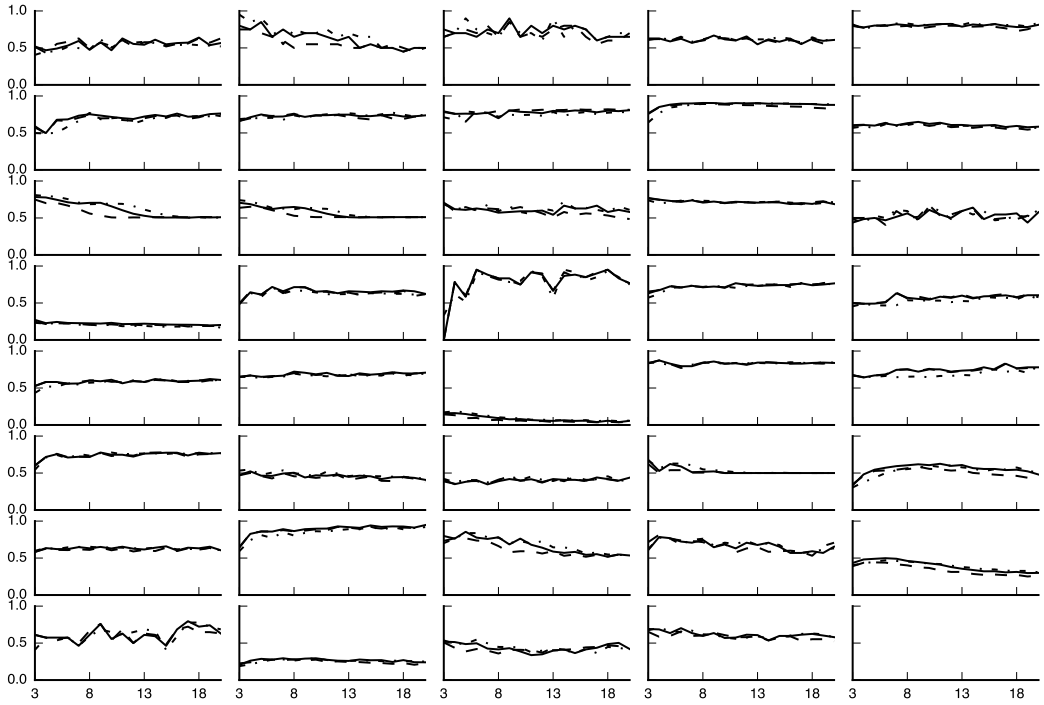

Figure 5.4: DSCo's classification accuracy with different parameter settings: short segments (dashed-dotted lines), long segments (dashed lines) and combined (solid lines).

regardless short or long segments (*words*) are configured. As a result, in the following comparisons, we fix $minWL$ to 2 and $maxWL$ to 20.

5.4.3 *Comparison of Classification Performance*

In the first round of experiments, we investigate the value added by the language modeling to the use of the SAX representations. Since SAX already comes with a distance metric which was demonstrated to yield better performance than the Euclidean distance on real-valued time series data [Lin et al., 2007], we compare DSCo against SAX-distance-based 1NN classification. Table 5.1 shows respectively the classification accuracy and alphabet size when each classification approach performs best. DSCo outperforms in *74% (29/39)* datasets. The results further shows no explicit correlation between DSCo and SAX-distance-based 1NN classification performances, both obtained mostly with unrelated alphabet sizes. Since both DSCo and SAX-distance-based 1NN leverage SAX, these results thus suggest that DSCo's performance cannot be directly attributed to the usage of SAX representation, but rather to the language modeling process.

In the second round, we compare the classification results of DSCo against the benchmark 1NN with Euclidean distance. In *74% (29/39)* of the datasets, DSCo performs better in terms of classification accuracy. We also compare the improvement brought by DTW – the state-of-the-art approach – over Euclidean distance: DTW-

Table 5.1: Classification accuracy comparison between the best performance of DSCo and 1NN with SAX distance, where $|\alpha|$ is the alphabet size when best performance is achieved.

Data-	SAX's Best		DSCo's Best		Data-	SAX's Best		DSCo's Best		Data-	SAX's Best		DSCo's Best	
set	$\|\alpha\|$	acc.	$\|\alpha\|$	acc.	set	$\|\alpha\|$	acc.	$\|\alpha\|$	acc.	set	$\|\alpha\|$	acc.	$\|\alpha\|$	acc.
1	14	**0.79**	11	0.62	14	20	**0.81**	3	0.77	27	11	0.34	4	**0.53**
2	4	0.75	5	**0.95**	15	3	0.59	14	**0.64**	28	14	0.37	17	**0.45**
3	4	0.65	9	**0.90**	16	8	**0.56**	3	0.27	29	18	0.57	3	**0.68**
4	6	0.51	9	**0.67**	17	19	0.49	6	**0.72**	30	20	**0.75**	11	0.64
5	12	0.81	12	**0.83**	18	3	0.33	6	**0.95**	31	20	0.42	19	**0.66**
6	20	**0.77**	20	**0.77**	19	13	0.76	17	**0.77**	32	20	0.70	15	**0.94**
7	14	0.74	20	**0.76**	20	20	**0.63**	7	**0.63**	33	11	0.69	5	**0.85**
8	3	**0.79**	3	0.78	21	20	0.59	20	**0.62**	34	9	**0.88**	5	0.78
9	7	**0.93**	11	0.91	22	20	0.64	8	**0.72**	35	14	**0.95**	6	0.50
10	19	0.50	9	**0.65**	23	8	0.06	4	**0.18**	36	3	0.50	17	**0.80**
11	20	0.66	3	**0.78**	24	18	0.78	4	**0.87**	37	17	**0.63**	7	0.30
12	11	0.56	3	**0.71**	25	7	0.71	17	**0.83**	38	5	0.38	6	**0.54**
13	14	0.66	3	**0.70**	26	19	0.67	18	**0.78**	39	8	0.61	6	**0.71**

based 1NN beats 1NN with Euclidean distance in *64% (25/39)* datasets. We further compare directly DSCo with DTW-based 1NN classification. Figure 5.5 illustrates the results where we consider the best performance with DTW (i.e., with the best warping window size) and the best performance of DSCo (i.e., with the best SAX alphabet size). As shown, DSCo performs similarly to DTW in most datasets. DSCo appears to have good performance in image recognition tasks (for example, `BeetleFly` and `BirdChicken`), while it performs badly for some datasets where the training set has unbalanced distribution of different classes (e.g., `WordSynonyms`), making it difficult for DSCo to extract discriminatory n-grams. Overall, DSCo performs better than DTW in *64% (25/39)* datasets, indicating good classification performance.

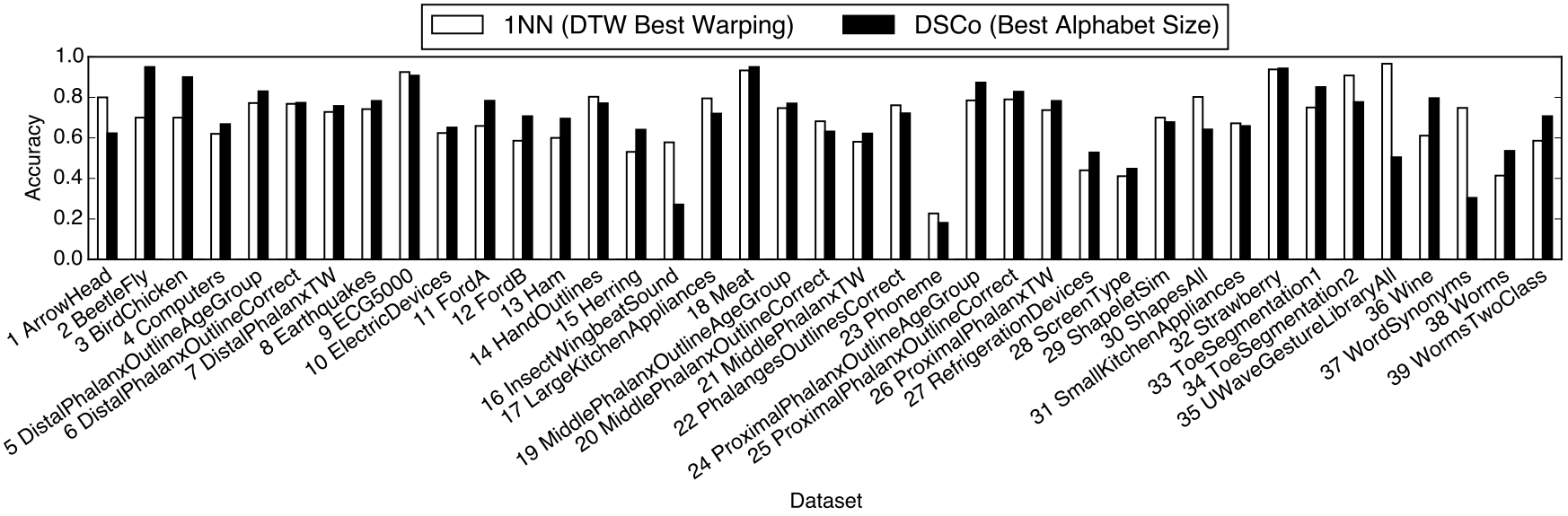

Figure 5.5: Overall accuracy comparison between 1NN with DTW distance and DSCo.

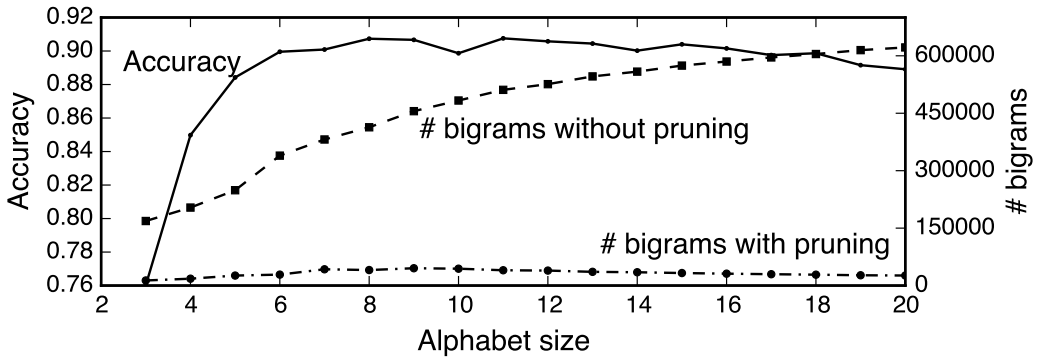

Figure 5.6: For the ECG5000 dataset, classification accuracy remains the same after pruning up to 95.8% bigrams.

Finally, we investigate if pruning bigrams affect classification accuracy. Figure 5.6 illustrates the case with the ECG5000 dataset, where we removed all bigrams that has frequencies lower than the mean value from each language model and use the pruned language models for classification. In some cases the pruned language models has less than 5% bigrams from the original ones. Yet surprisingly, the classification accuracy remains exactly the same. This indicates high redundancy in the language models and pruning techniques can be used in order to reduce memory footprints and speed up the classification process.

5.4.4 *Time and Space Complexity*

SAX has a linear time and space complexity when transforming real-valued time series into PAA and then to text representation. DSCo's language model inference step processes each training sample constant times and stores the models to external storage, resulting an $O(n)$ time and space complexity. We have shown that the fitness metric algorithm for classification in DSCo has a computational complexity of $O(nL^2)$ (in our experiments, $n \leqslant 100$ and $L = 20$). In comparison, the state-of-the-art DTW-based metric has a computational complexity of $O(n^2)$, although with LB_Keogh lower bounding technique this complexity can be reduced to $O(n)$ [Smith and Craven, 2008]. However, for complete classification processes, where a tested sample is compared against all samples from the training set, the 1NN approach using DTW with LB_Keogh yields a complexity of of $O(mn)$ where m is the size of training set. DSCo, on the other hand, has a complexity of $O(cnL^2)$ where c is the number of classes ($c \ll m$ for most scenarios).

Next, we compute the space complexity of DSCo . Given a time series sample T_i of length n, Algorithm 5.1 produces maximum $n - l + 1$ unique *words* of length l. We denote T_i's l-sized words as $W_{i,l}$, then

$$1 \leqslant |W_{i,l}| \leqslant n - l + 1 \tag{5.3}$$

We denote the set of *unigrams* from a specific class of time series consisting of m instances ($T = \{T_1, ..., T_m\}$) as W, then

$$m \leqslant |W| \leqslant m \sum_{l=minWL}^{maxWL} (n - l + 1) \tag{5.4}$$

Since we use *bigrams* in our DSCo implementation, it thus has a space complexity of $O(m^2n^2)$, equivalent to that of shapelet-based algorithms, which also have a time complexity of $O(m^2n^2)$ for their full search procedures [Hills et al., 2014].

5.4.5 *Limitations*

We have demonstrated DSCo's performance through extensive experiments and complexity analysis. However, it also has its own limitations. Since DSCo essentially summarizes training samples into models, it performs best when there are a sufficient number of training samples in each class. Furthermore, our current implementation is built on top of SAX, tweaking SAX's parameters – for example, alphabet size – may require users to look inside data samples in order to achieve best performance. Nevertheless, we believe DSCo has offered a new perspective for TSC tasks and its limitations can be tackled in the near future by employing more advanced symbolization and corpus acquisition algorithms. Another issue with DSCo lies in its excessive memory usage when calculating the fitness score of one sample against language models, which makes it impractical for real-world applications.

5.5 IMPROVING DSCO

In this section, we set to improve DSCo's time and space complexity and propose a next generation of DSCo: DSCo-NG . We follow our initial intuition that time series data are similar to *sentences from different languages or dialects*, but apply a more efficient approach to find nuances of difference from these *languages*. Specifically, unlike in DSCo where we try to find the best way to recursively segment time series, DSCo-NG breaks time series into smaller segments of the same size, and this simplification of the classification process also leads to simplified language model inference in the training phase.

The main complexity of original DSCo lies in the classification process, where testing instances are recursively segmented in order to produce the best segmentation result using a language model, in DSCo-NG we try to break the testing instances into sub-sequences of the same length. Then we calculate the product of bigram probability of these sub-sequences. This scheme is inspired by the intuition that when using a sliding widow of size w to iterate over the training set, all possible unigrams and bigrams are already captured within the language model

of a specific class. As a result, there is no need to use a sliding window of variable length during the classification process, thus reducing the classification complexity. To better illustrate how DSCo-NG works, we detail it in three steps in the subsections below.

5.5.1 *Compressing Time Series into Texts*

There are potentially many approaches that can compress time series data into texts. In DSCo-NG we still employ the same technique – SAX – to transform real-valued time series into texts, in order to investigate whether our proposed improvements still work.

5.5.2 *Extracting Language Models*

Once time series are compressed to texts, a language model can be extracted to summarize each time series class. Since the text representation does not have word boundaries, we need to create artificial words. To that end, we employ a sliding window mechanism that generates such words as specified in Algorithm 5.1. This algorithm collects all possible sub-strings of length w within a string, so that no descriptive segment is left uncaptured from the original time series. For example, we can break string `abcde` into the following 2-alphabet words: *ExtractWords(abcde, 2)* produces an output of $[\mathtt{ab}, \mathtt{bc}, \mathtt{cd}, \mathtt{de}]$.

Next, we build ngram language models for each time series class in our training set, which is illustrated in Algorithm 5.4. Unlike DSCo that requires a minimum word length and a maximum word length to capture words, here we use a single length w. Note that the probability of ngrams are calculated independently, since different classes may have different number of training instances.

Algorithm 5.4 Build language models (*LMs*) from a list (SL) of $(string, label)$ pairs.

```
1:  procedure BuildLM(SL, w)
2:      LMs ← ∅
3:      for all (string, label) ∈ SL do
4:          if NGrams_label ∉ LMs then
5:              NGrams_label ← ∅
6:          words ← ExtractWords(string, w))
7:          for all ngram ∈ GetNGrams(words) do
8:              InsertOrIncreaseFreq(NGrams_label, ngram)
9:          LMs ← LMs ∪ NGrams_label
10:     ConvertFreqToProbability(LMs)
11:     return LMs
```

5.5.3 *Classifying Unlabeled Instances*

As mentioned earlier, classification in DSCo-NG is performed by checking which language model is the best fit for the tested sample. Specifically, we compare the sample's fitness scores to each model, which is calculated following the ngram statistical language model probability.

In practice, bigrams ($n = 2$) and trigrams ($n = 3$) are most prominent [Bellegarda, 2004]. We have opted for the bigram model due to its simplicity for both the language model extraction process and fitness score calculation, which is approximated as $P(w_1, ..., w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-1})$. During the classification process, we need to break time series strings into words. Unlike original DSCo which breaks *sentences* into variable sized *words*, here we adopt the same sliding window size w as the uniform word length. As we shall show later, this simplified process yields similar classification accuracy but greatly reduces the complexity compared with DSCo.

5.5.4 *Time and Space Complexity*

During the preprocessing phase, SAX has a linear time and space complexity when transforming real-valued time series into text representation. When extracting language models in the training phase, each training sample is went through once and models are stored to external storage, resulting an $O(n)$ time and space complexity. Finally, the classification process go through testing samples constant times with language models loaded from external storage, yielding linear time complexity. Language models loaded to memory has a theoretic complexity of $O(\alpha^w)$ where α is the alphabet size used when using SAX to compress real-valued data, and w is the length of *artificial words*. In practice, language models seldom exceed a few megabytes, due to the fact that time series in a domain have a very limited number of *words*.

DSCo-NG's real advantage comes when the training set is large. Given a training set of m_1 time series of length n, when classifying a testing set of m_2 instances, traditional kNN approaches have to conduct $m_1 \times m_2$ pairwise comparisons. Even when using a linear similarity measure such as Euclidean distance, the overall time complexity goes up to $O(m_1 m_2 n)$. On the other hand, DSCo-NG would only have a computational complexity of $O(c m_2 n)$ where c is the number of classes and $c \ll m_1$, making DSCo-NG a magnitude faster than kNN. And this is indeed great improvement even compared with DSCo which has a time complexity of $O(c m_2 n w^2)$ and space complexity of $O(m_1^2 n^2)$.

5.6 EXPERIMENTAL EVALUATION OF DSCO-NG

In order to evaluate performance of our new approach, we have implemented DSCo-NG and tested it on an open dataset archive. To facilitate reproducibility, we have open sourced our implementation with full documentation and tutorials on GitHub[2]. We opt for testing with the UCR Time Series Classification Archive [Chen et al., 2015] for three reasons: 1) this archive has a large number of publicly accessible datasets; 2) these datasets are from a wide range of domains, from environmental monitoring to medical diagnosis; 3) it comes with precomputed classification accuracy rates for DTW-based 1NN, which is the most widely used similarity measure in the research community and has become the *de facto* state-of-the-art benchmark for TSC. In the experiments below we consider 39 datasets from the *Newly Added Datasets* sub-archive because of its uniform file format and structure.

5.6.1 *Implementation and Setup*

In theory, when calculating ngram probabilities, the larger n is, the more accurate these probabilities will be. However, in practice it is seldom the case, due to the lack of training data and the rise of complexity when n becomes larger. As a result, our implementation considers the bigram model with unigram fallback as a trade-off between efficiency and accuracy. Note that falling back to unigrams may not always work, when specific unigrams are missing from the training set. In this case it is necessary to employ a penalty mechanism to offset the influence of such unigrams. From our experience, these missing unigrams' probability could be set as a constant of low probability value, so that the missing probabilities do not overwhelm the existing ones and lead to inaccurate classification.

5.6.2 *Parameter Optimization*

Ideally, time series mining approaches should have as few parameters as possible, even parameter-free, so as to avoid presumption on data [Keogh et al., 2004]. In reality it is extremely difficult to achieve. For instance, even the popular DTW distance requires a warping window size to be set in order to produce optimal results. In DSCo-NG , we essentially have two parameters: the cardinality of SAX alphabet when compressing real-valued data to text strings and the sliding window size or length of artificial words. Normally, approaches based on SAX have to specify both the cardinality and a PAA size to which time series are reduced. Since DSCo-NG does not necessarily need dimensionality reduction, we only need to fix for a suitable cardinality, i.e., a good alphabet size that keeps sufficient information during time series compression. To that end, we try to reduce time series using different cardinality values from 3 to 20, which is range supported by major SAX

2 Repository is available at https://github.com/serval-snt-uni-lu/dsco

implementations. For length of artificial words, we also fix a range to 2 to 20 in order to avoid extremely long words, in order to limit the size of language models.

Figure 5.7 presents the classification accuracy from four datasets across different domains. As shown, although these four datasets have different characteristics in terms of training dataset size, time series length and number of classes, there is a clear trend when high classification accuracy is achieved. That is, generally good accuracy is achieved with small to medium SAX alphabet size and the alphabet size has more impact than the word length (imagine projecting the 3D plots to the 2D plane defined by the alphabet size and accuracy axis). This is extremely useful to narrow down the parameter space, even though in fact our parameter space is already small ($18 * 19 = 342$ combinations in total). Note that there are other methods available for finding the optimal parameters. For instance, in [Wang et al., 2016b] the authors have adopted an algorithm named DIRECT. Thanks to the small parameter space and efficiency of DSCo-NG we employ a brute force approach for finding the best parameters for different datasets. Naturally, there is not a single parameter setting that guarantees good performance, since different datasets can be totally different in number of classes, size, time series lengths and variation amplitude. However, it is indeed possible to set the same parameters for datasets with similar characteristics.

5.6.3 *Comparison of Classification Performance*

Now that we have fixed the parameters for DSCo-NG , here we set to compare its performance with its predecessor DSCo and the state-of-the-art approach DTW-based 1NN classifier. As an intuitive and simple benchmark, we only consider the classification accuracy here because the accuracy results are available in the UCR archive and it is in general what the time series classification community compare with. Figure 5.8 presents the classification results. It clearly demonstrates that DSCo-NG outperforms its predecessor. In fact, in *90% (35/39)* of the datasets, DSCo-NG is more or equally accurate compared with DSCo, indeed suggesting performance improvement in accuracy. This is probably due to the fact that DSCo tries to find the best way to segment time series; however, with insufficient training data this segmentation process will result in suboptimal segmentation and thus not as high accuracy. Furthermore, we note that in *72% (28/39)* of the datasets, DSCo-NG also outperforms the state-of-the-art DTW-based 1NN, indicating its superiority in specific datasets. Besides, we would like to remind the readers that DSCo-NG is potentially more scale than 1NN based approaches, especially for datasets with a large training set, e.g., the *ElectricDevices* dataset.

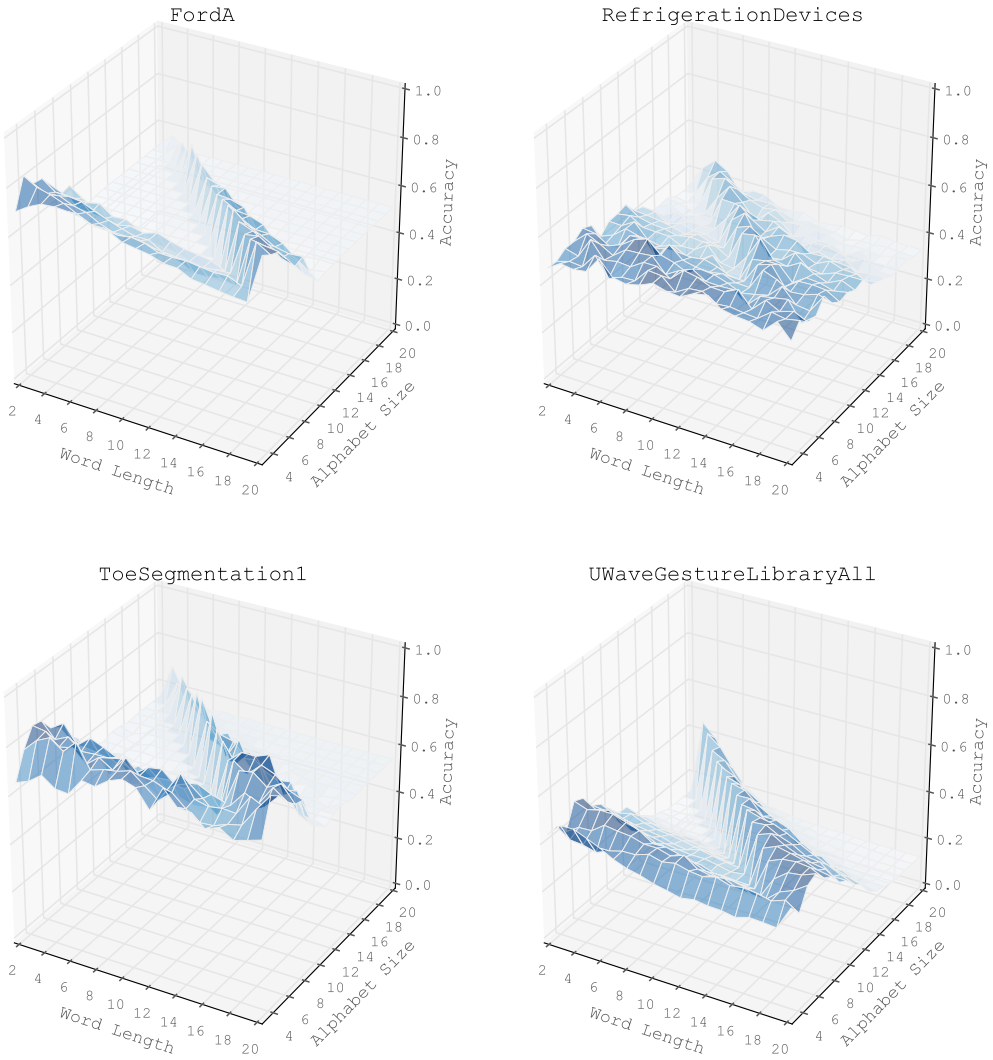

Figure 5.7: 3D surface plots of classification accuracy with different parameters, darker blue indicates higher accuracy.

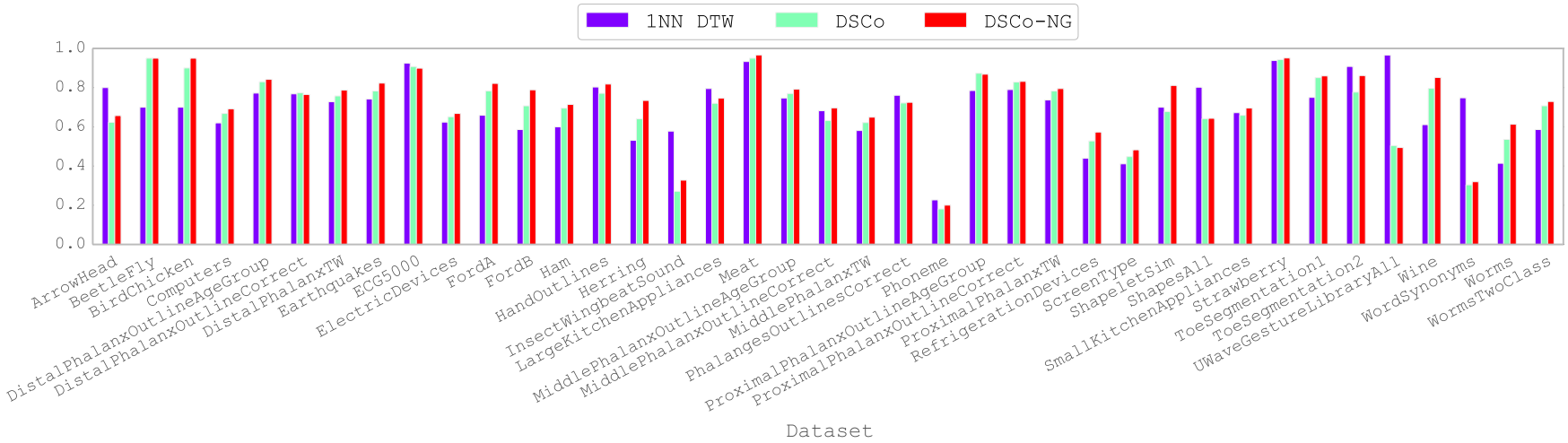

Figure 5.8: Overall accuracy comparison between 1NN with DTW distance, DSCo and DSCo-NG.

We have demonstrated the performance of DSCo-NG through complexity analysis and extensive experiments. Although DSCo-NG outperforms our previous work in vast majority of tested datasets, it remains unclear why DSCo-NG outperforms DTW-based 1NN in certain datasets while underperforms in other ones. To this end, we investigate in which scenarios DSCo-NG performs better. Obviously the size of training set can be an important factor, because our model-based approach has to capture from different and a large number of instances the representative patterns, while for instance-based approaches – e.g. kNN – one representative instance could potentially help accurately classifying all similar instances. This is a major reason why DSCo-NG greatly underperforms 1NN for the *WordSynonyms* dataset, which has many (25) classes but very few (267) training instances. Besides, some classes in this dataset has as few as two instances, making the language model extraction highly inaccurate for DSCo-NG .

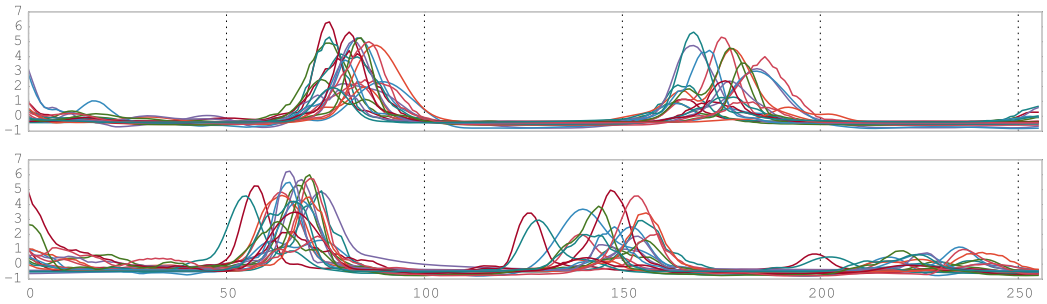

Figure 5.9: All instances of two classes (1 and 5) from *InsectWingbeatSound*'s training set.

Besides training set size, in this study we found another important factor that lies in how small segments constitute a time series. Figure 5.9 shows why DSCo-NG does not perform well for *InsectWingbeatSound*: these two classes consist of similar segments installed in different positions of time series. Thus DSCo-NG will consider these segments as the same word unless we set an extremely long word length. Similarly, DSCo-NG underperforms for *UWaveGestureLibraryAll* because instances in this dataset are composed of three different segments.

Finally, we demonstrate with one example why DSCo-NG outperforms DTW-based 1NN. Consider the two classes from the *FordA* dataset as shown in Figure 5.10. It is obvious that visually it is impossible for a human being to distinguish these two classes, because there are two many samples that are not properly aligned like in Figure 5.9. As a result, for 1NN classifier, these samples could be distracting so that it fails to find similar samples given a testing instance. However, DSCo-NG is able to aggregate samples within a class so that it finds the overall descriptive way to differentiate different classes.

5.7 RELATED WORK

Due to TSC's wide application scenarios, there are a plethora of algorithms made available by the research community. An extensive review of time series mining

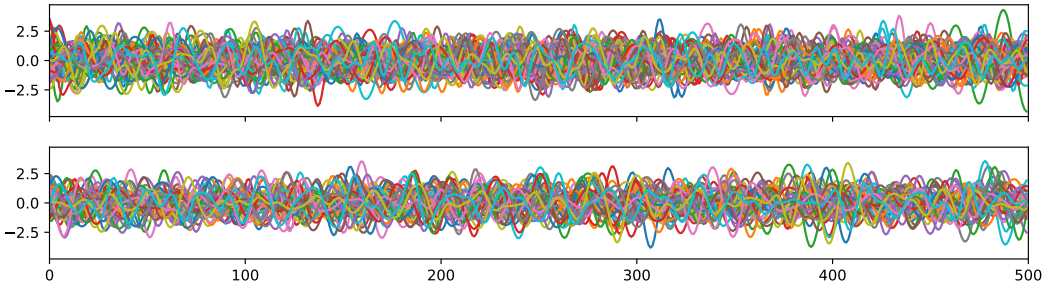

Figure 5.10: First 100 instances of two classes (-1 and 1) from *FordA*'s training set.

has been done by Fu [Fu, 2011]. Here we only survey the works that are closely related to ours due to space limitation. Since DSCo-NG is a compression-based approach, we introduce related approaches that also takes advantage of time series compression techniques.

There are basically two methods for compressing time series, i.e., dimensionality reduction that works on the time axis and numerosity reduction that works on the value axis. Dimensionality reduction mechanisms include Piecewise Linear Representation (PLR) [Keogh, 1997], Piecewise Aggregate Approximation (PAA) [Keogh et al., 2001], and methods that keeps only perceptually important points (PIP) [Chung et al., 2001]. Our previous work [Li et al., 2016c] takes advantage of Discrete Wavelet Transform for dimensionality reduction. On the value axis, Xi et al [Xi et al., 2006] have proposed using numerosity reduction to speed up TSC. Common numerosity reduction techniques include parametric regression, non-parametric clustering and sampling, *etc.*.

Symbolic representation of time series has opened a new avenue for TSC since it makes it possible to borrow paradigms from the text mining community. For instance, the *bag-of-words* approach has inspired the *bag-of-features* [Baydogan et al., 2013; Wang et al., 2013] and SAX-VSM [Senin and Malinchik, 2013] approach for TSC. Furthermore, Representative Pattern Mining (RPM) [Wang et al., 2016b] compresses time series to strings using SAX and then tries to identify the most representative patterns in the training set. These patterns are then used to match against testing instances during classification. Unlike RPM, DSCo does not try to find which patterns are representative or not. Instead, we evaluate testing instances' fitness to each class in an overall perspective.

Note that our compression-based approach is not to be confused with compression-based time series similarity measures [Keogh et al., 2004], which compares the compression ratios of time series under the assumption that compressing similar series would produce higher compression rates than compressing dissimilar ones.

5.8 CONCLUSIONS AND FUTURE WORK

In this work, we have brought up a novel approach named DSCo-NG for time series classification. It works on symbolized time series data and builds per-class language models, against which testing samples are fitted in order to predict their corresponding class labels. Through extensive experiments we are able to prove that DSCo-NG performs similarly or better than some state-of-the-art TSC approaches that works with original numeric data, namely 1NN with Euclidean and DTW distance. By taking advantage of mature algorithms from the NLP community, DSCo-NG is able to achieve close-to-linear time complexity, which will be a great advantage for real-time applications.

Our future work will focus on further improving DSCo's performance, including reducing computation overhead and memory consumption by more effectively pruning n-grams in language models, improving classification accuracy and finding key defining subsequences for better user comprehension. In addition, other symbolization techniques can be taken advantage of to make DSCo more generalized and parameter free.

6

MULTISCALE VISIBILITY GRAPH

横看成岭侧成峰，远近高低各不同。

苏轼
题西林壁

Outline
This chapter presents a multiscale visibility graph representation for time series as well as feature extraction methods for time series classification (TSC). We augment time series by means of their multiscale approximations, which are further transformed into a set of visibility graphs. After extracting probability distributions of small motifs, density, assortativity, *etc.*, these features are used for building highly accurate classification models using generic classifiers (*e.g.*, Support Vector Machine and eXtreme Gradient Boosting). Thanks to the way how we transform time series into graphs and extract features from them, we are able to capture both global and local features from time series. Based on extensive experiments on a large number of open datasets and comparison with five state-of-the-art TSC algorithms, our approach is shown to be both accurate and efficient: it is more accurate than Learning Shapelets and at the same time faster than Fast Shapelets.

6.1 INTRODUCTION

Time series data refer to sequences of data that are ordered either temporally, spatially or in another defined order. They can be frequently found in a variety of domains, including financial data analysis, medical and health monitoring and industrial automation applications. Recently, it turns out to be feasible to model software systems as time series in order to conduct malware detection and classification [Wojnowicz et al., 2017]. Due to their wide application scenarios and abundance, there has been an increasing need for efficient knowledge discovery methods to extract useful information from time series databases. One of the major tasks in time series mining is time series classification (TSC), which consists of applying a learning algorithm on labeled data to train a model that will then be used to predict the classes of samples from an unlabeled data set. Due to the sequential characteristic of time series data, state-of-the-art classification algorithms (such as SVM and Random Forest [Fernández-Delgado et al., 2014]) that perform well for generic data are generally not suitable for TSC. It is thus important and beneficial to have a feature extraction mechanism that transforms the sequential

characteristics of time series data into unordered feature vectors, so that any modern classification algorithm can be taken advantage of. After all, one of the most challenging aspects of TSC lies in the sequentiality property.

Traditionally, researchers often rely on one of the simplest classifiers for TSC: the k Nearest Neighbor (kNN) algorithm. As stated in [Batista et al., 2011], "all of the current empirical evidence suggests that simple nearest neighbor classification is very difficult to beat". To perform well, kNN classifiers leverage the Dynamic Time Warping (DTW) [Berndt and Clifford, 1994; Rakthanmanon et al., 2012] distance which mitigates problems caused by distortion in the time axis. One intrinsic issue with DTW, however, is that it focuses on finding *global* similarities, *i.e.*, the overall curve *shape* of time series. It also requires applications to specify a proper warping window size or to properly align data samples. Figure 2.3 shows an example where DTW fails to identify two subsequences cropped from the same parent-curve due to different data alignment, *i.e.*, a phase shift. Specifically, plots b and c are subsequences in a with the same length. However, the DTW distance between b and c is larger than that between b and d or c and d. In this specific case, all test samples resembling the overall shape of b or c may be mistakenly labeled as the class of d, leading to poor classification accuracy. As a result, DTW can be sensitive to data alignment/segmentation and it performs better when data are properly curated. In practice, however, well-aligned time series data are difficult or expensive to come by [Hu et al., 2013].

To address the phase issue of DTW- and other distance-based 1NN approaches, the research community has proposed approaches that focuses on finding defining *local* features/subsequences in order to be invariant to data alignment and rotation. Popular methods that fall into this category include Bag-of-Patterns [Lin et al., 2012], SAX-VSM [Senin and Malinchik, 2013] and shapelets-based algorithms [Ye and Keogh, 2009], such as Fast Shapelets (FS) [Rakthanmanon and Keogh, 2013] and Learning Shapelets (LS) [Grabocka et al., 2014]. The majority of these techniques have taken advantage of text feature extraction approaches – *e.g.*, TF-IDF – after converting time series into alphabetical strings. Such conversion is often done via Symbolic Aggregate approXimation (SAX) [Lin et al., 2007], which requires two parameters (*i.e.*, cardinality and PAA window size) to be set and it may not always be trivial to find the best pair. Besides, many approaches attempt to find time series subsequences that are representative of each class, *e.g.*, shapelets by definition are defining time series subsequences that are calculated by exhaustive or optimized search. Overall, many of these methods have suffered from high computation complexity or suboptimal classification accuracy [Wang et al., 2016b].

Graph representations for TSC, on the other hand, have not been investigated extensively by the data mining research community possibly due to their high computation complexity. Nevertheless, thanks to recent development of graph mining algorithms [Newman and Girvan, 2003; Batagelj and Zaversnik, 2003], some of the formerly complex problems can be solved extremely efficiently with optimization and parallelization techniques [Ahmed et al., 2015]. Such advances give us the opportunity to re-evaluate the possibility of taking advantage of graph represen-

tations and extracting graph features for building an efficient and accurate TSC algorithm.

This chapter proposes a novel approach for TSC that considers time series as complex graphs/networks and extracts from these networks important statistical features, which are fed to modern generic classifiers to learn structural knowledge from the original time series. After evaluating the classification performance with a large open dataset, we find out that our approach is capable of making efficient and accurate classification predictions. The main contributions of this chapter are listed as follows:

- We present a multiscale graph representation for time series, so that both global and local features from time series can be captured, making this approach agnostic to time series alignment and outperform major distance-based TSC algorithms.
- Since we transform time series that are intrinsically sequential into unordered feature vectors, it is then suitable for taking advantage of modern generic classifiers (*e.g.*, RF, SVM and XGBoost) for efficient feature selection and classification. This clear separation of feature extraction and actual classification can help researcher focus on finding insightful characteristics in time series data without the need of reinventing the wheel and designing a classifier from scratch specifically for time series.
- We propose a novel feature extraction and classification method for time series based on calculating probability distributions of small motifs (*i.e.*, repeated patterns) in visibility graphs and other statistical features such as density, degree statistics, assortativity and coreness. This feature extraction mechanism is parameter-free, so that it can be easy to use and help yield reproducible results.
- We have intentionally chosen a collection of statistical features that are computationally efficient to extract from graphs and validated their effectiveness in controlled experiments. Moreover, since our feature extraction and classification process is inherently parallel, it is suitable for and capable of large scale data explorations.
- After extensively evaluating our approach with a large number of open datasets and comparing with related research efforts, experiment results indeed suggest that accurate and efficient classification can be obtained following this paradigm.

The remainder of this chapter is structured as follows. Section 6.2 lays down the necessary background. Next, we present our approach in section 6.3 and detail TSC accuracy and efficiency evaluation results along with a case study in section 6.4. For interested readers, section 6.5 introduces research work related to ours. Finally, we conclude the chapter with future research directions in section 6.6.

6.2 BACKGROUND

In this section, we set to prepare readers for the necessary background. Especially, we present how graph representations can be used for time series and TSC. For the sake of clarity, we reproduce a few definitions from Chapter 2.

Traditionally, time series refer to a sequence of numbers that are chronologically ordered. However, in the research community time series have a much broader scope and do not associate strictly with timestamps:

Definition 6.1 (Time series)
A time series instance T *is an ordered sequence of* n *real-valued variables,* i.e., $T = (v_1, ..., v_n), v_i \in \mathbb{R}$.

If we consider each point in a time series as a single feature in a vector, then time series data usually have a huge number of features. When considering these features as a vector in an n-dimensional space, time series data are often high dimensional. Due to difficulties to conduct knowledge discovery tasks on high dimensional data, it is frequently required to reduce the dimensionality of time series in order to improve computation efficiency:

Definition 6.2 (Dimensionality reduction)
The dimensionality of a time series sample T *is the length of* T*, denoted by* $|T|$*. If* T' *is an approximated representation of* T *and* $|T'| \ll |T|$*, then* T' *is a dimension-reduced representation of* T.

The research community has proposed a number of dimensionality reduction techniques for time series, including sampling [Åström, 1969], Piecewise Linear Representation (PLR) [Keogh, 1997], Piecewise Aggregate Approximation (PAA) [Keogh and Pazzani, 2000], *etc.*. Among them PAA is perhaps one of the simplest and most widely applied approaches. PAA reduces time series $T = (v_1, ..., v_n)$ from n dimensions to s dimensions by firstly dividing the data into s segments of equal size, then the approximation is a vector of the mean values of the data readings per segment [Keogh and Pazzani, 2000; Lin et al., 2007]. Let $T' = (v'_1, ..., v'_s)$ be this vector where v'_i is computed by equation 6.1. For the sake of simplicity, $\frac{n}{s}$ is often chosen to be an integer or rounded to the nearest one.

$$v'_i = \frac{s}{n} \sum_{k=\frac{n}{s}(i-1)+1}^{\frac{n}{s}i} v_k \tag{6.1}$$

One of the core routines in distance-based classification algorithms involves evaluating the dissimilarity (or similarity) of two time series. There are a number of dissimilarity measures for time series, two of the most frequently used measures in the research community are Euclidean distance and DTW distance. The Euclidean distance maintains a one-to-one mapping of all the points in two series.

On the other hand, the DTW distance tries to find the best mapping of points in two series using the dynamic programming paradigm, so that the minimum distance between these two series is achieved. The paradigm is called "time warping" since the time axis of series can be expanded or compressed in order to ensure the minimum distance, *i.e.*, an i^{th} point in X can be mapped to a j^{th} point (it is possible that $i \neq j$), or one point in X may even be mapped to multiple points in Y.

6.2.1 *Visibility Graph*

Aiming for taking advantage of graph theories as a way of characterizing time series, an algorithm named visibility graph (VG) [Lacasa et al., 2008, 2009] is proposed to transforms time series into a network structure. The creation of VGs relies on an extremely simple idea: each point in a time series is treated as a vertical bar, whose height is the corresponding numerical value. When considering these bars on a landscape, it is then straightforward that the top of a bar may be visible from the top of other bars. Assume each time-step as a vertex in a graph, then two vertices are connected if the top of the vertical bars are visible to each other, *i.e.*, there exists a straight line from the top of the two bars without intersecting with other bars. More formally,

Definition 6.3 (Visibility graph)
Given a time series $T = (v_1, ..., v_n)$*, its VG representation* $G = (V, E)$ *has* n *vertices:* $V = (1, ..., n)$*. An edge* $e = (i, j) \in E$ *iff* $\forall k$ *such that* $i < k < j$ $(1 \leqslant i, j \leqslant n)$ *inequality* $v_k < v_j + (v_i - v_j)\frac{j-k}{j-i}$ *is satisfied.*

It is obvious that VGs are undirected, although it is possible to create a directed version by limiting the direction of viewpoints from the vertical bars. Besides, VGs are always connected since each node will always be visible to its neighbors. Finally, VGs are invariant when time series undergo affine transformations, *i.e.*, the visibility criterion remains fulfilled when rescaling the time series either horizontally or vertically. However, VGs are not suitable for non-stationary time series, *i.e.*, those having monotonically increasing/decreasing trends in time. Such trends should be removed before applying VG generation. The goal of VGs is to characterize structural properties [Lacasa et al., 2009] of time series, such as periodicity, fractality, *etc.*, although it is claimed that it can be extended to weighted VGs in order to quantitatively distinguish generic time series [Supriya et al., 2016].

Creation of VGs from time series without optimization generally has $O(n^2)$ computation complexity, where n is the dimensionality of time series data. However, a more efficient VG generation algorithm [Afshani et al., 2017] can have a sub-quadratic computation complexity. Specifically, this algorithm can reduce the complexity of time series VG generation down to $O(n \log^2(n))$. When further taking advantage of parallelization, $O(n \log^2(n))$ work can be effectively solved within $O(\log^2(n))$ time. Another simplified variant of VG, horizontal visibility graph (HVG) [Luque et al., 2009], only connects nodes i and j if a horizontal line can be drawn between these nodes. Creation of HVGs without any optimization generally has a computation complexity of $O(n)$.

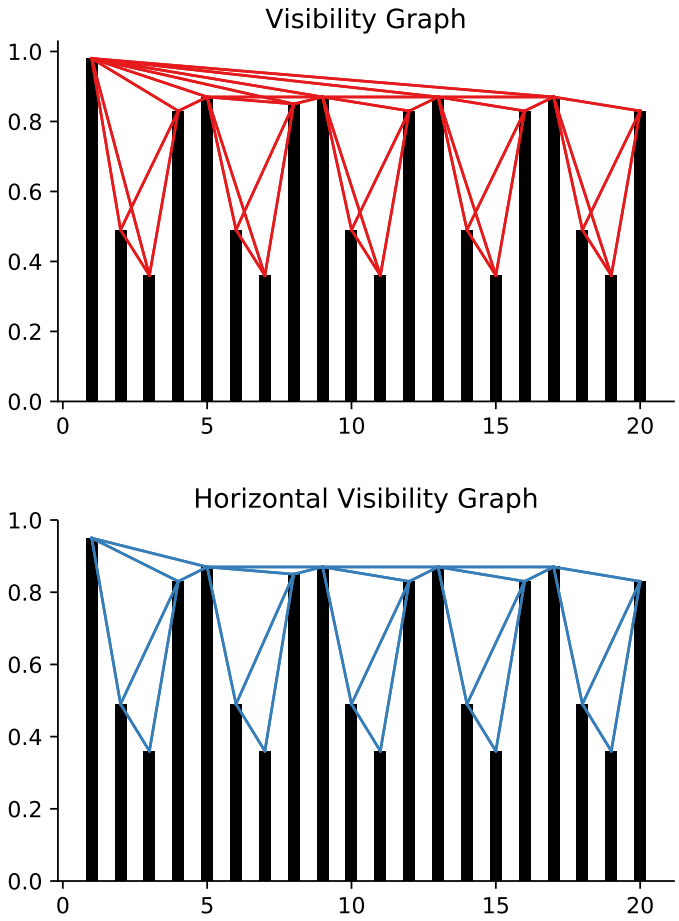

Figure 6.1: An example of converting time series to visibility graph and horizontal visibility graph.

Definition 6.4 (Horizontal visibility graph)
Given a time series $T = (v_1, ..., v_n)$*, its HVG representation* $\bar{G} = (V, E)$ *has* n *vertices:* $V = (1, ..., n)$*. An edge* $e = (i, j) \in E$ *iff* $\forall k$ *such that* $i < k < j$ $(1 \leqslant i, j \leqslant n)$ *inequality* $v_i, v_j > v_k$ *is satisfied.*

VG and its variants have been shown to be able to differentiate certain time series. For instance, [Iacovacci and Lacasa, 2016] have extracted motifs from HVGs and claimed the motif statistics can be used for differentiating various types of time series data, including white Gaussian noises, fully chaotic logistic maps and noisy fully chaotic logistic maps. The authors further claimed that HVG motif profiles from heart rate time series can be used to cluster different types of meditative activities.

Intuitively, VGs and HVGs are very similar concepts and HVG is a subgraph of VG. However, when extracting statistics from them, VGs can be more capable of capturing global features, while HVGs are often more sensitive to local variations. As a result, VGs and HVGs can be joined to provide more accurate representations of time series data. We will discuss more about such heuristics in section 6.4.2. For

simplicity, in the remainder of this chapter the term VG indicates the combination of VG and HVG if HVG is not explicitly specified.

6.2.2 *Graph Classification with Deep Neural Networks*

Graphs in computer science are often represented with edge lists or adjacency matrices. When visibility graphs are represented in a matrix form, classification of them can then be converted to a problem of image recognition. Consequently, can we first transform time series into visibility graphs – which are represented by means of matrices/images, and try to apply image recognition techniques for classifying time series? For instance, Figure 6.2 illustrates the converted images from time series WVGs. Human beings may immediately find the characteristics corresponding to most classes. Similar to this time series image recognition paradigm, [Wang and Oates, 2015] have also attempted encoding time series as images using Gramian Angular Fields (GAF) and Markov Transition Fields (MTF), so that CNN can be taken advantage of for classification.

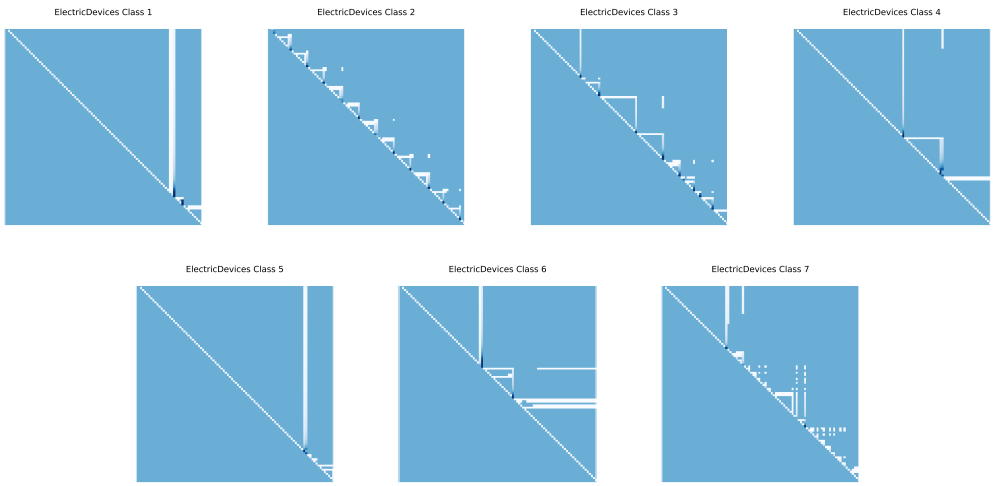

Figure 6.2: Time series instances from different classes of `ElectricDevices` are transformed into VGs and whose matrices are plotted as images.

In order to investigate if computers can tell different classes of WVG images apart, we employ one of the most advanced image recognition techniques named Residual Networks (ResNet) [He et al., 2016]. The experiments are conducted on a HPC platform [Varrette et al., 2014] so that we can take advantage of GPUs to accelerate the training tasks. Specifically, we use NVIDIA Tesla K40M GPUs with 2880 stream cores and 12GB memory for training ResNet models consisting of 50 layers. We have selected to experiment on datasets with large number of training instances per class in order to reduce the risk of overfitting and increase the generalization of the ResNet models. Besides, since WVG images created from some datasets can be large due to long time series lengths/dimensionality, to avoid memory errors we have to conduct Principal Component Analysis (PCA) to reduce the dimensionality of WVG matrices. Although GPUs can be a magnitude (*e.g.*, ten

times) faster than CPUs, training these models may still take several (more than 12) hours.

Table 6.1: Converting time series into WVGs and using ResNet for classification: classification accuracy compared to 1NN with Euclidean distance and 1NN with DTW.

Dataset	1NN ED	1NN DTW	ResNet50	ResNet50 (PCA)
ElectricDevices	55.00	62.40	69.55	57.18
FordA	65.90	65.90	N/A	76.51
FordB	55.80	58.60	N/A	68.76
PhalangesOutlinesCorrect	76.10	76.10	74.59	63.40
ProximalPhalanxOutlineCorrect	80.80	79.00	31.62	75.60
Strawberry	93.80	93.80	93.80	72.43

Table 6.1 presents the classification accuracy results. When compared with popular benchmark classifiers, *i.e.*, 1NN with Euclidean and DTW distance, ResNet does not appear to have yielded good results. Besides, ResNet models are extremely difficult to train and relies on GPU accelerators, which may not be widely accessible. As a result, it can be impractical to apply this approach for TSC and we may need to extract graph features for practical classification.

6.2.3 *Graph Features*

Extracting features from graphs has become a popular research topic thanks to the recent applications in social network analysis, physics as well as bio-informatics. There are a number of research avenues in graph mining, the most popular ones are about finding communities or clusters within large graphs and characterizing graphs by means of finding and counting recurrent patterns. Since time series VGs are always connected, it is thus not immediately helpful to extract clusters, since clusters in these graphs will always correspond to subsequences of original time series.

Table 6.2: Computation time of finding motifs from an undirected graph with 1000 nodes and 1300 edges using GTrieScanner.

Motif size	4	5	6	7	8
No. of connected motifs	6	21	122	853	11,117
Time (s)	0.011	0.25	5.46	105	1,732

Graph motifs or graphlets, however, are especially interesting since they are subgraph structures or patterns in a larger graph that are recurrent and statistically significant. Finding motifs in graphs is generally a complex issue, but there have been a number of research work for efficiently extracting motifs from graph data, such as GTrieScanner [Ribeiro et al., 2010] and Parallel Parameterized Graphlet Decomposition (PGD) [Ahmed et al., 2015]. Due to the fact that the number of motifs increases exponentially with motif size, the complexity of finding motifs

is also exponentially correlated with motif size (*cf.* Table 6.2), researchers thus generally focus on optimizing computation efficiency of locating small motifs (of size up to four). PGD is the state-of-the-art approach for efficiently finding and counting small motifs in graphs and can be several magnitudes faster than other approaches. Table 6.3 shows all possible small graph motifs up to size four. Note that PGD works only for undirected graphs, while GTrieScanner works on both directed and undirected graphs. However, motifs extracted by GTrieScanner are only connected motifs and the running time is significantly slower than PGD. As a result, in this chapter we take advantage of PGD for small motif counting.

Table 6.3: All graph motifs up to size 4. Note that connected graphs may contain disconnected motifs.

	#	Motif	Name		#	Motif	Name
Connected	$M2_1$		2-edge	Disconnected	$M2_2$		2-node-independent
	$M3_1$		3-triangle		$M3_3$		3-node-1-edge
	$M3_2$		3-path		$M3_4$		3-node-independent
	$M4_1$		4-clique		$M4_7$		4-node-triangle
	$M4_2$		4-chordal-cycle		$M4_8$		4-node-star
	$M4_3$		4-tailed-triangle		$M4_9$		4-node-2-edges
	$M4_4$		4-cycle		$M4_{10}$		4-node-1-edge
	$M4_5$		4-star		$M4_{11}$		4-node-independent
	$M4_6$		4-path				

Researchers argue that motif distribution extracted from VGs can be extremely helpful for identifying different types of time series. For instance, Figure 6.3 illustrates that motif distributions are easily distinguishable with VGs that are generated from artificial time series data.

However, in practice the motif distributions may not be as distinguishable as those from artificial data. For example, Figure 6.4 illustrates the motif distributions of three different classes of time series from the `ArrowHead` dataset. As shown, it can be very difficult for human beings to tell one class apart from another given only these distributions since the probability distributions of different classes tend to overlap, especially for instances from class 2 and 3. Furthermore, if we only consider connected motifs, the differences of motif distributions are slightly bigger, as shown in Figure 6.5, but it is still difficult to tell specific classes apart due to overlapping of distribution. As a result, although motif distributions can be helpful, more features from VGs need to be extracted in order to draw clearer lines between different classes during classification.

Besides small motifs, other graph features are also easy to obtain, *e.g.*, vertex and edge statistics as well as structural metrics. In this chapter we consider some additional graph features listed as follows:

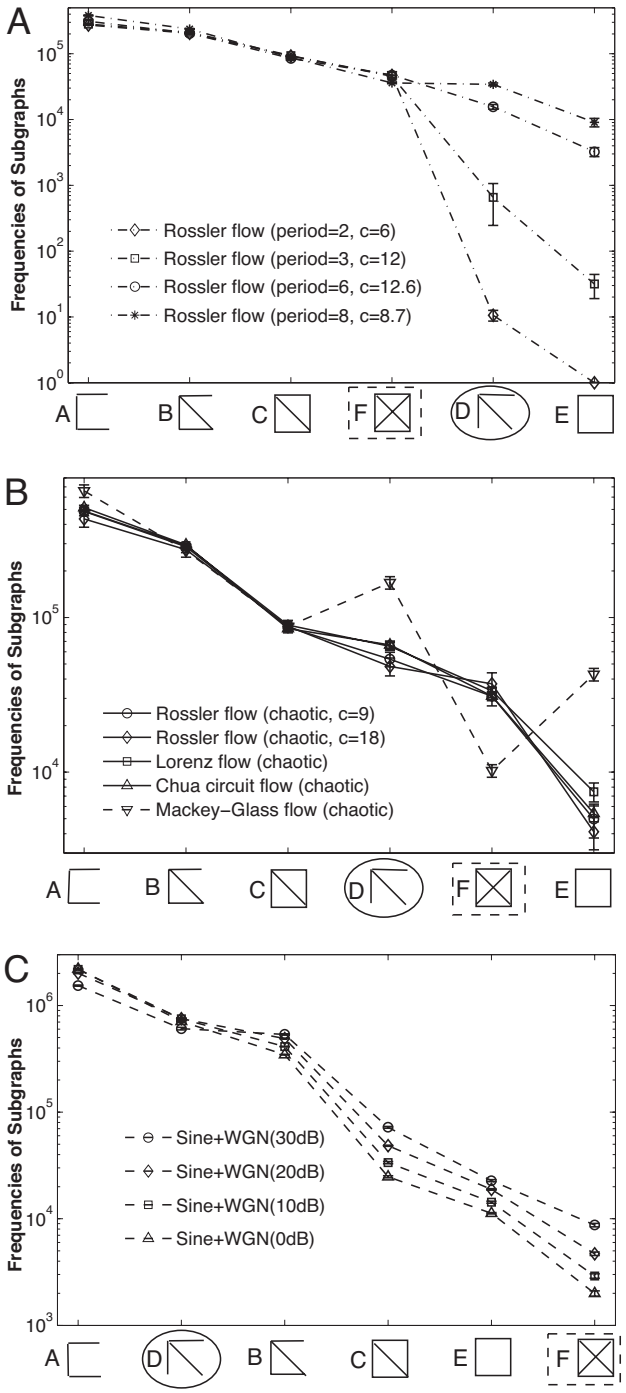

Figure 6.3: Motif distribution of artificially generated data by [Xu et al., 2008]. (A) Periodic flow. (B) Chaotic flow. (C) Periodic flow with Gaussian noise.

- Density: the ratio of the number of edges to the number of all possible edges. Graph density is computed following equation 6.2 and has a computation complexity of $O(1)$.

$$p = \frac{2|E|}{|V|(|V|-1)} \tag{6.2}$$

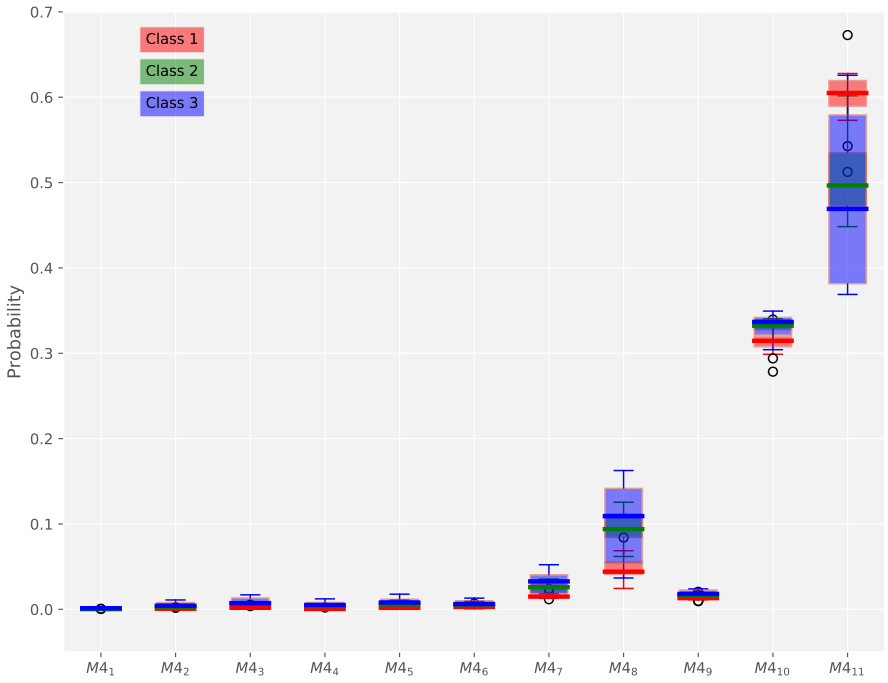

Figure 6.4: Boxplots of all motif probability distribution of different classes from the ArrowHead Dataset's training set.

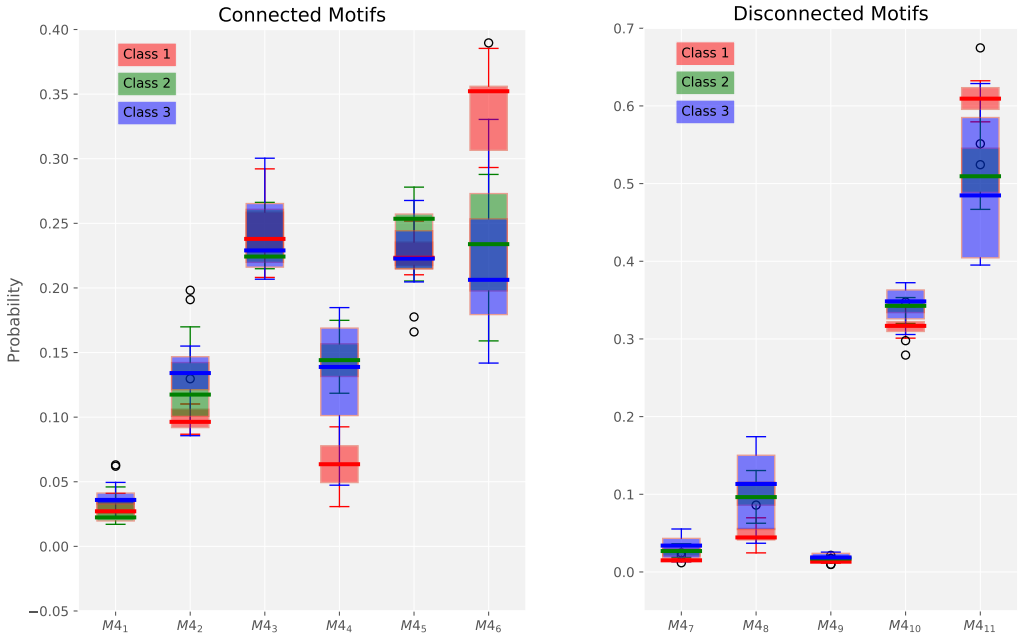

Figure 6.5: Boxplots of connected and disconnected motif probability distribution of different classes from the ArrowHead Dataset's training set.

- K-core: the K-core of a graph $G = (V, E)$ is a maximal subgraph $H = (V', E')$ in which each vertex has a degree of at least K, *i.e.*, $\forall v \in V' : \deg_H(v) \geqslant K$. Consequently, it is a cohesiveness measurement of interlinked subgraphs within a network and K is computed by equation 6.3. Computing K has an $O(|E|)$ time complexity [Batagelj and Zaversnik, 2003].

$$K = \underset{H \in G}{\operatorname{argmax}} \operatorname{Core}_H \geqslant K \tag{6.3}$$

- Assortativity coefficient: a metric to measure the correlation of vertices in a graph through calculating the Pearson correlation coefficient of degree between pairs of connected vertices. Equation 6.4 shows how assortativity coefficient is computed,

$$r = \frac{\sum_{xy} xy(e_{xy} - a_x b_y)}{\sigma_a \sigma_b} \tag{6.4}$$

  where a_x and b_y represents respectively the fraction of edges that start and end at vertices with values x and y, and e_{xy} is the measure of assortativity, such that $\sum_{xy} e_{xy} = 1$, $\sum_y e_{xy} = a_x$ and $\sum_x e_{xy} = b_y$. Finally, σ_a and σ_b are the standard deviations of the distributions a_x and b_y. Computing r has an $O(|E|)$ time complexity [Newman and Girvan, 2003].

- Degree statistics including max, min and mean degree per vertex. Naturally, computing such statistics has an $O(|V|)$ time complexity.

Note that there are a plethora of features that can be extracted from graphs [Costa et al., 2007], and such features are not necessarily equally easy to obtain. For instance, the diameter of a graph – which is the shortest distance between the two most distant vertices in the graph – can be computationally expensive to calculate since it demands $O(|V|(|V| + |E|))$ computation. Nonetheless, this chapter does not intend to find the exclusive set of features that are both efficient and descriptive for time series VGs. Rather, we try to investigate if these aforementioned statistical features are indeed helpful for TSC.

6.3 MULTISCALE VISIBILITY GRAPH

Time series data differ greatly in characteristics depending on how they are captured, sampled and their underlying applications: in one domain global features may be helpful, while in another domain local features (*i.e.*, defining subsequences) can become more important for classification. After transforming real-valued time series into VGs, specific features from such VGs – such as probability distributions of small motifs – are more reflective of local features than global ones. We refer to this type of representation as Uniscale Visibility Graph (UVG) in the remainder of this chapter. In this case, extracting more global features (*i.e.*, prob-

ability distributions of large motifs) becomes exponentially expensive for computation. To mitigate this problem, we propose the Multiscale Visibility Graph (MVG) representation such that each time series sequence is transformed into a set of dimensionality-reduced approximations: these downscaled approximations are then converted into a set of VGs. Consequently, features are extracted from each graph in MVGs, thus approximating the process of extracting features of different scales. This approach is inspired by computationally efficient wavelet transform [Cohen et al., 1993], with the exception that time series in each scale are represented with graph structures instead of numerical values. We first define the multiscale representation of time series as follows:

Definition 6.5 (Multiscale Approximation)
Given a time series $T_0 = (v_1, ..., v_n)$*, its approximated multiscale representation is a set of time series* $\hat{\mathbb{T}} = (T_1, T_2, , ..., T_m)$*, where* T_i $(1 \leqslant i \leqslant m)$ *is a downscaled approximation of* T_0*, such that* $|T_i| = |T_0|/2^i = n/2^i$*. In addition, to avoid tiny and meaningless representations we enforce a constant threshold* τ *for the downscaled approximations,* i.e., $|T_m| > \tau$.

The downscaling of T_0 can be achieved with widely used dimensionality reduction techniques such as PAA (*cf.* equation 6.1). Since the size of downscaled time series representations follows an exponential decay, time series multiscale representations $\hat{\mathbb{T}}$ often consists of a small number downscaled series. Besides, considering $\sum_{i=1}^{\infty} n * 2^{-i} = n$, theoretically the maximal dimension of $\hat{\mathbb{T}}$ when fully expanded is n.

Definition 6.6 (Multiscale representation)
Given a time series T_0 *and its approximated multiscale representation* $\hat{\mathbb{T}} = (T_1, T_2, , ..., T_m)$*, its multiscale representation is the union of* $\hat{\mathbb{T}}$ *and* T_0*,* i.e., $\mathbb{T} = (T_0, T_1, T_2, , ..., T_m)$.

Approximated multiscale representations can help smoothing time series and reducing noises, while full multiscale representations consist of both the original time series and augmented versions. Each series in multiscale representations can be transformed into VGs, thus we have a natural definition for multiscale visibility graphs, which are supersets of approximated multiscale visibility graphs (AMVGs):

Definition 6.7 (Multiscale visibility graph)
Given a time series T *and its multiscale representation* $\mathbb{T} = (T_0, T_1, T_2, , ..., T_m)$*, its multiscale visibility graph is a set of graphs* $\mathbb{G} = (G_0, G_1, G_2, , ..., G_m)$*, where* G_i $(1 \leqslant i \leqslant m)$ *is the corresponding visibility graph created from* T_i.

Since the number of vertices in G_i equals the dimensionality of T_i, to avoid meaningless trivial graphs it is natural to set τ to a small integer (*e.g.*, $\tau = 15$), such that the smallest graph in $\mathbb{G}$ contains more than τ vertices. Note that τ should not be considered as a parameter for the feature extraction process. Rather, it is more an optimization trick and bearing a default value of 0 will not cause any issues, since feature selection is done during classification.

6.3.1 *Feature Extraction*

As shown in Algorithm 6.1, feature extraction in MVGs follow the same paradigm as extracting features from individual graphs in an MVG and concatenating all features together, since graph features extracted in this chapter are solely statistical and do not pertain orders. Consequently, these features can be fed into any generic classification algorithms [Fernández-Delgado et al., 2014] well studied in the machine learning and data mining community. It is utterly important that this feature extraction transforms the sequential characteristics of time series data into unordered feature vectors, so that any modern classification algorithm can be taken advantage of.

Algorithm 6.1 Building time series MVGs and extracting features from them.

procedure ExtractFeatures(T)
  $\mathbb{F} \leftarrow \varnothing$ ▷ Feature set
  $\mathbb{G} \leftarrow \varnothing$ ▷ MVGs
  $T' \leftarrow T$
  while $|T'| > \tau$ **do** ▷ Ignore trivial graphs
    $\mathbb{G} \leftarrow \mathbb{G} \cup \mathrm{BuildVGAndHVG}(T')$
    $T' \leftarrow \mathrm{DimensionalityReduction}(T)$ ▷ $|T'| \leqslant \frac{|T|}{2}$
  for $G \in \mathbb{G}$ **do**
    $\mathbb{M} \leftarrow \mathrm{MotifProbabilityDistribution}(G)$
    $\mathbb{M} \leftarrow \mathrm{Normalize}(\mathbb{M})$
    $\mathbb{F} \leftarrow \mathbb{F} \cup \{\mathbb{M}, \mathrm{OtherStatistics}(G)\}$
  return $\mathbb{F}$

Although features have become unordered, it is nevertheless important to carefully curate them in order to achieve better classification results. Note that PGD is very efficient in counting motifs in graphs, and the dominant features from time series MVGs will be the probability distribution of motifs of different sizes:

Definition 6.8 (Motif probability distribution)
Given a time series visibility graph G *and the set of motifs* $\mathbb{M}$*, the motif probability distribution* $\mathbb{P}_G$ *is the set of probabilities corresponding to each motif in* $\mathbb{M}$*.*

Empirically, the distribution of connected and disconnected motifs vary greatly in graphs. It is thus desirable to calculate separately motif probability distributions depending upon motif connectivity. To that end, the motif probability distributions (MPDs) are calculated per motif size and connectivity. This essentially normalizes extracted motif features. Specifically, MPDs are normalized according to the following five groups (*cf.* Table 6.3): $\{\{M2_1, M2_2\}, \{M3_1, M3_2\}, \{M3_3, M3_4\}, \{M4_1, ..., M4_6\}, \{M4_7, ..., M4_{11}\}\}$. Finally, it is obvious that other graph features – *e.g.*, graph density and assortativity coefficient – are independent of MPDs. As a result, no further curation for such features is needed.

6.3.2 *Classification*

When features are extracted, we can feed them into a generic classifier to learn from labeled samples and predict the class of unlabeled ones. We are in favor of most well-known and widely accepted and well optimized algorithms such as SVM, Random Forest and eXtreme Gradient Boosting (XGBoost) [Chen and Guestrin, 2016] for this task. Especially, as a highly optimized distributed gradient boosting library, XGBoost runs on major distributed environment. It has gained remarkable adoption since its inception and is well known for its performance in machine learning and data mining competitions.

Typical classification tasks involve learning models and selecting a performant estimator through the process of validation. Although in this study we have proposed a parameter-free method for feature extraction, it is still required to tune the hyper-parameters for generic classifiers. Note that hyper-parameters in machine learning refers to parameters that are external to the model and such values cannot be estimated from the training data. As a result, they are often set using heuristics. For instance, the k in kNN classifier can be considered as a hyper-parameter, since "there is no analytical formula available to calculate an appropriate value" [Kuhn and Johnson, 2013]. In order to tune hyper-parameters in our classifiers, we conduct cross-validation to evaluate the performance of estimators and apply grid search to find the most satisfactory estimator based on the cross entropy scores:

$$-\log P(\hat{y}|y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{6.5}$$

where y is the ground truth, and $\hat{y}$ is the probabilistic predictions by an estimator. Since datasets may be highly imbalanced, which will lead to degraded classification results, we can apply random oversampling techniques over the minority class and use stratified cross validation to preserve class balance when validating models.

6.4 EVALUATION

To evaluate our approach, we conduct experiments using a large number of publicly accessible datasets. We first introduce the datasets. Then we list and validate our heuristics, followed by creating an accurate classifier with stacked generalization. Next, we compare the accuracy and efficiency of our method with state-of-the-art approaches. Finally, we close this section with a case study and discussions.

6.4.1 *Datasets*

When validating our heuristics, we consider the most popular and largest open dataset for TSC: the UCR Time Series Classification Archive [Chen et al., 2015].

Specifically, we use a subset from the UCR archive that are more recent (added after the summer of 2015), which include datasets from various fields ranging from electrocardiograms (ECG) to intra-species image recognition data. These datasets have a uniform file format and consistent internal data structures, making it possible to conduct batch processing in a content-agnostic manner. Furthermore, datasets in this subset are generally larger in size, making it more reliable to evaluate the scalability of TSC algorithms.

When comparing our approach with state-of-the-art approaches, we take advantage of the UEA & UCR Time Series Classification Repository since it contains all datasets from the UCR archive. In addition, this repository also provides open source implementation for more than 18 TSC algorithms [Bagnall et al., 2017] and benchmarking results are publicly available from the repository website[1]. We compare our results with relevant algorithms using the default train and test split from this repository.

6.4.2 *Validating Heuristics*

Before diving directly into MVG representations, it would be a prerequisite to validate our heuristics step by step. We summarize these heuristics below:

(a) Motif statistics from VGs and HVGs can serve better during classification when combined with other graph features such as degree statistics.

(b) Features from HVGs are more capable of capturing *local* characteristics while those from VGs are capable of capturing *global* characteristics. Combining features from both HVGs and VGs can help yield more accurate classification results.

(c) Multiscale representations are able to reveal time series features at different scales, and a generic classifier is able to conduct feature selection and find important features to perform more accurate classification than using features without multiscale augmentation.

When validating these heuristics, we feed the features corresponding to each heuristic to an XGBoost classifier for 3-fold cross-validation and model selection. A set of hyper-parameters have been set for grid search, including the learning rate (three choices from 0.01 to 0.3), number of estimators (10 choices from 10 to 100), max tree depth (10 or 20). In order to prevent overfitting, the subsampling and column sampling hyper-parameters are both set to 0.5 to randomly collect half of the data instances and features to grow trees. To reduce the impact of random over sampling on minority classes and floating-point summation issues in parallel processing, all experiments have been repeated five times and average accuracy scores are calculated. Finally, all experiments are conducted on a Linux server with two Intel Xeon E5-2430 CPUs with a clock rate of 2.20GHz. With this

1 http://timeseriesclassification.com/

setup, we illustrate step by step how experiments are setup and present the final classification results in Table 6.4.

Table 6.4: Error rates of classifying 39 UCR datasets compared with 1NN-Euclidean and 1NN-DTW. Different heuristic combinations are taken into account. Bold-faced values indicate lowest error rates (including ties) for specific datasets in all experiments.

					Scales →		UVG					AMVG	MVG
					Type of Graphs →		HVG		VG		VG+HVG		
					Features →		MPDs	All	MPDs	All	All		
Dataset	#Cls.	#Train	#Test	Dim.	1NN-ED	1NN-DTW	A	B	C	D	E	F	G
ArrowHead	3	36	175	251	**0.200**	**0.200**	0.482	0.449	0.406	0.407	0.385	0.405	0.398
BeetleFly	2	20	20	512	0.250	0.300	0.440	0.410	0.250	0.170	0.250	**0.150**	0.180
BirdChicken	2	20	20	512	0.450	0.300	0.260	0.150	**0.000**	**0.000**	**0.000**	0.050	0.050
Computers	2	250	250	720	0.424	0.380	0.294	0.284	0.367	0.338	0.281	0.292	**0.266**
DistalPhalanxOutlineAgeGroup	3	139	400	80	0.218	0.228	0.204	0.202	0.214	0.196	0.202	0.196	**0.188**
DistalPhalanxOutlineCorrect	2	276	600	80	0.248	0.232	0.409	0.389	0.251	0.263	0.251	0.264	**0.231**
DistalPhalanxTW	6	139	400	80	0.273	**0.272**	0.342	0.348	0.298	0.300	0.315	0.275	0.279
ECG5000	5	500	4500	140	**0.075**	**0.075**	0.289	0.182	0.137	0.112	0.116	0.076	**0.075**
Earthquakes	2	139	322	512	0.326	0.258	0.262	0.255	0.286	0.265	**0.245**	0.276	0.283
ElectricDevices	7	8926	7711	96	0.450	0.376	0.503	0.493	0.392	0.357	0.366	0.368	**0.338**
FordA	2	1320	3601	500	0.341	0.341	0.009	0.009	0.254	0.220	0.007	0.167	**0.006**
FordB	2	810	3636	500	0.442	0.414	0.328	0.318	0.313	0.290	0.271	0.257	**0.230**
Ham	2	109	105	431	0.400	0.400	0.463	0.463	0.347	0.345	0.389	0.389	**0.343**
HandOutlines	2	370	1000	2709	0.199	**0.197**	0.293	0.288	0.275	0.225	0.221	0.215	0.206
Herring	2	64	64	512	0.484	0.469	0.425	0.419	0.450	0.484	0.381	0.431	**0.288**
InsectWingbeatSound	11	220	1980	256	0.438	**0.422**	0.808	0.763	0.586	0.577	0.557	0.484	0.488
LargeKitchenAppliances	3	375	375	720	0.507	**0.205**	0.461	0.414	0.490	0.478	0.380	0.346	0.325
Meat	3	60	60	448	0.067	0.067	0.497	0.490	0.160	0.097	0.117	**0.050**	0.080
MiddlePhalanxOutlineAgeGroup	3	154	400	80	0.260	0.253	0.274	0.279	0.246	**0.238**	0.267	0.266	0.247
MiddlePhalanxOutlineCorrect	2	291	600	80	**0.247**	0.318	0.534	0.531	0.305	0.294	0.337	0.426	0.314
MiddlePhalanxTW	6	154	399	80	0.439	0.419	0.471	0.443	0.419	0.426	**0.401**	0.434	0.410
PhalangesOutlinesCorrect	2	1800	858	80	**0.239**	**0.239**	0.397	0.391	0.302	0.291	0.288	0.272	0.264
Phoneme	39	214	1896	1024	0.891	0.773	0.798	0.786	0.812	0.797	0.759	0.772	**0.730**
ProximalPhalanxOutlineAgeGroup	3	400	205	80	0.215	0.215	0.207	0.194	0.185	0.192	0.178	**0.152**	0.170
ProximalPhalanxOutlineCorrect	2	600	291	80	0.192	0.210	0.280	0.267	0.167	0.165	0.174	0.181	**0.144**
ProximalPhalanxTW	6	205	400	80	0.292	0.263	0.303	0.307	0.257	0.259	0.257	0.300	**0.233**
RefrigerationDevices	3	375	375	720	0.605	0.560	0.533	0.523	0.526	0.494	0.478	**0.415**	0.417
ScreenType	3	375	375	720	0.640	0.589	0.506	**0.486**	0.678	0.669	0.572	0.506	0.499
ShapeletSim	2	20	180	500	0.461	0.300	0.189	0.194	0.067	0.051	**0.017**	0.161	0.047
ShapesAll	60	600	600	512	0.248	**0.198**	0.715	0.595	0.585	0.485	0.448	0.332	0.313
SmallKitchenAppliances	3	375	375	720	0.659	0.328	0.239	0.212	0.282	0.261	**0.205**	**0.205**	0.206
Strawberry	2	370	613	235	**0.062**	**0.062**	0.217	0.205	0.117	0.113	0.097	0.099	0.094
ToeSegmentation1	2	40	228	277	0.320	**0.250**	0.354	0.339	0.289	0.301	0.296	0.261	0.259
ToeSegmentation2	2	36	130	343	0.192	**0.092**	0.185	0.185	0.205	0.182	0.185	0.218	0.185
UWaveGestureLibraryAll	8	896	3582	945	0.052	**0.034**	0.551	0.498	0.516	0.482	0.386	0.278	0.265
Wine	2	57	54	234	**0.389**	**0.389**	0.493	0.404	0.530	0.519	0.448	0.556	0.548
WordSynonyms	25	267	638	270	0.382	**0.252**	0.794	0.770	0.662	0.610	0.578	0.559	0.571
Worms	5	77	181	900	0.635	0.586	0.492	0.470	0.414	0.409	**0.387**	0.448	0.409
WormsTwoClass	2	77	181	900	0.414	0.414	0.328	0.306	0.242	0.248	0.243	0.239	**0.233**
				Comparison versus	G	G	B	D	D	E	F	G	E
				Number of more accurate datasets	26	23	32	30	29	27	19	29	30
				Wilcoxon test p-value	0.01	0.1638	9.48e-7	3.09e-3	9.56e-5	5.01e-3	0.8623	1.72e-4	8.74e-4

6.4.2.1 *Choosing Graph Features*

We first investigate which graph features are helpful for extracting distinguishable information from time series. Specifically, we transform time series into UVGs consisting of both VG and HVG representations and try to extract MPDs as well as other features (density, assortativity and degree statistics) from these graphs. We are interested in finding out whether MPDs are sufficient for TSC and if extracting other statistic features from graphs are necessary. Next, we take advantage of XGBoost classifier to train on different feature sets and classify the test datasets. Columns A, B, C and D in Table 6.4 presents the classification error rates using HVG with only MPDs, HVG with MPDs and other graph features, VG with only MPDs and VG with all features. Comparison on the bottom of the table shows that including features such as density and assortativity coefficient indeed helps improving classification accuracy. For HVGs, including features other than MPDs increases classification accuracy in 32 datasets, while for VGs 29 datasets saw accuracy improvements. A Wilcoxon signed rank test with p-values *9.48e-7* and *9.56e-5* suggests that such improvement is indeed significant (both p-values < 0.05). Figure 6.6 shows the classification results in the form of scatter plots, where each point represent a dataset. Such results along with the Wilcoxon test indeed suggest that our Heuristic (a) is valid.

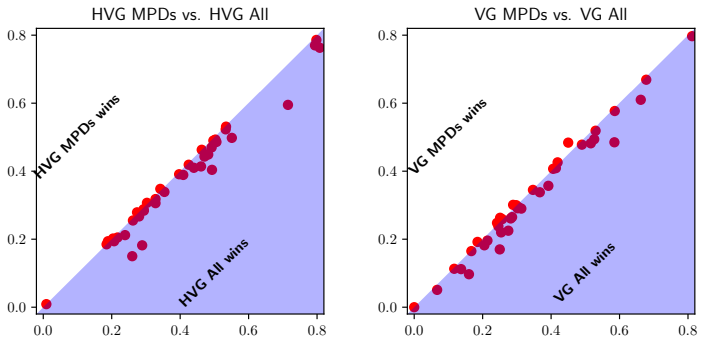

Figure 6.6: Comparison of classification error rates: using MPDs with or without other graph features.

6.4.2.2 *VG and HVG*

Next, it is necessary to show that graph features extracted from both VGs and HVGs are important. Recall that, intuitively, VGs are helpful for capturing global features while HVG can help locating local features. We separately test the distinguishing power of VGs and HVGs for time series in order to make sure that both can lead to satisfactory classification results. Furthermore, we conduct experiments to investigate if combining VGs and HVGs can better capture both global and local features for time series and thus lead to more accurate classification.

Figure 6.7 illustrates the comparison of classification accuracy with VG and HVG features as well as combining both VG and HVG. The first scatter plot

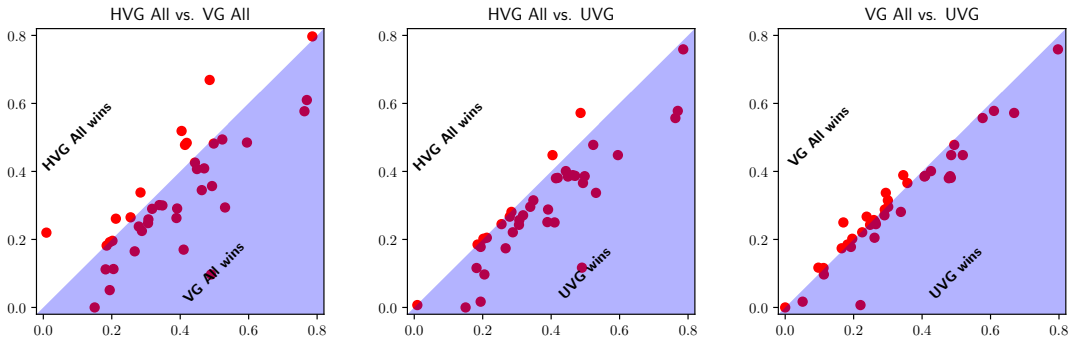

Figure 6.7: Comparison of classification error rates: using HVGs, VGs or combining two together (denoted as UVG here).

shows that for majority datasets, VG features can yield more accurate classification performance. However, it is still worth mentioning that HVG features can also outperform VG features in some specific datasets, possibly due to the reason that local features are more influential in such datasets. Moreover, it is obvious from the other two scatter plots in Figure 6.7 that combining both VG features and HVG features can greatly boost the classification accuracy. This is probably due to the excellent feature selection capability of XGBoost, so that the classifier is able to find out which features are more important during the training process. In addition, as shown in Table 6.4, using VGs outperforms HVGs in 30 datasets with a p-value of *3.09e-3*, suggesting VGs are capable of capturing more characteristics in time series. Finally, combining features from both VGs and HVGs yields more accurate results in 27 datasets with a p-value of *5.01e-3*, indicating significant improvement when combining two different types of graphs. As a result, the validity of our Heuristic (b) can be confirmed.

6.4.2.3 *UVG, AMVG and MVG*

Since we have demonstrated that taking advantage of both VG and HVG features can improve classification accuracy for UVG representations, now we can investigate whether Heuristic (c) holds, *i.e.*, if multiscale representations can help achieving even more accurate results. To visually inspect which representation suites best for TSC, we further draw scatter plots of the accuracy results in Figure 6.8. It is then obvious that AMVG and UVG (scatter plot on top) lead to similar classification accuracy, which suggests that AMVG can be good approximations for original time series data. Furthermore, the two scatter plots in the bottom of Figure 6.8 indeed confirm that MVG representations result in better classification performance, since almost all dots representing results from different datasets fall on the side of MVG.

From Table 6.4, we can see that multiscale approximations outperforms uniscale time series in 19 datasets, with a Wilcoxon p-value of $0.8623 > 0.05$. As a result, AMVG representations are not statistically significantly different than UVGs. On the other hand, MVGs outperform AMVG in 29 datasets with a p-value of *1.72e-4*

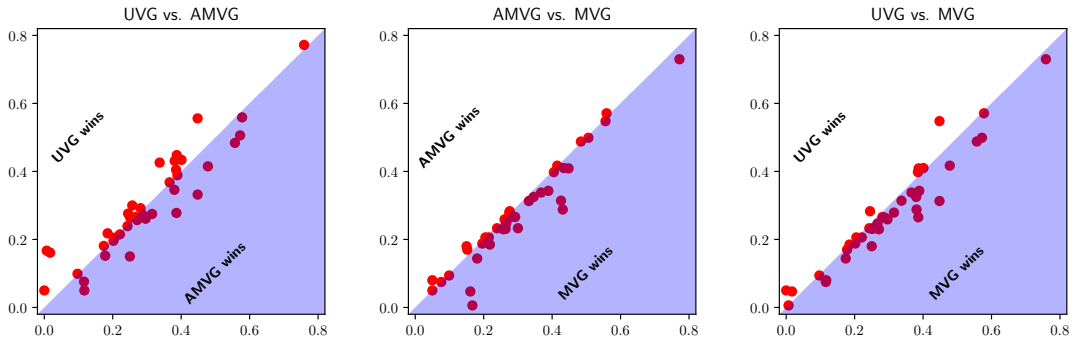

Figure 6.8: Comparison of UVG, AMVG and MVG's error rates.

and MVGs are more accurate than UVGs in 30 datasets with a p-value of *8.74e-4*, suggesting that MVG representations indeed contribute significantly towards a more accurate feature extraction and TSC process. As a result, our Heuristic (c) is valid.

6.4.2.4 *Summary of Heuristic Validation*

Overall, all our heuristics are supported by experiment results and each heuristic contributes significantly to a more accurate TSC process. Table 6.4 also shows the comparison between our approach and Euclidean distance- as well as DTW-based nearest neighbor classification. Our approach outperforms 1NN-Euclidean in 26 datasets with a Wilcoxon p-value of *0.01*, suggesting MVG features with XGBoost is significantly more accurate than 1NN-Euclidean. Moreover, MVG appears to be in par with 1NN-DTW in terms of classification accuracy since 23 datasets are in favor of MVG with a p-value of *0.1638*. Next, we will try to build a more robust and accurate classifier using stacked generalization.

6.4.3 *Stacked Generalization*

Previously we have solely taken advantage of XGBoost for classifying time series with features extracted from graphs. However, it can also be interesting to investigate if other classifiers can achieve similar results. Besides, although modern classifiers such as XGBoost and RF are capable of efficiently conducting feature selection during training, we can still conduct feature selection processes and feed such *a prior* information to classifiers in order to achieve better classification accuracy. To that end, we propose feeding different sets of features to a collection of classifiers and then create a meta-classifier using stacked generalization (*a.k.a* stacking or blending) [Wolpert, 1992]. This meta-classifier will then hopefully generate better final predictions. In fact, a variant of this technique has led to winning the Netflix Prize with a reward of one million dollars [Sill et al., 2009].

Before building a meta-classifier with stacked generalization, we first make sure that features we previously extracted are suitable inputs for different classifiers such as RF and SVM. Generally, tree-based classifiers are not so sensitive to monotonic transformations of individual features. As a result, it is often not required to have features scaled to similar magnitudes for RF and XGBoost. However, since SVM's kernel functions (in an Euclidean space) are usually sensitive to different feature magnitudes, we use Min-Max scaling to transform each feature into range of zero and one. After that, we compare the classification performance of these three classifiers.

In order to evaluate the significance of the differences a generic classifier can incur, we take advantage of the Nemenyi test [Dunn, 1964], which is a post-hoc test aiming for finding whether groups of data differ after a statistical test of multiple comparisons. This test can be illustrated by means of a critical difference diagram, where average ranks of all approaches are presented. Specifically, the location of vertical lines indicate the average ranking of an approach and bold lines (insignificance lines) indicate groups of approaches that are not significantly different. In this case, for one approach to be considered significantly better than another, its overall ranking has to be at least 0.5307 higher than its competitor. As shown in Figure 6.9, XGBoost performs slightly better than RF in general, and both are significantly more accurate than SVM. Such results are not surprising, since [Fernández-Delgado et al., 2014] have empirically tested hundreds of classifiers and concluded that RF produces the most accurate classification results. This study was conducted before the initial release of XGBoost, and the recent adoption momentum of XGBoost indeed suggests that XGBoost is great for yielding accurate classification results.

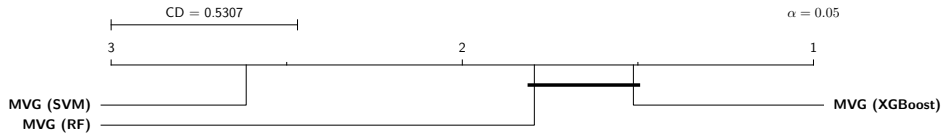

Figure 6.9: Critical difference diagram comparison of RF, SVM and XGBoost.

Now that generic classifiers can be used for graph features extracted from time series, we then set to investigate whether stacking can help further increase classification accuracy. We first stack top performing classifiers in each family before blending classifiers from different families. Specifically, we first select the top five most accurate classifiers from RF, SVM and XGBoost through cross validation. Then these five classifiers are stacked to produce a meta-classifier, which is used for producing final predictions. Finally, when stacking classifiers of different families, five classifiers from each family have been selected, thus the meta-classifier has been trained with 15 classifiers. Our algorithm is described in Algorithm 6.2.

Our experiments show that, for RF and SVM, stacking their top performing classifiers indeed increases final classification. In the case of XGBoost, however, stacking its most accurate classifiers does not seem to significantly increase clas-

Algorithm 6.2 Algorithm for creating an ensemble classifiers using stacked generalization.

Input: Training dataset $\mathbb{D} = \{\mathbb{X}_i, y_i\}_{i=1}^{m} (\mathbb{X}_i \in \mathbb{R}^n, y_i \in \mathbb{Z})$
Base classifiers $\mathbb{H}$ (with different hyper-parameters)
Output: An stacked ensemble E

procedure BUILDSTACKINGENSEMBLE($\mathbb{D}, \mathbb{H}$)
  $\mathbb{S} \leftarrow$ CreateStratifiedKFolds($\mathbb{D}$, cv=3) ▷ 3-fold CV
  $\mathbb{E} \leftarrow \varnothing$ ▷ Best performing base estimators
  for all $h \in \mathbb{H}$ **do**
    $H \leftarrow \varnothing$
    for all $S = \{\mathbb{D}_{train}, \mathbb{D}_{validation}\} \in \mathbb{S}$ **do**
      TrainClassifier($h, \mathbb{D}_{train}$)
      $\hat{y} \leftarrow$Predict($h, \mathbb{X}_{validation}$)
      score $\leftarrow -\log P(\hat{y}|y_{validation})$
      $H \leftarrow H \cup \{h, score\}$
    $H \leftarrow \arg\,sort(H)$ ▷ Sort estimators by score
    $\mathbb{E} \leftarrow \mathbb{E} \cup$ slice(H, k) ▷ Select top-k estimators
  $W \leftarrow$ ComputeEstimatorWeights($\mathbb{E}$) ▷ with logistic regression
  $E \leftarrow \sum_{i=1}^{|\mathbb{E}|} W_i \mathbb{E}_i$
  return E

sification accuracy, since the classification results of stacked generalization is on par with those with a single most accurate classification during cross validation. It possibly indicates that XGBoost has already very good generalization capabilities. Finally, stacking most accurate classifiers from three different families can help achieving better classification accuracy than single best XGBoost classifier in most datasets. As a result, stacked generalization can indeed be helpful for further improving the performance of MVG.

Figure 6.10 demonstrates how stacked generalization can help boosting classification accuracy. It is straightforward that stacking XGBoost and SVM produces similar classification accuracy, while stacking top performers from all three families can be significantly more accurate than using a single family. As a result, we are confident that staked generalization is favorable for more accurate TSC.

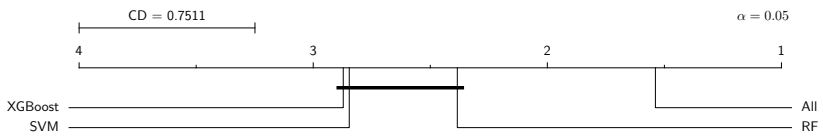

Figure 6.10: Critical difference diagram comparison of stacking single family of classifiers versus all families of classifiers.

Table 6.5: Classification error rates compared with five benchmark approaches and running time statistics (in seconds).

Dataset	# Cls.	#Train	#Test	Dim.	Classification Error Rate						MVG Runtime			FS Runtime
					1NN-ED	1NN-DTW	LS	FS	SAX-VSM	MVG	FE	Clf.	$\sum$	
ArrowHead	3	36	175	251	0.200	0.297	**0.154**	0.406	0.211	0.371	8	29	37	**30**
BeetleFly	2	20	20	512	0.250	0.300	0.200	0.300	0.100	**0.050**	2	21	**23**	54
BirdChicken	2	20	20	512	0.450	0.250	0.200	0.250	**0.000**	**0.000**	4	22	**26**	38
Computers	2	250	250	720	0.424	0.300	0.416	0.500	0.380	**0.252**	22	51	**73**	1293
DistalPhalanxOutlineAgeGroup	3	400	139	80	0.374	0.230	0.281	0.345	**0.158**	0.254	11	86	97	**45**
DistalPhalanxOutlineCorrect	2	600	276	80	0.283	0.283	**0.221**	0.250	0.272	0.281	19	93	112	**101**
DistalPhalanxTW	6	400	139	80	**0.367**	0.410	0.374	0.374	0.396	0.381	12	204	216	**59**
ECG5000	5	500	4500	140	0.075	0.076	**0.068**	0.077	0.090	0.069	162	190	352	**170**
Earthquakes	2	322	139	512	0.288	0.281	0.259	0.295	**0.252**	**0.252**	22	78	**100**	3689
ElectricDevices	7	8926	7711	96	0.448	0.398	0.413	0.421	**0.295**	0.332	406	6344	6750	**3558**
FordA	2	3601	1320	500	0.335	0.445	0.043	0.213	0.173	**0.014**	207	410	**617**	44832
FordB	2	3636	810	500	0.394	0.380	**0.083**	0.272	0.249	0.333	183	534	**717**	45874
Ham	2	109	105	431	0.400	0.533	0.333	0.352	**0.190**	0.343	8	32	**40**	670
HandOutlines	2	1000	370	2709	0.138	0.119	0.519	0.189	**0.092**	0.200	4431	182	**4613**	179745
Herring	2	64	64	512	0.484	0.469	0.375	0.469	0.375	**0.344**	19	30	**49**	286
InsectWingbeatSound	11	220	1980	256	0.438	0.645	**0.394**	0.511	0.453	0.459	85	130	**215**	705
LargeKitchenAppliances	3	375	375	720	0.507	0.205	0.299	0.440	**0.123**	0.288	32	89	**121**	7301
Meat	3	60	60	448	0.150	0.067	0.100	0.067	0.067	**0.050**	10	31	**41**	120
MiddlePhalanxOutlineAgeGroup	3	400	154	80	0.481	0.500	**0.429**	0.455	0.455	0.435	9	88	97	**50**
MiddlePhalanxOutlineCorrect	2	600	291	80	0.234	0.302	**0.220**	0.271	0.323	0.289	15	87	102	**80**
MiddlePhalanxTW	6	399	154	80	0.487	0.494	0.494	**0.468**	0.513	0.481	9	177	186	**62**
PhalangesOutlinesCorrect	2	1800	858	80	0.239	0.272	**0.235**	0.256	0.290	0.248	48	209	**257**	332
Phoneme	39	214	1896	1024	0.891	0.772	0.782	0.826	0.895	**0.692**	134	1419	**1553**	25604
ProximalPhalanxOutlineAgeGroup	3	400	205	80	0.215	0.195	**0.166**	0.220	0.176	**0.166**	11	70	81	**41**
ProximalPhalanxOutlineCorrect	2	600	291	80	0.192	0.216	0.151	0.196	0.172	**0.144**	18	80	98	**80**
ProximalPhalanxTW	6	400	205	80	0.293	0.239	0.224	0.298	0.390	**0.220**	12	160	172	**49**
RefrigerationDevices	3	375	375	720	0.605	0.536	0.485	0.667	**0.347**	0.421	37	77	**114**	12798
ScreenType	3	375	375	720	0.640	0.603	0.571	0.587	0.488	**0.480**	35	89	**124**	8473
ShapeletSim	2	20	180	500	0.461	0.350	0.050	**0.000**	0.283	**0.000**	6	22	**28**	76
ShapesAll	60	600	600	512	0.248	**0.232**	**0.232**	0.420	0.302	0.255	168	1833	**2001**	7939
SmallKitchenAppliances	3	375	375	720	0.656	0.357	0.336	0.667	0.421	**0.208**	30	75	**105**	5667
Strawberry	2	613	370	235	0.054	0.059	0.089	0.097	**0.043**	0.076	29	75	**104**	314
ToeSegmentation1	2	40	228	277	0.320	0.228	0.066	**0.044**	0.070	0.197	8	26	34	**30**
ToeSegmentation2	2	36	130	343	0.192	0.162	**0.085**	0.308	0.138	0.185	6	22	**28**	38
UWaveGestureLibraryAll	8	896	3582	945	0.052	0.108	**0.047**	0.211	0.201	0.258	470	343	**813**	32047
Wine	2	57	54	234	0.389	0.426	0.500	0.241	**0.037**	0.519	4	27	31	**22**
WordSynonyms	25	267	638	270	0.382	**0.351**	0.393	0.569	0.509	0.522	39	1017	1056	**907**
Worms	5	181	77	900	0.545	0.416	0.390	0.351	0.442	**0.182**	26	98	**124**	6852
WormsTwoClass	2	181	77	900	0.390	0.377	0.273	0.273	0.286	**0.130**	27	45	**72**	5044
Number of best (including ties)					1	2	12	3	10	16			24	15
Wilcoxon test p-value					0.0023	0.0044	0.3421	0.0005	0.5767	–	Total time →		21379	395075

6.4.4 *Accuracy Benchmarking*

Finally, we compare our results with the state-of-the-art and relevant approaches. Table 6.5 presents the classification error rates as well as the running time statistics. Specifically, datasets in this experiment are from the UEA & UCR Time Series Classification Repository thanks to the many benchmarking results that are publicly available. Note that although names of the datasets used in this repository are exactly the same compared to our previous experiments, the similarity of their contents is not guaranteed. In fact, the training and testing datasets may have been swapped for a number of datasets. An obvious example is the `FordA` dataset, where in this experiment the training dataset and testing set are of size 1320 and 3601 re-

spectively, while in our previous experiments the training set has 3601 samples and the testing test has 1320.

We compare our method MVG with five state-of-the-art approaches: two distance-based global similarity matching algorithms including Euclidean- and DTW-based nearest neighbor classification (1NN-ED and 1NN-DTW), and three local pattern matching algorithms including Learning Shapelets (LS) [Grabocka et al., 2014], Fast Shapelets (FS) [Rakthanmanon and Keogh, 2013] and SAX-VSM [Senin and Malinchik, 2013]. Among all five approaches, LS is recognized as the most accurate classifier [Wang et al., 2016b] by the research community. However, LS is also known for its high computation complexity. We first compare the classification accuracy of different approaches in this section and then investigate MVG's efficiency later in section 6.4.5.

Viewing the error rate comparison columns in Table 6.5, it is obvious that MVG is the most accurate classifier with 16 winning datasets. LS then follows MVG with 12 winning datasets. A Wilcoxon test between MVG and LS yields a p-value of $0.3421 > 0.05$, suggesting that MVG should not be considered significantly better than LS. SAX-VSM is ranked third with 10 winning cases. The Wilcoxon test again suggest that such difference is not statistically significant. However, MVG is indeed significantly more accurate than FS, 1NN-DTW and 1NN-ED. Moreover, scatter plots in Figure 6.11 shows that most of the points in each comparison are located away from the diagonal line, suggesting that MVG is indeed a very different approach when compared to the state-of-the-art. Having confirmed MVG's excellent accuracy, we continue to investigate its runtime efficiency.

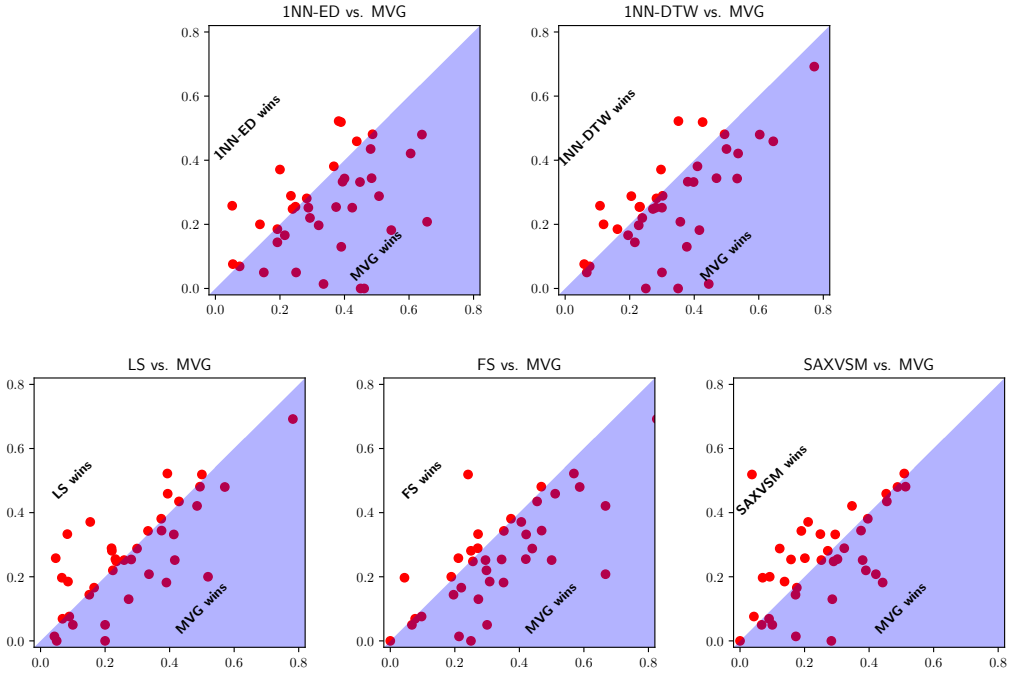

Figure 6.11: Comparison of classification accuracy with five state-of-the-art approaches in the form of scatter plots.

6.4.5 *Efficiency*

The pipeline of MVG consists of a feature extraction phase and a train-validate-test process. During the feature extraction process, time series are firstly transformed in to VGs and then features are extracted from such graphs. With optimization from [Afshani et al., 2017], transforming time series of length n into VGs has a computation complexity of $O(n \log^2(n))$ and it can be effectively solved within $O(\log^2(n))$ time when taking full advantage of parallelization. Extracting motif features from VGs may be potentially expensive, fortunately PGD provides a fully parallel way of counting small motifs. For instance, counting graph cliques of size 4 – one of the most time consuming tasks in PGD – has a computation complexity of $O(m \cdot \Delta \cdot T_{max})$, where m is the number of 4-motifs (*i.e.*, 11), Δ is the maximum degree and T_{max} is the maximum number of triangles incident to an edge and $T_{max} \ll \Delta$. PGD is hundreds of times faster than other motif counting algorithms: counting motifs from a graph with more than 26,000 vertices can take only 0.01 seconds on a twelve-core commodity CPU [Ahmed et al., 2015]. Since other statistic features such as density, k-core, assortativity and degree statistics extracted from time series VGs are intentionally chosen to be simple metrics, collecting these features has a time complexity of $O(\max(|V|, |E|))$. Since we use a multiscale graph representation, ultimately graph generation has a monothread complexity of $O(n \log^3(n))$ and in parallel an $O(\log^3(n))$ time complexity. As a result, feature extraction per graph has a computation complexity of $O(\max(|V|, |E|) + m \cdot \Delta \cdot T_{max})$. The classification process leverages state-of-the-art classifiers that are widely used and well-optimized. Overall, the efficiency of MVG may be best illustrated when we compare our method against our benchmarking approaches.

Previous research [Wang et al., 2016b] has shown that LS is extremely time consuming, and FS as an approximated approach can be 100X faster than LS. As a result, FS will be a good and strong baseline to which the running time of our approach can be compared. Thus we record the running time of FS and MVG and compare their efficiency. These experiments are conducted on a computer with Intel i7-4980HQ quad-core CPU clocked at 2.80GHz, 16GB memory and solid-state drive suitable for fast I/O. We use the FS implementation by its original authors [Rakthanmanon and Keogh, 2013] with default parameters. Columns located on the right side of Table 6.5 shows the running time per dataset for MVG and FS. For MVG, running time for feature extraction and training are recorded separately and then summed. Overall, MVG completes faster than FS for 24 datasets. The total running time of FS for all 39 datasets amounts to 395075 seconds, which is more than 18 times that of MVG. A scatter plot of the running time is provided in Figure 6.12, obviously MVG can be up to 100X faster than FS, suggesting that it is indeed an efficient TSC approach. In addition, we note that FS appears to be time consuming with large datasets with time series of high dimensionality, while the running time of MVG remains reasonable as datasets grow larger. Furthermore, since we have experimented only on a quad-core commodity computer that is not really powerful in terms of parallel processing, the efficiency of MVG can be easily

boosted by adding more computing cores. Overall, MVG can indeed be considered an efficient TSC approach.

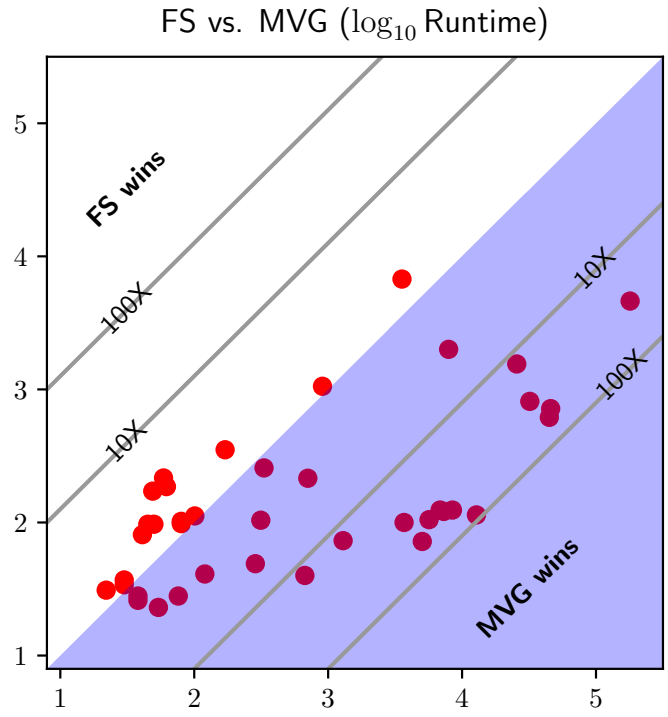

Figure 6.12: Runtime comparison between FS and MVG.

6.4.6 *Case Studies*

Although accuracy and efficiency are very important measurements for TSC approaches, it is also desirable for TSC methods to have comprehensible classification processes, so that users may gain extra insights into his/her time series data. Popular approaches based on shapelets and subsequences may have a natural advantage in classification comprehensibility. However, since features extracted in MVG are solely statistical, it can be difficult to locate exactly where a distinguishing subsequence lies in the original time series. However, for MVG it is still possible to understand which features have contributed most to correct classification. For instance, when feeding features to train an XGBoost classifier, it will assign weights to all the features. After ranking these features, we can also gain insights into the data.

For instance, Figure 6.13 plots all the test samples of three different classes from the `Meat` dataset. For humans it can be extremely difficult to spot the differences across different classes of data. With the help of MVG, extra insights can be gained. Figure 6.14 illustrates a scatter matrix plot for the test dataset of `Meat` showing ten most important features learned by the classifier. Out of ten features, only four

are features from VGs, which are created from the original series (T_0), indicating that mulitiscale representations are indeed helpful.Furthermore, it appears that most highly ranked graph features for this dataset are MPDs and assortativity, confirming that statistics other than MPDs are helpful. Finally, visually from the kernel density estimation it is obvious that certain features – *e.g.*, T_5 VG $P(M4_4)$ – are already descriptive, again confirming that it is feasible for MVG to present visually comprehensible cues regarding its classification decisions.

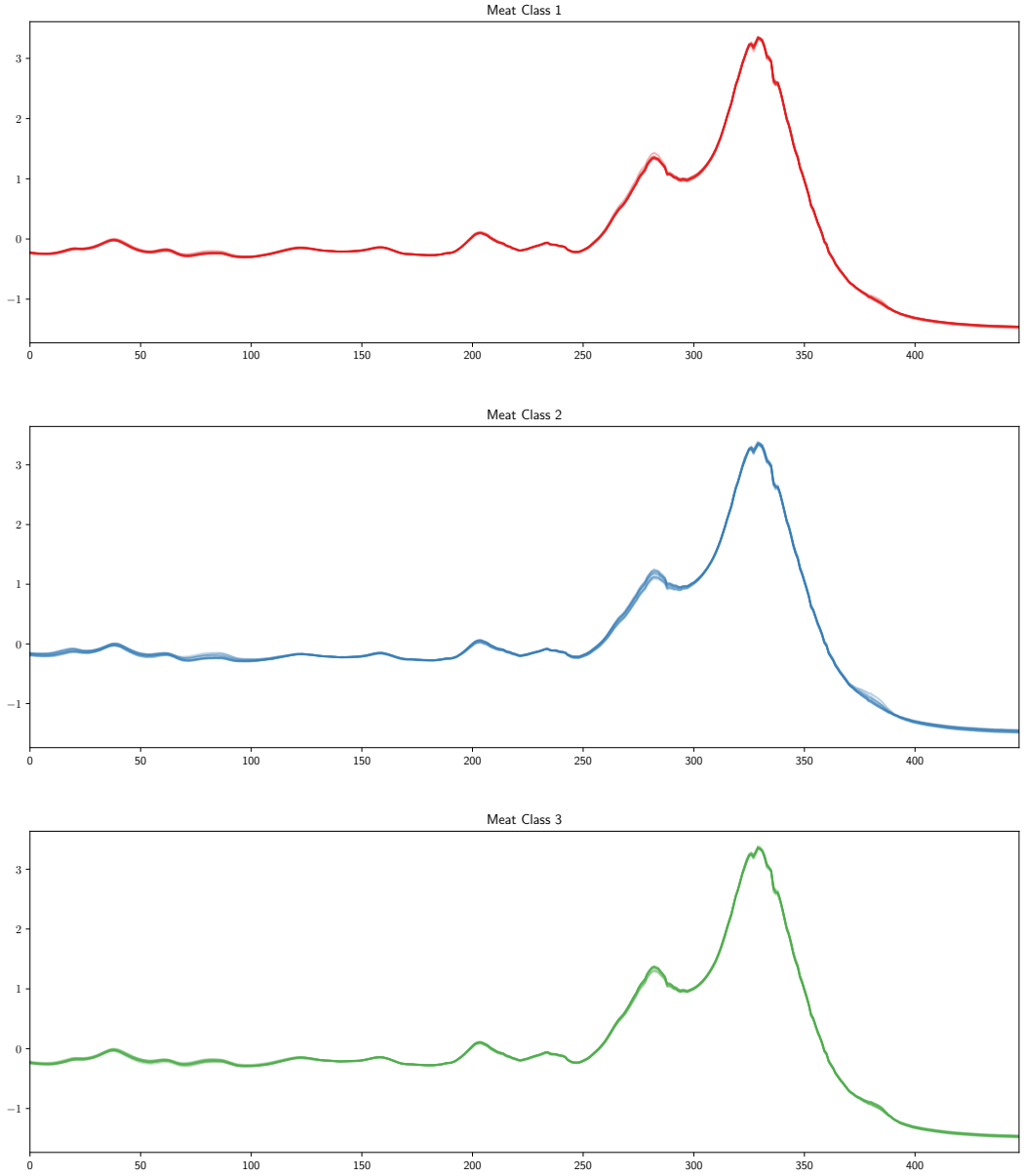

Figure 6.13: Samples from the `Meat` test dataset.

Figure 6.15 shows another example, where all the test samples of three different classes from the `Meat` dataset are plotted. Unlike Figure mvg-fig:meat where all different classes seems to be similar fro human eyes, it is the opposite in this case: there does not appear to be a clear pattern within each class. Again, with the help

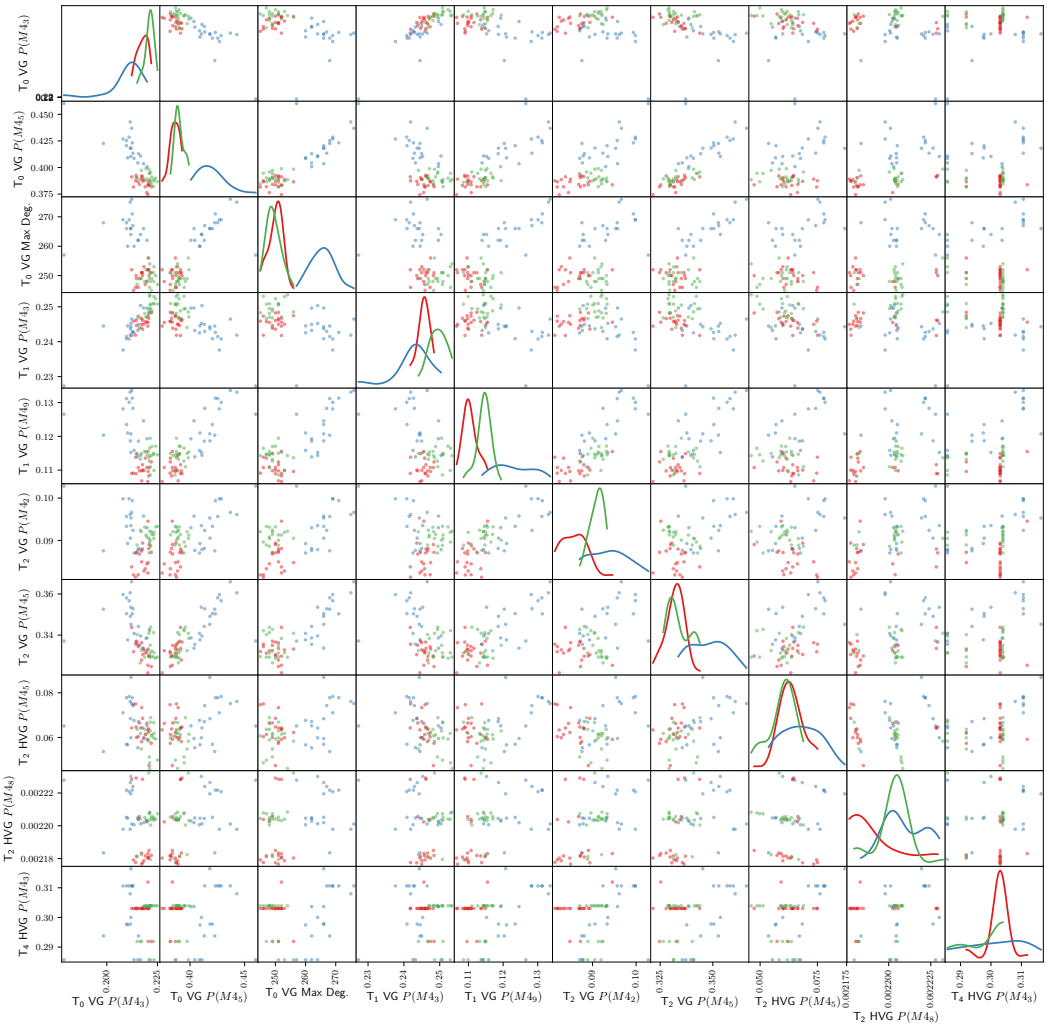

Figure 6.14: Scatter matrix of ten most important features for `Meat`'s test dataset. Different point colors indicate different classes and the diagonal shows the Gaussian kernel density estimation for each feature.

of MVG, we investigate if extra insights are readily available. Figure 6.16 illustrates a scatter matrix plot for the test dataset of `Worms` showing ten most important features learned by the classifier. Out of ten features, seven are features from VGs, which are created from the original series (T_0) as well as multiscale approximations (T_1 and T_2). HVGs features from dimensionality-reduced approximations (T_2 and T_4) are also present. Moreover, it appears that most highly ranked graph features for this dataset are MPDs and max degree, suggesting that statistics other than MPDs are indeed helpful. Finally, visually from the kernel density estimation it is obvious that some features alone – *e.g.*, T_0 VG $P(M4_5)$, T_0 VG Max Degree and T_2 VG $P(M4_2)$ – can already provide good classification guidelines, suggesting that it is feasible for MVG to present visually comprehensible cues regarding its classification decisions.

To save space, we show more examples comparing original time series and important MVG features on project website. Interested readers are recommended to visit http://daoyuan.li/mvg/ for more illustrations.

6.4.7 *Discussions*

Due to specific characteristics of VGs, obviously MVG may not be suitable for every TSC scenarios. For instance, VGs are agnostic of affine transformations in time series. That is, in applications where the absolute oscillation is more important, MVG is less likely to detect such characteristics. Furthermore, when dealing with non-stationary time series data where trends are very frequently found, MVG works best when trends are not the deciding factor since VGs may not be able to capture long-term monolithic trends. Since many graph features have been taken into account in MVG, we believe it can be robust in practical applications.

In addition to limitations inherited from VGs, our approach is more suitable for applications where the size of training datasets is larger, so that classification models can generalize better. In fact, when we review the classification accuracy of MVG, it seems that a majority of its winning datasets have either long time series or large training datasets. This is perhaps innate to the nature of statistics: sample size has to be sufficiently large to make accurate estimations. Based on running time comparison, MVG also scales well with large datasets, which can be an important advantage in the era of big data.

6.5 RELATED WORK

TSC is a major task in time series mining thanks to its wide application scenarios. As a consequence, there are a plethora of classification algorithms for TSC. Classical TSC approaches involve utilizing 1NN classification together with similarity measures specific to time series data, *e.g.*, DTW [Berndt and Clifford, 1994] and its variants with lower bounding [Ratanamahatana and Keogh, 2005] and early abandoning techniques [Rakthanmanon et al., 2012], Time Warp Edit Distance (TWED) [Marteau, 2009] and Minimum Jump Cost (MJC) [Serra and Arcos, 2012].

Another line of research transforms time series into texts and resort to text classification algorithms. For example, inspired by the well known *bag-of-words* approach, [Lin et al., 2012] proposes the Bag-of-Patterns approach for TSC. SAX-VSM [Senin and Malinchik, 2013] also takes advantage of bag-of-words approach and builds term frequency-inverse document frequency (*TF-IDF*) vectors in its training phase. It defines a similarity measure of two vectors (that are constructed from original series) based on their inner product. Both Bag-of-Patterns and SAX-VSM relies on SAX [Lin et al., 2007] for transforming time series into texts. A more recent work, Representative Pattern Mining (RPM) [Wang et al., 2016b], also tries to classify time series by means of finding the most representative SAX-symbolized subsequences. [Schäfer, 2015] invents another symbolic representation based on Fourier transform and proposes a bag-of-patterns approach named BOSS ensemble based on this symbolic representation method.

Recently, shapelet-based approaches are gaining popularity among the research community. A shapelet [Ye and Keogh, 2009] is a single subsequence in time series that is representative of its class. However, these approaches [Grabocka et al., 2014; Hills et al., 2014] are known for their high computation complexity. As we have demonstrated in this chapter, FS [Rakthanmanon and Keogh, 2013] as an approximated approach can also take long time to run. High computation complexity is also present for TSC approaches using deep neural networks [Yang et al., 2015]. Finally, [Bagnall et al., 2015] introduce an ensemble algorithm named Collective of Transformation-based Ensembles (COTE) – an ensemble of 35 different classifiers – that has shown to be very accurate but has a time complexity of $O(m^4n^2)$, where m is the dimensionality of time series and n is the size of training dataset.

Although the concept of time series VGs has appeared for almost a decade, using it for TSC has just started picking up. Machine learning approaches taking advantage of VG and its variants are generally using artificially generated data [Xu et al., 2008] or EEG signals. For instance, [Supriya et al., 2016] convert EEG signals into weighted VGs for the detection of epilepsy. Finally, graph kernel methods [Kondor and Pan, 2016] can be used for evaluating graph similarity, which may potentially be used for TSC as well.

6.6 CONCLUSIONS AND FUTURE WORK

This chapter proposes a graph-based multiscale time series representation named MVG and a feature extraction method for TSC. MVG transforms time series into a collection of visibility graphs of various scales and feature extraction are conducted by investigating statistical graph features such as probability distributions of small motifs, assortativity as well as degree statistics. After a large scale evaluation with open datasets and comparison with a number of state-of-the-art TSC algorithms, our proposed approach appears to be both accurate and efficient: it can be more accurate than LS and up to ~100X faster than FS.

In the future, we plan to further investigate other useful and efficient graph features – such as degree distribution entropy, centrality, bipartivity, *etc.* [Costa et al., 2007] – for MVG in order to further improve its accuracy. Currently we have only evaluated MVG with univariate time series data, we are also excited to investigate the possibility of adopting MVG for multivariate TSC.

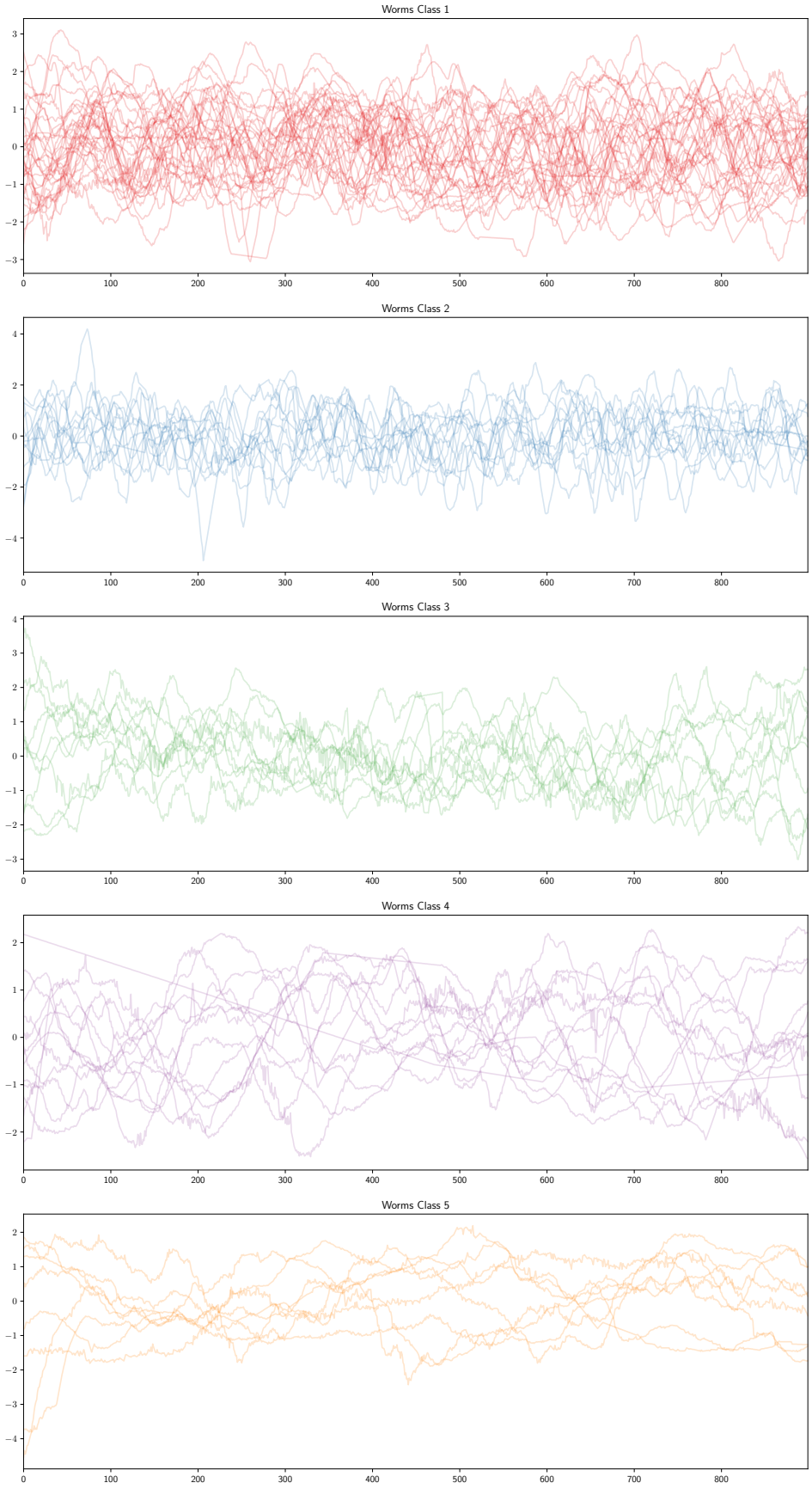

Figure 6.15: Samples from the `Worms`' test dataset.

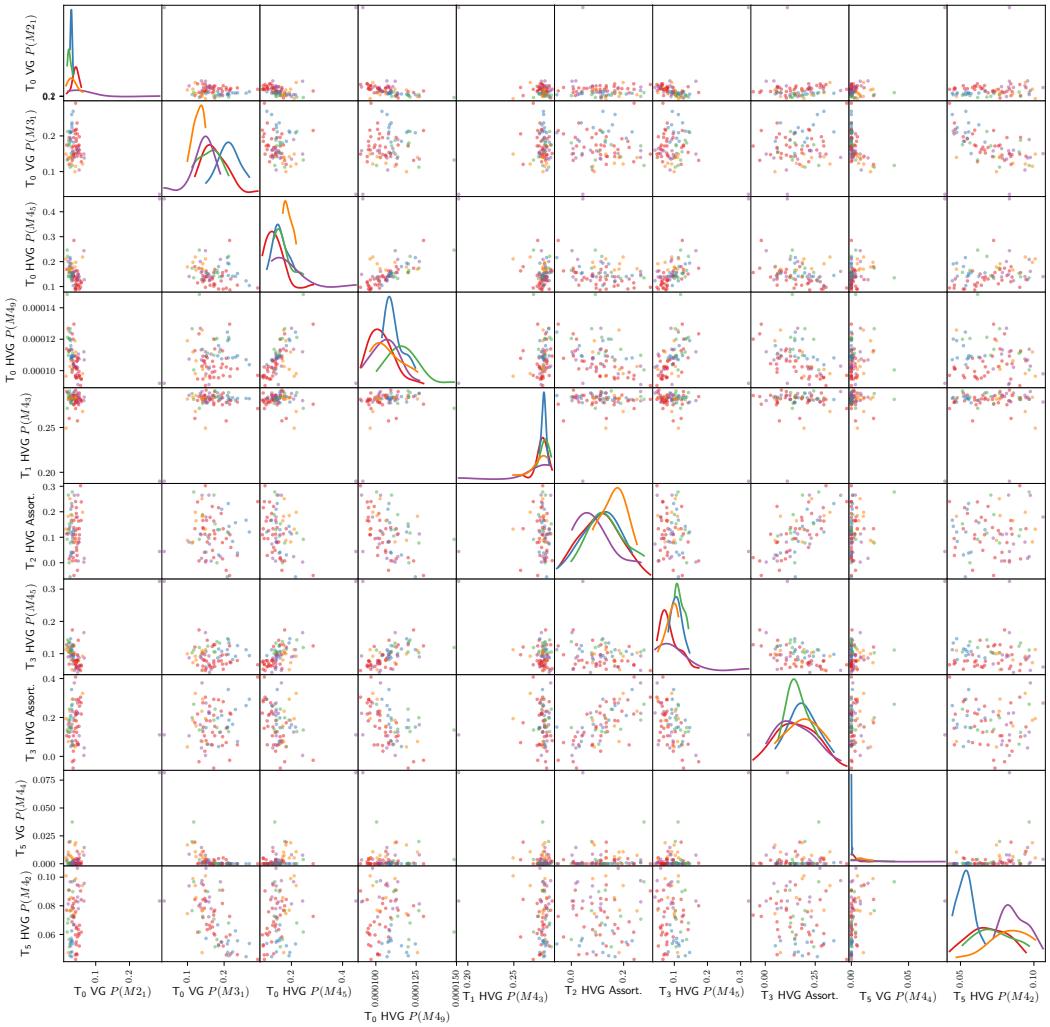

Figure 6.16: Scatter matrix of ten most important features for `Worms`' test dataset. Different point colors indicate different classes and the diagonal shows the Gaussian kernel density estimation for each feature.

Part IV

APPLICATIONS

7

PROFILING HOUSEHOLD APPLIANCES

> In theory, there is no difference between theory and practice. But, in practice, there is.
>
> *Walter J. Savitch*
> *Pascal: An Introduction to the Art and Science of Programming*

Outline
In this chapter, we apply DSCo in the domain of Non-Intrusive Load Monitoring (NILM). Specifically, we profile the electricity consumption of different household appliances with language models and experiment against real-world datasets.

7.1 INTRODUCTION

Citizens in modern societies are increasingly aware of the urgent need to make better use of energy resources and move to more sustainable and greener development paths. The are some signs that the rate of improvement in energy efficiency has been increasing slightly in the last few years, but much remains to be done, especially to turn consumers' awareness into concrete and tangible actions. Typically, while one of the major sources of electric consumption is household appliances, studies show that inhabitant behaviors have not yet fully shifted [Morales Pedraza, 2015]. The introduction of smart meters in households is predicted as a defining factor to change such behaviors and incur reduction of energy usage [Darby, 2001], as they are promising means for gaining insights on energy consumption by mining electricity usage patterns.

Smart meters measure energy consumption and provide online readings on electricity usage from households, thus allowing a fine-grained visualization of real-time consumption. This data can be leveraged to detect malfunctioning appliances or discover abnormal situations once usage patterns of appliances have been identified. In order to avoid reluctance and involve the general public in the usage pattern identification task, the research community is investigating automated processes (e.g., to prevent inhabitants being requested to manually indicate which appliance is generating a given consumption signal). In this regard, household appliance classification has become increasingly popular among the research community. The development of Non-Intrusive Load Monitoring (NILM)

techniques [Hart, 1992] has made it possible, in a non-intrusive manner, to disaggregate households' single-point energy consumption measurements into individual devices' consumption. Analyzing the usage patterns of different household devices helps, among other things, to better predict future consumptions and ease the macro-management of overall power demand for smart grids [Barbato et al., 2011].

State-of-the-art research works in this area often apply time series classification techniques for identifying appliance electricity usage profiles [Lines et al., 2011; Elhamifar and Sastry, 2015]. However, our research claims that such approaches are not suitable for large-scale electricity usage due to their time complexity, and the fact that they may appear sensitive to noises (leading to inaccuracies in classification of similar appliances). One research hypothesis of this study is that electricity usage of a given appliance can be viewed as unique and recognizable repeating signals that could be represented as a sequence of *sentences*. In this respect, our research aims to develop an innovative time series classification approach – DSCo that is based on language modeling techniques as discussed in Chapter 5 – that help to identify the "language" of each type of appliances. In a more concrete level, the language modeling approach aims to capture with its n-grams both local discriminative features and overall curve shapes. On the one hand, local discriminative features (e.g., sharp edges) help to distinguish/identify appliances with different electricity usage patterns during turn-on and turn-off phases. On the other hand, overall curve shapes are essential to minimize the impact of noise, which can affect local features. Existing techniques [Zeifman and Roth, 2011] often focus on detecting whether a given appliance is on or off, and at what time it has been switched. As a preliminary investigation in the direction of energy disaggregation, our study assesses the potential of the proposed language modeling approach to identify the contributions of several appliances in a combined electricity usage data, and particularly to identify whether pairwise combinations of appliance consumption data can be accurately profiled. Overall, the chapter's contributions are:

- Proposal and development of an innovative accurate and practical approach to profiling household appliances based on their electricity consumption readings.
- Evaluation – *using a large dataset* – of our approach against the state-of-the-art approach to time series classification.
- Presentation of preliminary findings on profiling of electricity consumption of combinations from different household appliances.

Related work in the literature and the necessary background for the approach are respectively introduced in Sections 7.2. We have detailed our DSCo approach in chapter 5, as a result we skip the methods used in this chapter and present experiments and evaluation results in Section 7.3; discussion and conclusion follow in Sections 7.4 and 7.5.

7.2 RELATED WORK

A set of techniques for appliance classification and profiling have been introduced in the literature. For example, NILM systems break summarized appliance usage information into individual appliance usage patterns. Typically, household appliances can be divided into four categories: 1) permanent devices that work constantly with steady active and reactive power consumption; 2) on-and-off devices that could be either on or off at any point of time (e.g., lights, toasters...); 3) appliances with many states where energy consumption may switch from one state to another (those appliances can be modeled as Finite State Machines – FSM); and 4) continuously variable devices, which exhibit no usage patterns. Given these categories, only categories 2) and 3) can be detected using NILM techniques [Zeifman and Roth, 2011]. Indeed, the main NILM research effort has been on appliance signature extraction, which is often viewed as a pattern recognition task. Numerous techniques have been applied for that purpose, including k Nearest Neighbors (kNN) [Berges et al., 2010] that looks for the most similar patterns from the existing pattern pool, Naïve Bayes classifier [Lines et al., 2011] that is a probabilistic classifier modeling patterns as feature vectors (assuming feature independence within vectors), and Powerlets [Elhamifar and Sastry, 2015] that uses *sharp edges* in appliance usage patterns as discriminative features. It is important note that some recent NILM research takes advantage of features other than appliances' real and reactive power (e.g., transients and harmonics) [Laughman et al., 2003], however, such techniques require very high sampling frequency leading to expensive measurement hardware.

Regarding appliance usage representation, many research efforts have adopted the time series model, which is the "natural" representation of device energy consumption data. Lines et al. [Lines et al., 2011] classify household devices by electricity usage profiles using a time series classification approach, and compare classification results using different algorithms such as kNN, Naïve Bayes, Random Forest, *etc.*. Similarly, Basu et al. [Basu et al., 2012] adapted a time series model by introducing a multi-label classifier approach to predict appliance usage in the near future.

Time series classification is an active research field in the machine learning community, where a number of approaches and algorithms have been proposed, including neural networks [Nanopoulos et al., 2001], decision trees [Rodríguez and Alonso, 2004] and SVM [Wu and Chang, 2004], while empirical studies have shown that kNN works exceptionally well [Batista et al., 2011]. To perform best, kNN classifiers leverage the Dynamic Time Warping (DTW) distance, thus mitigating problems by warping the time axis [Ratanamahatana and Keogh, 2005]. Another line of research – *shapelets-based classifiers* [Ye and Keogh, 2009] – focuses on finding the most discriminative features. *Shapelet* algorithms are proven to be accurate for time series classification tasks, but they are generally time consuming due to the exhaustive process searching for the best features. As a result, latest research efforts focus on reducing the time complexity of these algorithms [Mueen et al., 2011; Rakthanmanon and Keogh, 2013].

7.3 EMPIRICAL EVALUATION

Table 7.1: Characteristics of appliance electricity usage data from the UCR archive and classification accuracy comparison between the performance of 1NN with Euclidean and DTW distance and our approach. Best classification accuracy results are highlighted in bold font.

	Characteristics				Classification Accuracy		
Dataset Name	**# Classes**	**# Training Instances**	**# Testing Instances**	**Instance Length**	**1NN (Euclidean)**	**1NN (DTW Best Warping)**	**Our Approach**
Computers	2	250	250	720	0.576	0.62	**0.668**
ElectricDevices	7	8,926	7,711	96	0.55	0.624	**0.651**
LargeKitchenAppliances	3	375	375	720	0.493	**0.795**	0.72
RefrigerationDevices	3	375	375	720	0.395	0.44	**0.528**
ScreenType	3	375	375	720	0.36	0.411	**0.448**
SmallKitchenAppliances	3	375	375	720	0.341	**0.672**	0.659

Our approach has been implemented in order to evaluate the classification performance against state-of-the-art approaches. The set of experiments has been conducted using the computing platform presented in [Varrette et al., 2014], and considering several datasets and scenarios. Unless otherwise specified, we extracted – *for all datasets* – subsequences of lengths ranging from 2 to 20. Furthermore, data samples longer than 100 are reduced to 100 using SAX in order to speed up experiments.The evaluation results and findings are presented and discussed through the sections 7.3.1 to 7.3.3. The source code of our prototype is released to the public[1] to increase reproducibility.

7.3.1 *Evaluation against normalized datasets*

Our approach is compared against state-of-the-art approaches with an openly accessible dataset archive (the UCR archive [Chen et al., 2015]), which is widely adopted in the research community. Besides, it comes with classification results for state-of-the-art classification algorithms for benchmark comparison, namely Euclidean- and DTW-distance based Nearest Neighbor as shown in Table 7.1. This archive contains many different datasets from interspecies images to medical applications such as electrocardiograms.

In this research, only datasets related to household appliance electricity usage profiles are tested since our main interest is on classifying electric devices. The characteristics of these datasets are summarized in Table 7.1, where performance of our approach is compared with Euclidean-based 1NN and DTW-based 1NN (best accuracy varying the warping size from 0 to 100 percent of the series length, while best accuracy varying SAX alphabet size parameter from 3 to 20). Although our approach performs well with regard to these datasets, a more in-depth study should be conducted due to the fact that data that compose these datasets have already been normalized, adding that each dataset contains only a small number of samples as well as classes. Furthermore, the training sets and testing datasets have been roughly equally divided in terms of number of instances.

1 https://github.com/serval-snt-uni-lu/profiling-appliances

7.3.2 *Evaluation against real-world readings*

Table 7.2: Confusion matrix from ten-fold cross-validation experiment on the most recorded appliances.

		Classified As										F-Measure
		Fridge Freezer	Kettle	Microwave	Dishwasher	Washing Machine	Shower	TV LCD	Light 1	Light 2	Vacuum Cleaner	
Actual	Fridge Freezer	**9,380**	1	7	97	58	1	509	40	16	1	0.931
	Kettle	3	**4,343**	215	21	90	109	6	0	6	267	0.831
	Microwave	36	885	**936**	64	171	242	29	7	6	164	0.482
	Dishwasher	20	23	35	**895**	203	28	12	9	3	32	0.660
	Washing Machine	46	22	58	159	**1,302**	48	53	7	3	32	0.676
	Shower	0	36	16	27	35	**246**	2	5	4	29	0.408
	TV LCD	476	20	24	99	186	37	**2,174**	116	106	22	0.699
	Light 1	52	5	13	27	28	13	101	**341**	40	10	0.575
	Light 2	16	8	8	18	15	34	70	32	**227**	2	0.540
	Vacuum Cleaner	1	52	33	46	32	48	3	0	0	**45**	0.104

To test our approach with real-world data, the dataset obtained from the UK Household Electricity Usage Survey (HEUS) project [Zimmermann et al., 2012] has been used, which contains electric appliance usage readings from 251 households (monitoring periods: 2010 to 2011). In our study, a subset containing more than 50 million appliance electricity usage readings from 27 discting household have been considered (readings being sampled at two-minute intervals). As a first step, from this subset the ten appliances that have the most recorded readings has been extracted, along with their electricity consumption patterns as depicted in Figure 7.1.

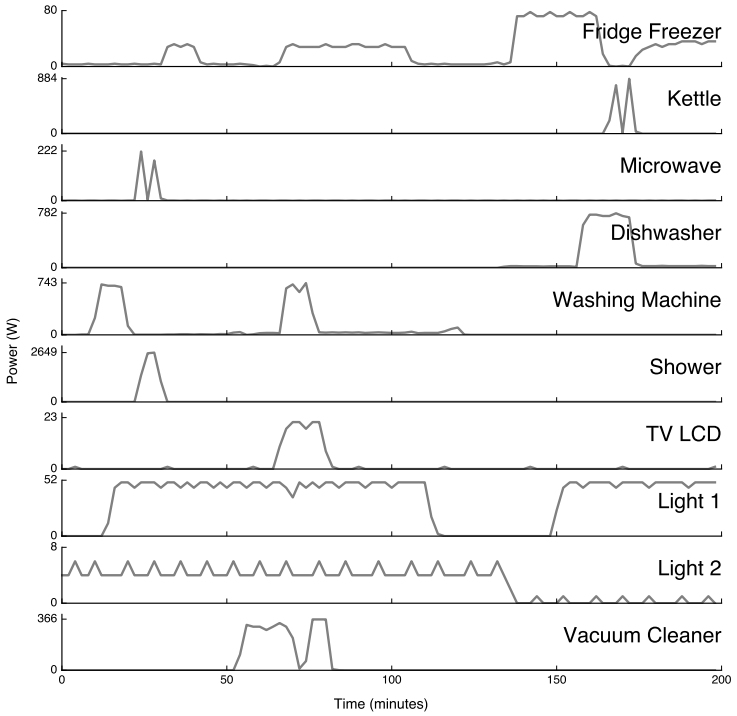

Figure 7.1: Electricity consumption patterns of ten most recorded appliances from 27 households surveyed in the HEUS project.

Afterwards, readings are clustered into time series with size of 100, which corresponds to appliances' electricity readings during a 200-minute period. Note that readings that are roughly constant during each that period have been removed, leading to 25,652 time series. A ten-fold cross-validation is then performed to classify these time series. More specifically, this dataset is firstly divided into ten portions, nine of them being used for training and the remaining one being used for testing. This process is repeated ten times and the average classification accuracy is reported at 77.4%. Compared with [Lines et al., 2011], where data is sampled with a 15-minute interval and clustered into weekly and daily series, the best results obtained are 61.34% and 55.81% respectively. In addition, since we have mixed appliance usage data across households, the good classification results suggests that certain kinds of household appliances exhibit similar profiles. Thus, it is sensible to build a profile repository for different appliances so that, later, unknown appliances can potentially be matched with profiles in the repository, without the need of setting up monitoring devices in each household and learning device profiles from scratch.

Table 7.2 presents the confusion matrix from the ten-fold cross-validation experiment. The standard F-measure scores (the harmonic mean of precision and recall calculated following Equation 7.1) are also reported in order to present the performance of our approach in a more interpretable manner.

$$F-measure = F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{7.1}$$

It can be observed that our approach provides good results for certain appliance categories such as *Fridge Freezer* and *Kettle*. Furthermore, our approach tends to perform better for appliances with larger number of reading records, indicating that good performance can be achieved with sufficiently large training dataset.

7.3.3 *Combining appliance consumption readings*

So far, the evaluation focused on exclusive energy readings for specific appliances. Unfortunately, in many cases, it is often difficult to get such readings. Instead, it is likely that some appliances (e.g., a fridge) stay on the `ON` mode, while others more occasionally used appliance (e.g., a microwave) is turned `ON` or `OFF`. Thus, a deeper investigation is carried out in this section in order to classify the combinations of appliances' energy consumptions. This study will help us to gain insights on the potential of the language modeling approach to support energy disaggregation schemes.

To this end, we first find out the ten most recorded appliances within the dataset following the same procedures mentioned previously. Then, we investigate which of these ten appliances are found in each household and sum up the power consumption data based on timestamps. In this way we are able to keep the original appliance usage patterns – i.e., which two appliances are used within the same period – from each household. Afterwards, we find out the most frequent consump-

tion sums representing appliance usage patterns and set to classify them using our language modeling based approach. All the combination samples are mixed in order to create a sufficiently large dataset. Finally, a ten-fold cross-validation experiment is conducted in order to differentiate various appliance usage consumption combinations.

Table 7.3: Classification results for ten-fold cross-validation between two appliance energy consumption combinations. Columns P, R and F stand for Precision, Recall and F-Measure respectively.

Combination 1	Combination 2	P	R	F
Fridge Freezer + Kettle	Kettle + Light 1	0.941	0.941	0.941
Fridge Freezer + Kettle	Kettle + Microwave	0.959	0.958	0.958
Fridge Freezer + Light 1	Fridge Freezer + Microwave	0.551	0.549	0.549
Fridge Freezer + Light 1	Fridge Freezer + Kettle	0.647	0.637	0.638
Fridge Freezer + Light 1	Kettle + Light 1	0.977	0.977	0.977
Fridge Freezer + Light 1	Kettle + Microwave	0.990	0.990	0.990
Fridge Freezer + Light 1	Fridge Freezer + Washing Machine	0.472	0.469	0.470
Fridge Freezer + Microwave	Fridge Freezer + Kettle	0.620	0.617	0.617
Fridge Freezer + Microwave	Kettle + Light 1	0.975	0.975	0.975
Fridge Freezer + Microwave	Kettle + Microwave	0.987	0.987	0.987
Fridge Freezer + Washing Machine	Fridge Freezer + Microwave	0.538	0.538	0.538
Fridge Freezer + Washing Machine	Kettle + Microwave	0.990	0.990	0.990
Fridge Freezer + Washing Machine	Kettle + Light 1	0.977	0.977	0.977
Fridge Freezer + Washing Machine	Fridge Freezer + Kettle	0.600	0.599	0.598
Kettle + Light 1	Kettle + Microwave	0.618	0.596	0.587
Kettle + Washing Machine	Kettle + Microwave	0.625	0.573	0.558
Kettle + Washing Machine	Kettle + Light 1	0.585	0.576	0.574
Kettle + Washing Machine	Fridge Freezer + Microwave	0.979	0.979	0.979
Kettle + Washing Machine	Fridge Freezer + Light 1	0.986	0.986	0.986
Kettle + Washing Machine	Fridge Freezer + Washing Machine	0.961	0.961	0.961
Kettle + Washing Machine	Fridge Freezer + Kettle	0.949	0.948	0.948
Microwave + Light 1	Fridge Freezer + Washing Machine	0.965	0.965	0.965
Microwave + Light 1	Fridge Freezer + Light 1	0.961	0.961	0.961
Microwave + Light 1	Fridge Freezer + Kettle	0.947	0.947	0.947
Microwave + Light 1	Fridge Freezer + Microwave	0.955	0.955	0.955
Microwave + Light 1	Kettle + Microwave	0.839	0.833	0.829
Microwave + Light 1	Kettle + Washing Machine	0.863	0.850	0.847
Microwave + Light 1	Kettle + Light 1	0.798	0.795	0.792

Table 7.3 presents the most frequently found appliance combinations and classification results. As shown, it is possible to differentiate combinations of appliance usage, e.g., the *Fridge Freezer* and *Light 1* combination can be differentiated from the *Kettle* and *Microwave* combination. This can be explained by the fact that each appliance has a different usage patterns(*Fridge Freezer* and *Light 1* tend to be constant while *Kettle* and *Microwave* both exhibit burst patterns). On the other hand, some combinations such as the *Fridge Freezer* and *Light 1 vs. Fridge Freezer* and *Washing Machine* are difficult to differentiate, whose possible reasons may be:

- Appliance usage data are sampled at a low frequency (a 2 minute interval may not be enough to pertain appliance usage characteristics, or to sum up the usage data).

- Our approach reduces real-valued power consumption into alphabets, and this numerosity reduction process may have incurred inaccurate data representation for combinations.
- The size of training data may not be sufficiently large, thus leading to insufficient training process and misclassified testing instances.

7.4 DISCUSSIONS

The approach proposed in this chapter relies on a mechanism to transform household appliance electricity usage data into a string, where strings of a same appliance category forms a appliance-specific dictionary. The results from our empirical experiments showed how performant our approach is, and made it possible to validate the analogy between appliance energy consumption patterns and natural language sentences. Besides classification accuracy, this approach has a few other advantages. For example, our approach is able to handle data samples of different length. This is helpful in that it eases data preparation and segmentation in practice. Furthermore, this approach makes it easy to store data samples as well as learned profiles/models in a distributed manner, so that existing distributed and cloud computing techniques can be easily taken advantage of and further boost the efficiency of our approach.

Nonetheless, some limitations of our approach can be discussed. First, the model-based nature underlying our approach demands a large training set so as to achieve a high performance. Then, with our current settings, $T_1 = [1, 1, 4, 4, 10, 10]$ and $T_2 = [100, 100, 300, 300, 600, 600]$ will be transformed into the same string (*aabbcc*) when SAX alphabet size is set to 3 and string length set to keep the original series length, which can be an issue and cause mis-profiling of appliances. On the opposite, this issue can benefit our approach when profiling a same category of appliances having different power draws. For example, a 1000 KW and 1500 KW hair dryer should indeed have same consumption patterns, while a 1000 KW dishwasher should exhibit different profiles compared with the 1000 KW hair dryer. For the same reason, when combining appliance consumption readings, data from one appliance with small power draw may be dwarfed by an appliance with a much larger power draw, so that the former becomes noises and the overall curve shape is dominated by the latter.

7.5 CONCLUSIONS AND FUTURE WORK

Electricity usage profiling of household appliances is becoming an important step for identifying malfunctioning devices and generating automatic alerts about unusual consumptions. In this context (i.e., household appliances profiling context), this chapter investigates the capability of a language modeling approach for time series classification. To this end, an innovative approach is proposed in this chapter, which aims to first transform energy consumption readings – *which consist of*

real-valued time series data – into texts, and then to build per-class language models (*i.e. profiles*) from these texts. Such class models can therefore be used for new electricity usage readings in order to predict the corresponding appliance category. The proposed approach has been implemented and evaluated through a set of experiments considering both normalized datasets from the research community and real world datasets from the UK Household Electricity Usage Survey project (27 households monitored over one year). These experiments show that our approach performs generally better than state-of-the-art time series classification approaches.

8

SENSING BY PROXY OF INDOOR TEMPERATURE MOVEMENTS

> "From a drop of water," said the writer, "a logician could infer the possibility of an Atlantic or a Niagara without having seen or heard of one or the other."
>
> *Arthur Conan Doyle*
> *A Study In Scarlet*

Outline

We take advantage of hierarchical clustering of time series to conduct anomaly and activity detection inside buildings. Specifically, we collect indoor temperature readings from different classrooms in a school and model the temperature readings as time series. Through hierarchical clustering of such time series, we are able to accurately infer the relative locations of different classrooms as well as coarse-grained type of activities inside different rooms.

8.1 INTRODUCTION

Citizens in a modern society spend a majority of their time everyday inside buildings working or relaxing. In turn, buildings consume a surprisingly large portion of total energy consumption by all sectors. For example, 41% of energy consumption attributes to buildings in the US and buildings consume even more energy than industry in the EU [Nguyen and Aiello, 2013]. Many initiatives have thus been proposed to combat the energy consumption issue. As the concept of Internet of Things (IoT) develops, more and more proposals focus on devising more efficient Building Energy and Comfort Management (BECM) systems, which try to fulfill users' comfort requirements while reducing energy footprints for building operations including heating, ventilation, and air conditioning (HVAC), lighting and plug loads. Indeed, research has shown that BECM systems can potentially reduce buildings' energy footprints in both simulated and real-world evaluations.

BECM systems often involves taking advantage of heterogeneous sensors, such as passive infrared (PIR) sensors, cameras, motion and presence detectors, and environmental sensors like temperature, humidity, CO_2, etc., to monitor the status of the building and especially occupant activities, since conservative behaviors can

help reducing building operation energy consumption by one-third compared to design point benchmark while careless ones may increase energy footprints by one-third [Nguyen and Aiello, 2013]. However, the assumption of tracking individual occupants in real-time is largely impractical and rarely adopted in real-world cases due to technological, construction and maintenance cost and privacy challenges.

To tackle with these challenges, we seek to devise an plug-and-play and cost-effective approach that takes advantage of existing BECM systems and investigates the feasibility of gaining extra intelligence about corresponding buildings and their occupants from such systems. Since user behaviors have a large impact on their indoor environments, we can profile these behaviors by proxy of the resulting impacts they have made. To be exact, sensing by proxy here refers to inferring latent factors with indirect measurements on activity traces rather than directly measuring activities. To make our approach more applicable to different scenarios and cost-effective, we take advantage of mature and widely deployed temperature sensors. We have conducted this study using real-world settings and all our data has been collected from a school building in western Europe, which was planned and constructed around 2000 and is equipped with basic sensors and actuators (e.g., temperature sensors and HVAC system) to facilitate building operations. The main contributions of this chapter include:

(a) We adopt a sensing by proxy paradigm to reduce costs and relax users' privacy concerns. We confirm that agglomerative clustering of temperature evolution of indoor environments (with minimal occupant activities) produces accurate adjacency maps with regard to the physical location of each temperature sensor. This adjacency map is helpful and can be complement to indoor floor plan inference and localization.

(b) We prove that it is feasible to infer coarse-grained intelligence about occupant activities using agglomerative clustering of temperature evolution of indoor environments with occupant activities even if data have been collected with low frequency in a non-intrusive manner.

(c) We provide a data analytics tool that extends off-the-shelf BECM systems for smart homes and smart buildings that helps owners and operators to understand the overall status the buildings with regard to its previous status. Our approach can also be used to track anomalies within buildings.

The remainder of this chapter is organized as follows. Section 8.2 prepares readers with the necessary technical background and Section 8.3 introduces works that are related to ours. We present our methodology in Section 8.4 and real-world experiment results in Section 8.5. Finally we conclude with future research directions in Section 8.6.

8.2 BACKGROUND

Time series clustering is a common type of unsupervised time series mining task that tries to partition time series into homogeneous groups while maximizing within-group similarity and between-group dissimilarity [Liao, 2005]. Clustering algorithms can be categorized into different families based on their underlying models, for instance hierarchical clustering which is based on connectivity and centroid-based clustering (e.g. k-means) where clusters are represented by a representative point. In this chapter, we are especially interested in the former, since hierarchical clusters can be represented as a dendrogram, which depicts the hierarchy arrangement of clusters that can be merged with another at certain distances. Hierarchical clustering do not attempt to generate an arbitrary number of clusters. Instead, it produces a hierarchy that is easier for users to understand and users can set the break points by themselves. Hierarchical agglomerative clustering has recently received great interests in pattern recognition and become especially popular in financial applications. Figure 8.1 shows an example of hierarchical clustering results, where companies with similar business domain and activities have been grouped together. Note that in dendrograms, the height of a branch indicates how different it is from another while the horizontal orientation is generally irrelevant.

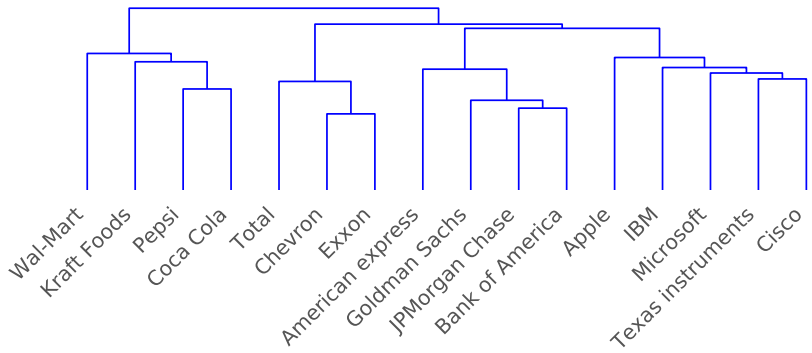

Figure 8.1: Example dendrogram from hierarchical clustering, using daily stock price variations from January 2012 to January 2016.

Hierarchical clustering can employ either a bottom-up (agglomerative) or top-down (divisive) approach. The former starts with a single instance from the dataset and gradually aggregates instances into clusters until all instances are grouped into a single cluster, while the latter starts with the whole dataset and iteratively divide the dataset into clusters. In general, agglomerative methods are computationally more efficient than divisive ones. Thus we favor the former, and especially the Ward's method [Ward Jr, 1963], which is a popular algorithm used to minimize the total within-cluster variance. Recall that Ward – as an agglomerative approach – works incrementally, the distance (namely the Ward's Linkage) of clusters $I \cup J$ and K are calculated based on a distance update formula $D_{Ward}(I \cup J, K)$ as specified below:

$$\sqrt{\frac{(|I| + |K|)D(I,K)^2 + (|J| + |K|)D(J,K)^2 - |K|D(I,J)^2}{|I| + |J| + |K|}}$$

where I and J are two clusters to be joined into a new cluster and K is any other cluster, and $|*|$ denotes the number of instances in one cluster. The computational complexity of Ward is $O(n^2)$, where n is the size of the dataset. Ward is widely available in many software packages, for example Matlab and Wolfram Mathematica.

8.3 RELATED WORK

Recent research on BECM systems focuses on collectively taking advantage of both real-time occupancy information and occupant preferences when designing more efficient building control systems. For instance, Chen et al. [Chen et al., 2009] propose a BECM system that keeps track of occupants' real-time location to enable fine-grained control of ambient environment including lighting, cooling, heating, etc. As sensors and actuators are deployed in buildings and these systems are connected to external networks such as the Internet, occupant security and privacy become a more challenging task since sensor data can be leveraged to make unwanted inferences about occupants and their behaviors [Zhu et al., 2015]. For instance, Yang et al. [Yang et al., 2014] have conducted empirical experiments using motion sensors in a three-person single-family home and electricity meters in a twelve-person university lab, and shown that data from these sensors can enable inferring real-time occupancy and even occupants' identities. Another approach [Zhen et al., 2008] takes advantage of RFID technologies and implements a localization algorithm that learns about the location of occupants.

Information about indoor environment is important for many applications including indoor localization services, security services like access control and alarms, and privacy protection. However, this information is often either unavailable or obtaining it is time-consuming due to effort-intensive negotiations with building operators. As a result, many approaches have been proposed to explore and infer indoor environments. Earlier approaches takes advantage of laser scanners [Okorn et al., 2010] to infer and reconstruct indoor floor plans, while more recent works leverage mainly commodity sensors available on smartphones [Shin et al., 2012] and takes a crowd sensing approach. For instance, CrowdInside [Alzantot and Youssef, 2012] takes advantage of sensors (including accelerometers, magnetometers, gyroscopes, etc.) on smartphones to construct occupants' motion traces and then infer floor plan as well as room and corridor shapes. [Jiang et al., 2013] introduces another indoor floor plan construction system that takes advantage of Wi-Fi signals to construct room adjacency graphs and leverages user motion data collected from smartphones to to estimate room sizes and orders. Unlike these approaches that involve taking advantage of heterogeneous or *ad hoc* (specific purposed) sensors, our approach does not require any sophisticated sensing hardware and utilizes only indoor temperature sensors, which are commonly found in modern buildings with HVAC control systems.

Indoor occupant activity inference and detection is another research trend since more and more sensors are installed in buildings and occupants are becoming

increasingly concerned about their own privacy. For instance, motion sensors and smart meters can be used for detecting whether a room is occupied and even for analyzing occupant identities [Yang et al., 2014]. A more recent work [Shih and Rowe, 2015] explores the resonance effect of rooms and devise models to infer the number of occupants by observing changes in the ultrasonic spectrum reflected back from a centrally located ultrasonic chirp transmitter. Our work has been largely influenced by that of Jin et al. [Jin et al., 2015], where the authors try to infer implicit factors by indirect measurements based on the physical environment. They argue that occupancy can be inferred by indoor CO_2 concentration. Since CO_2 sensors are not as widely available as temperature sensors, we try to investigate the sensing by proxy paradigm using temperature sensor readings. Note that our work mainly concerns inferring indoor environments and occupant activities from sensor data, instead of exploring the vulnerability of networking protocols such as KNX [Antonini et al., 2014].

8.4 METHODOLOGY

Since different buildings operate with different BECM systems, to extend such systems we have to find a common interface or common type of data when conducting latent sensing. Fortunately, temperature sensors are usually available in the majority of BECM systems because of the requirements by HVAC devices. Besides the availability, we believe indoor temperature movements are largely influenced by both natural factors and occupant behaviors, making temperature sensors a perfect data source for inferring relevant information from buildings and their occupants behaviors. For instance, we have collected data – including temperature measurements and set-points, lighting, alarms, etc. – from the BECM system located in a school building, and indoor temperature records attribute to a significantly large portion in our database. To be exact, around 60 percent of total records are indoor temperature measurements, while in comparison outdoor weather station data contributes around 15 percent. In this section, we introduce the whole pipeline of our approach from collecting data, processing these data so that it fits the agglomerative clustering algorithm, to the validation process.

8.4.1 *Data Collection and Processing*

We are interested in collecting indoor temperature data from buildings' BECM systems. Fortunately, majority of BECM systems are based on open communication standards such as KNX[1] or LON[2]. As a result, it is easy to get a compatible watchdog module and simply attach it to the control bus and start collecting data. We store all collected data in a database for ease of querying and retrieving purposes.

1 http://www.knx.org/
2 http://www.lonmark.org/

Note that in practice sensor readings generally exhibit different statistical characteristics. For instance, different temperature sensors report temperatures at different frequencies and the amplitude of values may also differ. Besides, abnormal and missing values are very common, making the collected data quite noisy. To proceed processing the data, we have to conduct data cleansing tasks as specified below:

(a) Resampling. Specifically, we choose to down-sample data records using a uniform frequency of one hour for temperature readings. This process helps reducing noises as noisy data points can be filtered out. Furthermore, down-sampling can greatly reduces the dataset size and improve computation efficiency.

(b) Interpolation. Some sensors may be missing values at certain timestamps even after down-sampling. Missing values are common in our case due to sensor failures and occasional server shutdown. There are many missing value imputation techniques [Royston et al., 2004], however, we choose to linearly interpolate these missing values since temperatures of indoor environments do not tend to change drastically.

(c) Normalization. Temperature sensors in different locations may report values of significantly different amplitude levels. This can result in inaccurate computation especially with distance measures such as the DTW distance. In this chapter, we are more concerned with the oscillation development for readings from each individual sensor. As a result, we divide each sensor's readings by their corresponding standard deviation for normalization, i.e., $t'_i = t_i / \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(t_i - \mu)^2}$ for $0 \leqslant i < n$, where $\mu = \frac{1}{n}\sum_{i=0}^{n-1} t_i$. This step is especially important if we are concerned with the overall temperature movements instead of the absolute terms, which are easy to get using simple algebras and statistical methods.

Afterwards, we calculate the pairwise distance of temperature movements of different rooms (in the form of time series) using both Euclidean and DTW distance and generate a corresponding distance matrix, which is then fed as input to the agglomerative clustering process.

8.4.2 *Baseline Establishment and Validation*

Since our intuition is that temperature movements are mainly influenced by natural factors and occupant behaviors, to validate this assumption we have to separate the influence of such two factors. If we can achieve this, then we can continue investigating how each factor contributes to the temperature movements. Separation of natural factors and occupant behaviors can be easy and sometimes maybe trivial since we can just find out when occupants are in the building. For example, offices and schools usually have a consistent schedule that tells us when rooms are occupied (e.g. daytime on weekdays) or not (e.g. nighttime or holidays). For simplicity, we choose to split all temperature data into daytime and nighttime readings.

When investigating how each factor contributes to indoor temperature movements, consider first the natural factors. It is obvious that the physical location can have the biggest impact on room temperature. For instance, rooms facing south (in the Northern Hemisphere) would generally fluctuate more drastically than rooms always in shadows, provided that all rooms in the same building have the same heat insulation features. In this case, we can validate the clustering results against the floor plans in order to evaluate the impact of natural factors. On the other hand, since human activities can greatly impact indoor environments, clustering temperature movements under human influence will tell us more about the actual activities.

In summary, agglomerative clustering of indoor temperature movement data collected when minimal occupant activities are present will likely tell us more information about the physical locations of rooms; while clustering of temperature movement data when occupant activities are present will likely enable us infer how close activities in one room is to those in another room. We continue validating these assumptions with real-world data in the following section.

8.5 EXPERIMENTAL EVALUATION

In order to validate our assumptions, we have conducted experiments using data from a real-world building that is in daily use. In this section we present the experiments and their results. We present our research questions (**RQs**) and answer then along with experiment results.

8.5.1 *Experiment Subject and Data Collection*

All our data has been collected from a single school building in western Europe, which has around 100 classrooms, labs and offices located on five different floors and 86 of these rooms are equipped with a single temperature sensor in each room. This building was planned and constructed around 2000 and most rooms as well outside facades are equipped with sensors and actuators to monitor and control temperature and heating, ventilation, illumination, etc. for the ease of building operations. In total, this building has more than 1000 connected sensors and actuators. All these sensors and actuators (such as light switches and dimming units) have been connected to a KNX bus, which is a broadcast networking protocol where all communication telegrams passes on the bus and pre-configured source/destination pairs may send and receive only relevant telegrams and react. KNX is a very popular building control protocol that has been deployed in several millions of installations worldwide.

We have implemented a system to collect data from the building, as shown in Figure 8.2. The broadcasting nature of KNX makes it easy for us to simply attach a KNX-to-USB interface to the KNX bus and listens to every telegram on the bus to a gateway server via the USB interface (in our case, it is a Weinzierl KNX USB

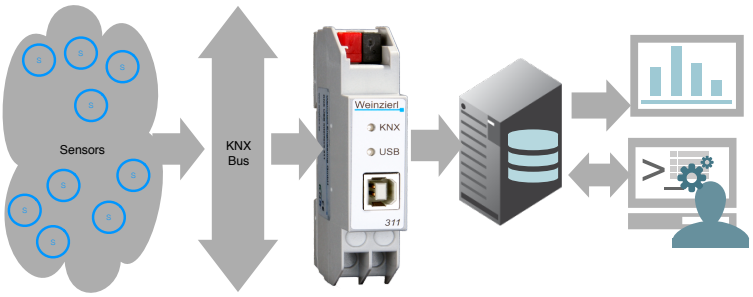

Figure 8.2: Overview of data collection and management process.

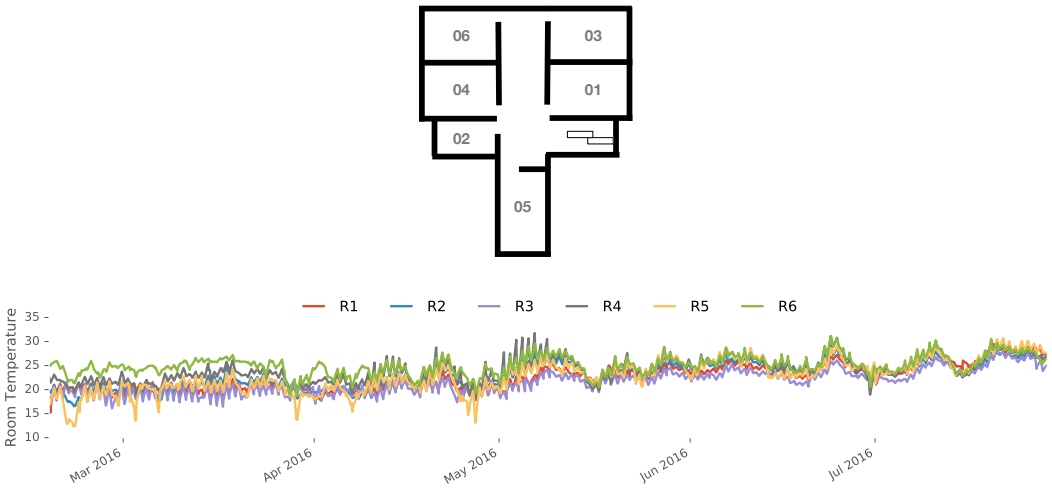

Figure 8.3: Simplified floor plan (top) and temperature readings during a course of around five months (bottom).

Interface 311, which costs around 200 Euros). This gateway then parses and stores all KNX telegrams to a Linux server. We started collecting data from this building from mid February, 2016. In this chapter, we have used data collected over a span of five months from February till July, 2016. Each record in our database consists of information such as telegram timestamp, source and destination addresses, KNX telegram type and message (generally a numerical value) parsed from this telegram. The commercial BECM system used by the school provides only a interface to monitor real-time readings from each sensor and no history data were store anywhere. As a result, we have also developed a dashboard (a web application) for the building operators to monitor and view charts about the real-time as well as historical records from sensors and actuators that are interesting to the building operator. In a backend, users and operators may configure a more customized dashboard interface by themselves.

8.5.2 *Inferring Indoor Environment*

Following our intuition that rooms located physically together should share similar patterns in room temperature movements due to similar natural influences such as sunshine, rain and wind. Optimally, human occupant impacts need to be ruled out when recording room temperatures that will be utilized for agglomerative clustering. To that end we choose those time periods when there is minimal occupant activities. Since our experiment subject is a school building and no one is in the building during night time, we split temperature readings into two subsets – daytime (07:00 to 19:00) and nighttime (19:00 to 07:00) readings – and explore only the nighttime temperature records. We start our research by investigating **if temperature sensor records can be used to correctly group physically nearby classrooms other than mere temperature fluctuations (RQ1).**

To answer **RQ1**, we extracted all the temperature records and down-sampled them to one-hour frequency to establish a tradeoff between reducing dataset size and lowering the amount of missing value interpolation. After normalization, we then calculate the pairwise Euclidean and DTW distance for all the preprocessed data and feed these distances to agglomerative clustering using Ward's linkage algorithm. We have experimented with data from all five floors. In order to make the readers understand better, we start with the top floor where temperature readings are available in only six classrooms.

Figure 8.3 shows the simplified floor plan (to protect the privacy of the school) and the temperature readings in each room for around five months. For human eyes, indoor temperature does not fluctuate greatly and these readings from different classrooms look more or less similar along the course. Especially, when the whether gets warmer, the temperature differences among different classrooms become smaller. By generating a distance matrix diagram of temperature movements from different rooms where darker blocks indicate more differences rather than similarities (cf. Figure 8.4 left), it may take some time (even for experts) to identify that R6 and R4 are more different than other classrooms. However, such a matrix does not tell us (or building facility management teams) about what can be the potential cause. Finally, note that the distance matrices in our case are symmetric, since $D(X, Y) = D(Y, X)$ for both Euclidean and DTW distance. We present whole matrices in this chapter for the sake of straightforwardness.

When applying agglomerative clustering techniques on the temperature movements (cf. Figure 8.4 right), our approach has produced two bottom level clusters $\{R1, R3\}$ and $\{R2, R5\}$, and moving up from the latter, R4 can be attached to $\{R2, R5\}$ to form a larger cluster, which can then be joined by R6. These results accurately corresponds with our floor plan in Figure 8.3, since rooms R1 and R3 are indeed located next to each other, and the so are the others. Also, R5 has been clustered with R2 but not with R1, probably due to the fact that there is a staircase between R5 and R1, while R2 and R5 are physically near each other. As a result, the answer to **RQ1** is indeed positive: room temperatures are good indicators of sensor

adjacency (and thus physical adjacency of rooms) inference using agglomerative clustering.

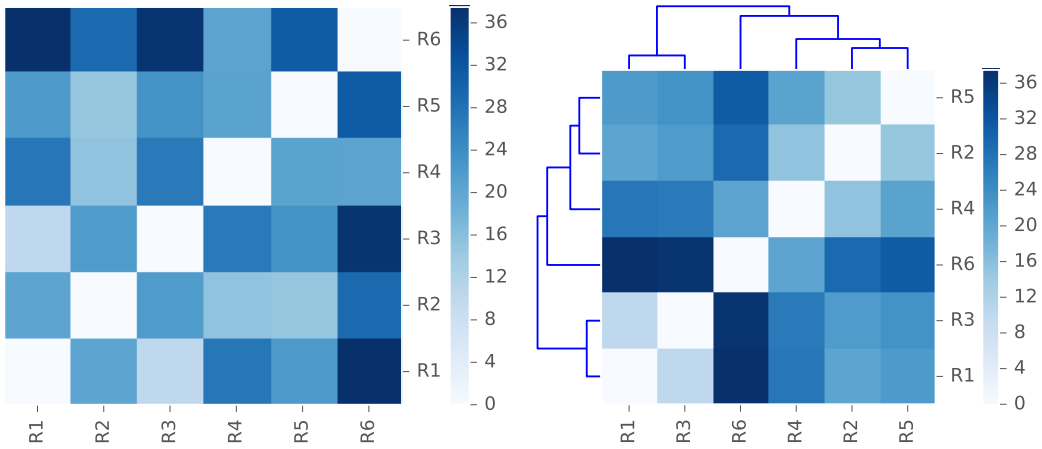

Figure 8.4: Distance matrix of temperature movements with Euclidean distance (left) and agglomerative clustering clustergram of temperature readings for six rooms with Ward (right).

Following **RQ1**, we wonder **how much data is needed for accurate inference of sensor adjacency (RQ2)** and **if Euclidean and DTW distance make a difference to clustering results (RQ3)**. To that end, we repeat our previous clustering process with the time span of room temperature readings using a sliding window with size varying from one to 160 nights and conduct the pairwise distance calculation with both Euclidean and DTW distance. In total, we have generated 26,080 dendrograms, which we programmatically validate if each dendrogram conflicts with our floor plan. A clustering output is considered as an error if any two rooms fall into one cluster in the dendrogram while these two are not strictly next to each other according to the floor plan, for instance, when a dendrogram reads that R2 and R6 or R4 and R1 belong to one cluster. Due to wide hallways, we do not consider classrooms located on different side of the hallway as close to each other.

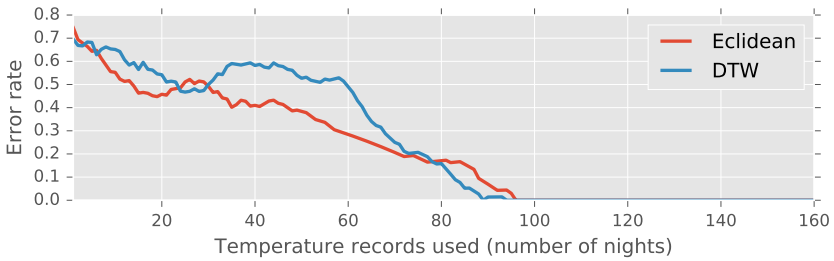

Figure 8.5: Error rate regarding the amount of data used for agglomerative clustering.

We present the error rate with each time span setting in Figure 8.5. It is obvious that the more data we use for clustering, the more accurate results are. And in our case with six classrooms, **using room temperature readings (with one-hour sampling frequency) during a span of three months produces clusters strictly correlated with regard to the floor plan (RQ2).** This span may seem long, how-

ever, in practice it may still be faster than effort-intensive negotiations with building operators (which took more than six months in our case to get the building floor plans). Besides, in this experiment we use hourly sampled temperature during nighttime, it is thus probable that using temperature readings with higher sampling rate would reduce the inference time. Regarding **RQ3**, it is obvious from Figure 8.5 that both Euclidean and DTW distance contributes to better clustering results with more data. Furthermore, DTW seems to be more sensitive to noises since its performance is not as stable as Euclidean with small datasets and smaller datasets are known to have smaller signal to noise ratio (SNR). Due to DTW's computational complexity is a magnitude higher than Euclidean, we find **Euclidean to be a more efficient and accurate distance measure than DTW (RQ3)**.

Next, we seek to find out **if our approach works with data from more classrooms and classrooms from different floors (RQ4)**. We have tested our approach with data from each of the five floors within the school building, results show that agglomerative clustering of room temperature indeed helps inferring the relative physical locations of classrooms even with as many as twenty rooms. Figure 8.6 presents an example clustering with nighttime temperature movements (during a span of five months) from all 20 rooms on another floor. In order to validate the result, we have assigned a color for each cluster in order to visualize the similarities between different rooms. To investigate if this clustering results corresponding to the physical locations of the rooms, we apply the same color scheme to the floor plan, which is shown in Figure 8.7. It is obvious that classrooms located next to each other generally fall into the same cluster, with the only exception that R28 is colored differently compared to its neighbors, indicating an anomaly. However, when comparing Figure 8.6, R28 is actually attached to the cluster of $\{R25, R27, R29, R31\}$ at a very late phase, suggesting that R28 is not so similar with the rest in its cluster. Besides, we have found out that R28 is used as a classroom for musical education while others are normal classrooms. This indicates that R28 may have special features (e.g., heat insulation or acoustic requirements) in design and construction stage.

Furthermore, as shown in Figure 8.8, our experiments demonstrate that clustering of rooms on higher floors generate more relevant results than rooms in lower ones, indicating that rooms located on higher floors of a building are more impacted by natural factors such as sun, rain and wind which influences buildings' energy performance. Especially, clustering results are terrible with indoor temperature movements in the basement, while better results are achieved above the ground. As a result, our results indeed indicate that **our approach are more sensitive to the floor location of rooms rather than the number of items to cluster (RQ4)**. This result suggest that this approach can potentially be performant with rooms in higher tower buildings as their indoor temperature are more impacted by natural factors.

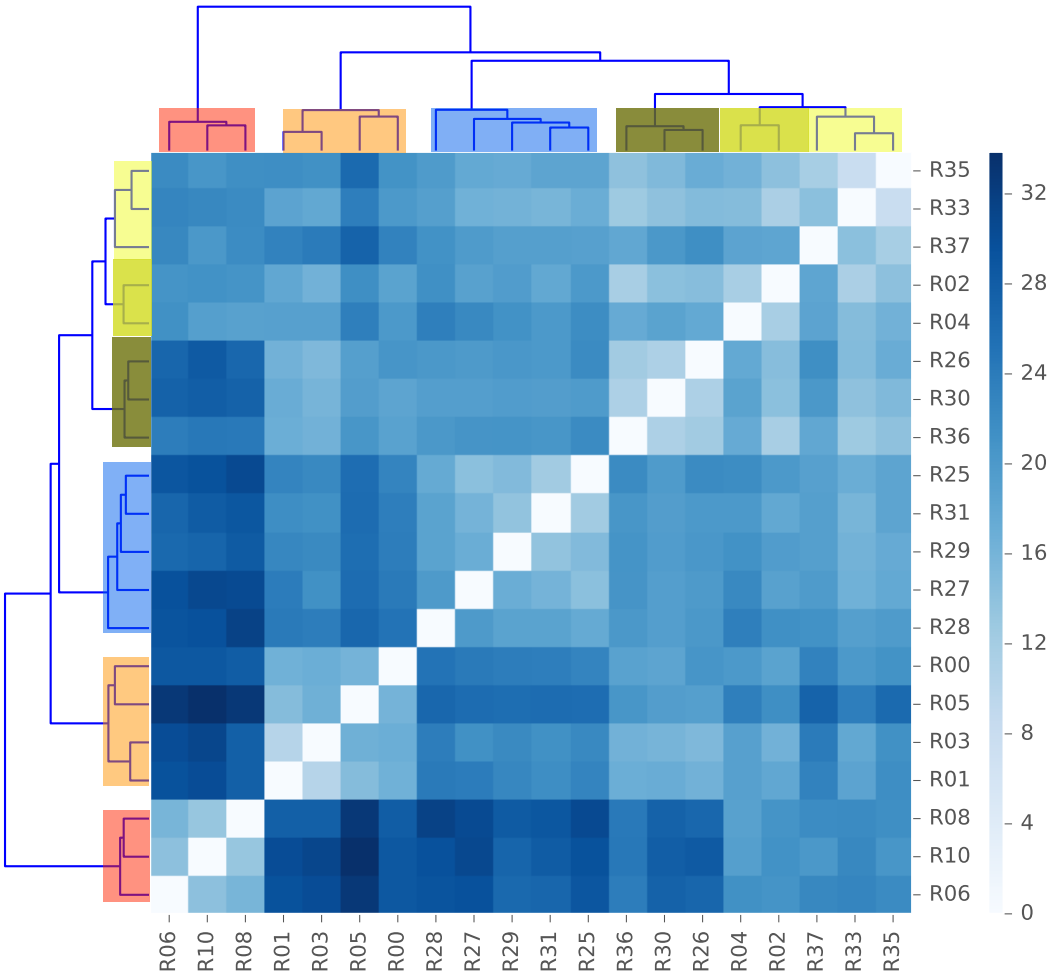

Figure 8.6: Agglomerative clustering on temperature movements of 20 classrooms.

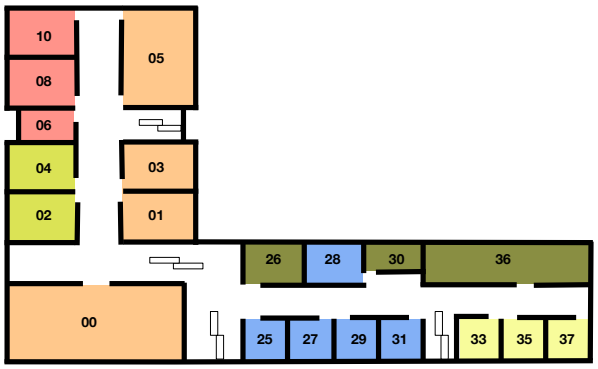

Figure 8.7: Simplified floor plan with coloring scheme from agglomerative clustering results.

8.5.3 *Towards Inferring Occupant Activities*

After validating that indoor temperature movements are closely correlated with rooms' physical locations, we set to investigate **the possibility of inferring occu-**

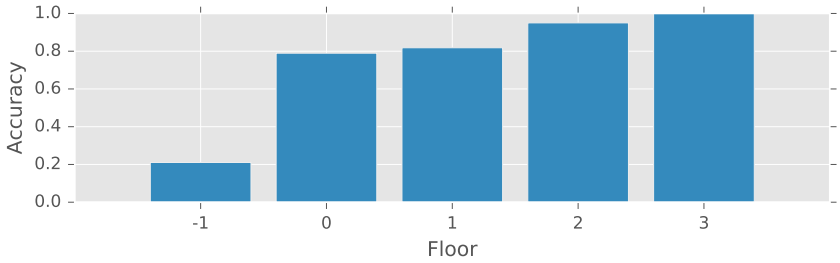

Figure 8.8: Clustering accuracy (number of correctly clustered rooms divided by total rooms on each floor) among different floors.

pant activities by proxy of temperature movements (RQ5). To this end, from the same school building we select temperature movements from 20 rooms that serve different functionality. For instance, some rooms are offices or labs, while others can be normal classrooms or libraries. Note that these rooms are located on different floors of the building and rooms with similar functionality are generally not physically close to each other. In this experiment we have only used temperature readings during daytime for a course of five months, so as to reduce the dilution by readings when no occupant activities are present.

Figure 8.9 presents the agglomerative clustering results. Much to our surprise, rooms with similar activities or functionality are generally clustered together. For instance, offices seem to have similar temperature movements and science labs do not share much similarity with other type of rooms other than slight similarity with offices. Besides, clustering results suggest that temperature movements in cafeteria, auditorium and reception are quite similar, probably due to the fact that all these rooms see bursts of occupants at specific time slots. Moreover, the rooms hosting kindergartens and preschool classes fall into one cluster, which can be joined by a meeting room, probably indicating that such rooms usually have smaller number of occupants. Last but not least, classrooms for training purposes (art, music and culinary) also have similar temperature movements.

While admittedly we are not able to predict what exactly a specific occupant activity is at the moment, by comparing with other activity traces we are still able to tell roughly what such an activity can possibly be. Furthermore, if we setup a database of how different occupant activities can impact indoor temperature movements with higher measuring requirements (e.g., higher sampling frequency, more accurate measurements and larger amount of records), we are confident that finer-grained activity inference can be achieved. As a result, **the answer to RQ5 can be positive when such a activity inference database is established.**

8.5.4 *Discussion*

Since most of the temperature sensors in our experiment subject report readings with low frequencies, it has been challenging to infer fine-grained information about indoor environment as well as occupant activities. Despite of dataset limi-

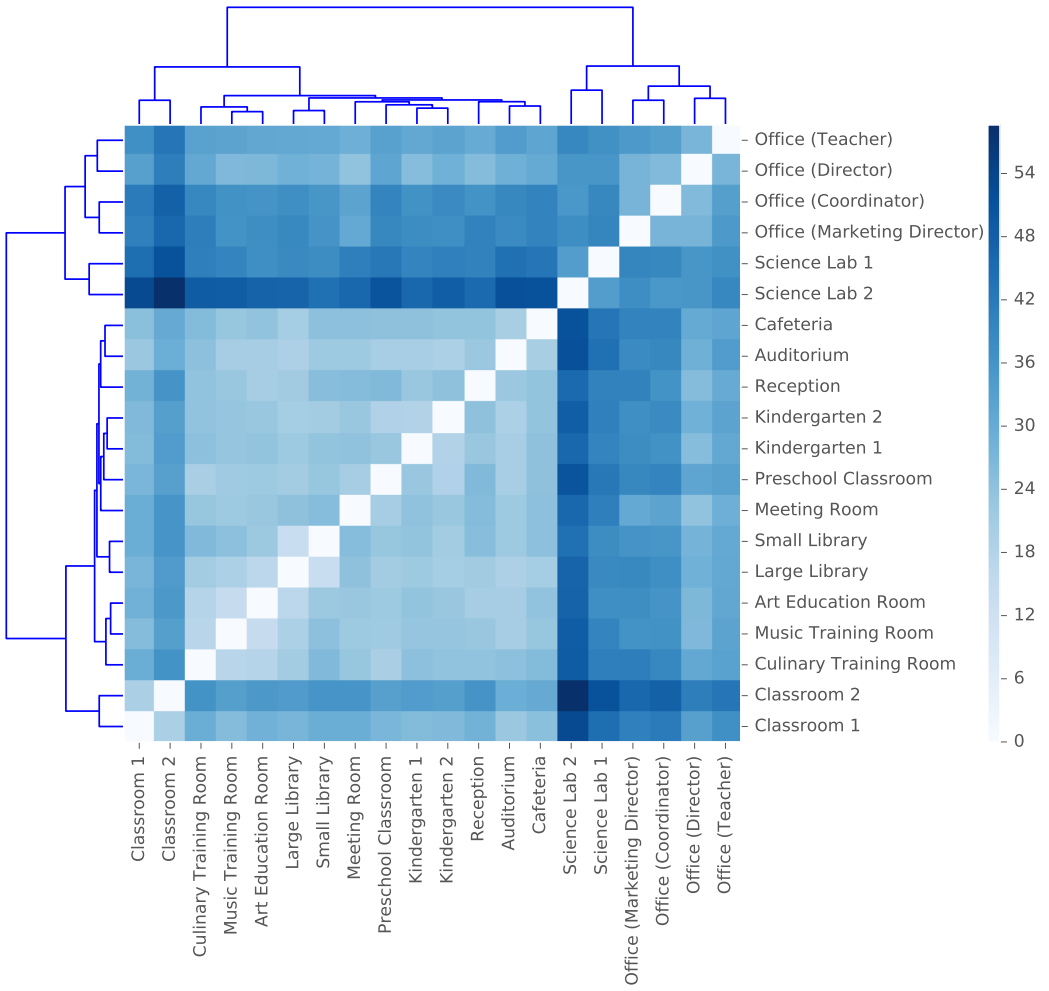

Figure 8.9: Agglomerative clustering on temperature movements of 20 rooms with different functionality.

tations, our approach is still able to discover relevant information such as indoor adjacency maps and coarse occupant activities. Furthermore, it is also beneficial to use our system for anomaly detection and building diagnosis. We are able to find groups of rooms whose temperature movements are different from all others. Such anomalies may indicate sensor failures, different HVAC configurations, malfunctioning heat insulation or simply abnormal occupant behaviors. In either case, this kind of information can provide a starting point that helps building owners and operators locating possible issues and fixing them. In addition, since our approach takes advantage of existing building control systems and requires minimal efforts for installation of new hardware, it has the potential to large scale deployment. In turn, when more data are collected, sensing by proxy can become more accurate.

8.6 CONCLUSION AND FUTURE WORK

It is well established that a thorough understanding of indoor environments and occupant activities is a key component in building control systems for better user comfort and more efficient energy usage. Unlike traditional approaches that leverage heterogeneous sensors or crowd-sensing paradigms to monitor indoor environments and activities, we adopt a non-intrusive sensing by proxy paradigm and take advantage of existing infrastructures to be cost-effective. Through extensive experiments with a school building that has 86 rooms equipped with temperature sensors, we are able to apply agglomerative clustering techniques on indoor temperature movements and infer useful information about both the physical features of rooms as well as the functionality of rooms based on traces from occupant activities.

In the future we plan to experiment our approach with more different buildings with respect to geolocations, heights, utility types (office or residential buildings) and number of occupants within rooms. It can also be beneficial to harvest finer-grained indoor temperature movements, i.e., collect temperature data with more accurate sensors and higher frequencies, so that we may infer more detailed information with regard to occupants' exact activities. In addition, other machine learning approaches can be helpful in the context of activity recognition and anomaly detection. Finally, other commonly available sensor and actuator data may also be interesting and adopted to our system.

Part V

SUMMARY

9 CONCLUSIONS

> What we call the beginning is often the end. And to make an end is to make a beginning.
>
> *T. S. Eliot*
> *Little Gidding, Four Quartets*

9.1 CONCLUSIONS

This dissertation has mainly investigated approaches for improving the efficiency and accuracy of TSC tasks. To that end, we have explored three different research avenues: 1) time series dimensionality reduction using DWT to speed up pairwise distance comparisons in distance-based TSC approaches; 2) language model-based DSCo to improve TSC efficiency in the case of large training datasets; and 3) extracting features from multiscale time series visibility graphs so that generic classifiers can be made suitable for TSC.

By means of a large scale empirical study involving taking advantage of discrete wavelet transform (DWT) for time series dimensionality reduction, we are able to provide assurances to practitioners by empirically showing, with various datasets and with several DWT approaches that DTW distance-based 1NN TSC algorithm yield similar accuracy on both compressed (i.e., approximated) and raw time series data. We also show that, in some datasets, wavelets may actually help in reducing noisy variations which deteriorate the performance of mining tasks. In a few cases, we note that the residual details/noises from DWT compression are more useful for recognizing data patterns.

Through innovatively taking advantage of mature techniques from both time series mining and NLP communities, we have brought up a language model-based approach for TSC named DSCo. Extensive experiments on an open dataset archive demonstrates that DSCo performs similarly to approaches (*e.g.*, DTW and Euclidean distance-based 1NN) working with original uncompressed numeric data. We further propose DSCo-NG to improve the computation efficiency and classification accuracy of DSCo. DSCo-NG breaks time series into smaller segments of the same size and to some extend simplifies DSCo, this simplification of the classification process also leads to simplified language model inference in the training phase and slightly higher classification accuracy.

Finally, as a consequence of extracting features from multiscale visibility graphs, we are able to transform sequential time series data into unordered feature vectors, so that these features can be fed into modern and generic classifiers for efficient and accuracy classification. Unlike traditional TSC approaches that seek to improve distance-based classification (e.g., 1NN with DTW distance) or to transform real-valued time series into texts (e.g., DSCo, SAX-VSM and BOSS), we take a very different path and augment time series by means of their multiscale approximations, which are further transformed into a set of visibility graphs. After extracting probability distributions of small motifs and other statistical features from such graphs, we are able to build reliable models that yield highly accurate classification. Thanks to the way how we transform time series into graphs and extract features from them, we are able to capture both global and local features from time series. Based on extensive experiments on a large number of open datasets and comparison with ten state-of-the-art approaches, our approach appears to be among the most accurate TSC algorithms and it is also shown to be highly efficient as well as scalable.

To sum up, this dissertation explores different possibilities to improve the efficiency and accuracy of TSC algorithms. To that end we employ a range of techniques including wavelet transforms, symbolic approximations, language models and graph mining algorithms. We experiment and evaluate our approaches using publicly available time series datasets. Comparison with the state-of-the-art shows that the approaches developed in this dissertation perform well, and contribute to advance the field of TSC. Although specific applications are not the focus of this dissertation, we have nonetheless applied TSC as well as clustering techniques in the field of smart buildings, demonstrating the feasibility of some of our proposed approaches for real-world applications.

9.2 FUTURE WORK

Obviously, it is not feasible to consider all related research topics in this dissertation. However, there are indeed a few research avenues that we are exited to explore in the future:

(a) In this dissertation we mainly covered classification for univariate time series. In reality, multivariate time series also have a large number of application scenarios. For instance, multivariate time series are often found in human activity detection and financial applications such as technical analysis of candle charts (OHLC charts). Theoretically, DSCo as well as MVG might also work for classifying multivariate time series, since multivariate time series can be considered a set of independent univariate series. However, it will remain a mystery unless empirically tested or theoretically proven. We are excited to experiment our proposed approaches for multivariate time series data in the future.

(b) We are delighted that graph representations can be used for TSC. However, we feel we have just scratched the surface in the direction of graph-based TSC. In the future, we are passionate to further investigate other graph representations for time series as well as useful and efficient graph features – such as degree distribution entropy, centrality, bipartivity and so forth – for MVG in order to improve its classification accuracy and efficiency. Thanks to the rapid advances in graph mining, a lot of graph features remain to be explored for TSC.

(c) During the past few years, ensemble methods have become increasingly popular in the TSC community. Although the MVG approach we have proposed in this dissertation is an ensemble method, we have only taken advantage of graph features. We are confident that when incorporating more features other than those extracted from visibility graphs, TSC accuracy can be further incremented. In the future, we plan to investigate and combine more heterogeneous features for TSC.

(d) As the performance of GPUs keep improving and deep neural network platforms mature, it is foreseeable that a large number of DM tasks will run on deep learning architectures. ALthough we have scratched the surface of making TSC suitable for deep learning architectures, we have not had great TSC performances. Thus it would be interesting to investigate how preprocessing and transformation techniques can be applied to time series so that efficient and accurate classification can be achieved via deep learning algorithms.

BIBLIOGRAPHY

P. S. Addison. Wavelet transforms and the ECG: a review. *Physiological measurement*, 26(5):R155, 2005. (Cited on pages 36 and 39.)

P. Afshani, M. de Berg, H. Casanova, B. Karsin, C. Lambrechts, N. Sitchinava, and C. Tsirogiannis. An efficient algorithm for the 1d total visibility-index problem. In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 218–231. SIAM, 2017. (Cited on pages 81 and 101.)

N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *ICDM*, pages 1–10, 2015. (Cited on pages 78, 84, and 101.)

M. Alzantot and M. Youssef. Crowdinside: automatic construction of indoor floorplans. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 99–108. ACM, 2012. (Cited on page 126.)

K. Amolins, Y. Zhang, and P. Dare. Wavelet based image fusion techniques – An introduction, review and comparison. *ISPRS Journal of Photogrammetry and Remote Sensing*, 62(4):249–263, 2007. (Cited on pages 37 and 39.)

K. Anderson, A. Ocneanu, D. Benitez, D. Carlson, A. Rowe, and M. Berges. Blued: A fully labeled public dataset for event-based non-intrusive load monitoring research. In *Proceedings of the 2nd KDD workshop on data mining applications in sustainability (SustKDD)*, pages 1–5, 2012. (Cited on pages 5, 35, and 53.)

A. Antonini, F. Maggi, and S. Zanero. A practical attack against a knx-based building automation system. In *Proceedings of the 2nd International Symposium on ICS & SCADA Cyber Security Research 2014*, pages 53–60. BCS, 2014. (Cited on page 127.)

A. Bagnall, J. Lines, J. Hills, and A. Bostrom. Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015. (Cited on pages 27 and 107.)

A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, Online First, 2016. (Cited on page 28.)

A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. 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. (Cited on page 92.)

A. Barbato, A. Capone, M. Rodolfi, and D. Tagliaferri. Forecasting the usage of household appliances through power meter sensors for demand management in the smart grid. In *IEEE International Conference on Smart Grid Communications*

(SmartGridComm), pages 404–409. IEEE, 2011. (Cited on page 114.)

K. Basu, V. Debusschere, and S. Bacha. Appliance usage prediction using a time series based classification approach. In *Thirty-Eighth Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 1217–1222. IEEE, 2012. (Cited on page 115.)

V. Batagelj and M. Zaversnik. An $o(m)$ algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003. (Cited on pages 78 and 88.)

G. E. Batista, X. Wang, and E. J. Keogh. A complexity-invariant distance measure for time series. In *SDM*, volume 11, pages 699–710, 2011. (Cited on pages 23, 78, and 115.)

M. G. Baydogan, G. Runger, and E. Tuv. A bag-of-features framework to classify time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11): 2796–2802, 2013. (Cited on pages 23, 24, and 74.)

J. R. Bellegarda. Statistical language model adaptation: review and perspectives. *Speech communication*, 42(1):93–108, 2004. (Cited on page 68.)

M. E. Berges, E. Goldman, H. S. Matthews, and L. Soibelman. Enhancing electricity audits in residential buildings with nonintrusive load monitoring. *Journal of industrial ecology*, 14(5):844–858, 2010. (Cited on page 115.)

D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370, 1994. (Cited on pages 5, 78, and 106.)

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (Cited on page 21.)

F. K.-P. Chan, A. W.-c. Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: with and without time warping. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3):686–705, 2003. (Cited on page 40.)

H. Chen, P. Chou, S. Duri, H. Lei, and J. Reason. The design and implementation of a smart building control system. In *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*, pages 255–262. IEEE, 2009. (Cited on page 126.)

L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005. (Cited on page 23.)

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016. (Cited on pages 21 and 91.)

Y. Chen, A. Why, G. Batista, A. Mafra-Neto, and E. Keogh. Flying insect classification with inexpensive sensors. *Journal of Insect Behavior*, 5(27):657–677, 2014. (Cited on page 3.)

Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`. (Cited on pages 15, 28, 40, 59, 60, 69, 91, and 116.)

F.-L. Chung, T.-C. Fu, R. Luk, and V. Ng. Flexible time series pattern matching based on perceptually important points. In *International joint conference on artificial intelligence workshop on learning from temporal and spatial data*, pages 1–7, 2001. (Cited on page 74.)

A. Cohen, I. Daubechies, and P. Vial. Wavelets on the interval and fast wavelet transforms. *Applied and computational harmonic analysis*, 1(1):54–81, 1993. (Cited on pages 36 and 89.)

L. d. F. Costa, F. A. Rodrigues, G. Travieso, and P. R. Villas Boas. Characterization of complex networks: A survey of measurements. *Advances in physics*, 56(1): 167–242, 2007. (Cited on pages 88 and 107.)

S. Darby. Making it obvious: designing feedback into energy consumption. In *Energy efficiency in household appliances and lighting*, pages 685–696. Springer, 2001. (Cited on page 113.)

I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 41(7):909–996, 1988. (Cited on page 36.)

I. Daubechies. Orthonormal bases of compactly supported wavelets ii. variations on a theme. *SIAM Journal on Mathematical Analysis*, 24(2):499–519, 1993. (Cited on page 36.)

H. Deng, G. Runger, E. Tuv, and M. Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013. (Cited on page 27.)

C. Duarte, P. Delmar, K. W. Goossen, K. Barner, and E. Gomez-Luna. Non-intrusive load monitoring based on switching voltage transients and wavelet transforms. In *Future of Instrumentation International Workshop (FIIW), 2012*, pages 1–4. IEEE, 2012. (Cited on pages 36 and 39.)

O. J. Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964. (Cited on page 97.)

E. Elhamifar and S. Sastry. Energy disaggregation via learning 'powerlets' and sparse coding. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. (Cited on pages 114 and 115.)

P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):12, 2012. (Cited on page 4.)

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res*, 15(1):3133–3181, 2014. (Cited on pages 21, 77, 90, and 97.)

R. Flanagan and L. Lacasa. Irreversibility of financial time series: a graph-theoretical approach. *Physics Letters A*, 380(20):1689–1697, 2016. (Cited on page 3.)

E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, and L. Trigg. Weka: a machine learning workbench for data mining. In *Data mining and knowl-*

edge discovery handbook, pages 1269–1277. Springer, 2009. (Cited on page 28.)

Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995. (Cited on page 21.)

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. (Cited on page 22.)

T.-C. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011. (Cited on pages 35, 53, 54, and 74.)

J. Grabocka, N. Schilling, M. Wistuba, and L. 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. (Cited on pages 25, 26, 78, 100, and 107.)

A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013. (Cited on page 22.)

M. Gray and W. Morsi. Application of wavelet-based classification in non-intrusive load monitoring. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pages 41–45. IEEE, 2015. (Cited on pages 36 and 39.)

G. W. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80 (12):1870–1891, 1992. (Cited on page 114.)

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (Cited on pages 22 and 83.)

J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4): 851–881, 2014. (Cited on pages 26, 66, and 107.)

C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. 2003. (Cited on page 30.)

B. Hu, Y. Chen, and E. Keogh. Time series classification under more realistic assumptions. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 578–586. SIAM, 2013. (Cited on page 78.)

J. Iacovacci and L. Lacasa. Sequential motif profile of natural visibility graphs. *Physical Review E*, 94(5):052309, 2016. (Cited on page 82.)

Y. Jiang, Y. Xiang, X. Pan, K. Li, Q. Lv, R. P. Dick, L. Shang, and M. Hannigan. Hallway based automatic indoor floorplan construction using room fingerprints. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 315–324. ACM, 2013. (Cited on page 126.)

M. Jin, N. Bekiaris-Liberis, K. Weekly, C. Spanos, and A. M. Bayen. Sensing by proxy: Occupancy detection based on indoor co2 concentration. In *The 9th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technolo-*

gies (UBICOMM'15), pages 1–10, 2015. (Cited on page 127.)

C. Kang, N. Park, B. A. Prakash, E. Serra, and V. Subrahmanian. Ensemble models for data-driven prediction of malware infections. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 583–592. ACM, 2016. (Cited on page 3.)

E. Keogh. Fast similarity search in the presence of longitudinal scaling in time series databases. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pages 578–584. IEEE, 1997. (Cited on pages 12, 74, and 80.)

E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005. (Cited on page 14.)

E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001. (Cited on pages 53 and 74.)

E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2004. (Cited on pages 28, 69, and 74.)

E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000. (Cited on pages 12 and 80.)

R. Kondor and H. Pan. The multiscale laplacian graph kernel. In *Advances in Neural Information Processing Systems*, pages 2990–2998, 2016. (Cited on page 107.)

L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 17:1–5, 2016. (Cited on page 31.)

M. Kuhn and K. Johnson. *Applied predictive modeling*, volume 810. Springer, 2013. (Cited on pages 30 and 91.)

L. Lacasa, B. Luque, F. Ballesteros, J. Luque, and J. C. Nuno. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, 2008. (Cited on page 81.)

L. Lacasa, B. Luque, J. Luque, and J. C. Nuno. The visibility graph: A new method for estimating the hurst exponent of fractional brownian motion. *EPL (Europhysics Letters)*, 86(3):30001, 2009. (Cited on page 81.)

C. Laughman, K. Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong. Power signature analysis. *Power and Energy Magazine, IEEE*, 1(2):56–63, 2003. (Cited on page 115.)

A. K.-m. Leung, F.-t. Chau, and J.-b. Gao. A review on applications of wavelet transform techniques in chemical analysis: 1989–1997. *Chemometrics and Intelli-*

gent Laboratory Systems, 43(1):165–184, 1998. (Cited on page 39.)

D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. DSCo-NG: A Practical Language Modeling Approach for Time Series Classification. In *The 15th International Symposium on Intelligent Data Analysis (IDA 2016)*, October 2016a. (Cited on pages 7 and 40.)

D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. Time Series Classification with Discrete Wavelet Transformed Data. In *International Journal of Software Engineering and Knowledge Engineering*, volume 26, pages 1361–1377. World Scientific, November & December 2016b. doi: 10.1142/S0218194016400088. (Cited on page 7.)

D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. Time Series Classification with Discrete Wavelet Transformed Data: Insights from an Empirical Study. In *The 28th International Conference on Software Engineering and Knowledge Engineering (SEKE 2016)*, pages 273–278, July 2016c. (Cited on pages 7, 36, 47, and 74.)

D. Li, T. F. Bissyandé, S. Kubler, J. Klein, and Y. Le Traon. Profiling Household Appliance Electricity Usage with N-Gram Language Modeling. In *The 2016 IEEE International Conference on Industrial Technology (ICIT 2016)*, pages 604–609. IEEE, March 2016d. (Cited on pages 7 and 40.)

D. Li, L. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. DSCo: A Language Modeling Approach for Time Series Classification. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition: 12th International Conference, MLDM 2016, New York, NY, USA*, pages 294–310. Springer International Publishing, July 2016e. ISBN 978-3-319-41920-6. doi: 10.1007/978-3-319-41920-6_22. (Cited on pages 7 and 40.)

D. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon. Sensing by Proxy in Buildings with Agglomerative Clustering of Indoor Temperature Movements. In *The 32nd ACM Symposium on Applied Computing (SAC 2017)*, pages 477–484, April 2017. doi: 10.1145/3019612.3019699. (Cited on page 8.)

D. Li, J. Lin, T. F. Bissyandé, J. Klein, and Y. Le Traon. Extracting Statistical Graph Features for Accurate and Efficient Time Series Classification. In *The 21st International Conference on Extending Database Technology (EDBT)*, March (To Appear) 2018. (Cited on page 8.)

Y. Li and J. Lin. Approximate variable-length time series motif discovery using grammar inference. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, page 10, 2010. (Cited on page 16.)

T. W. Liao. Clustering of time series data – a survey. *Pattern recognition*, 38(11): 1857–1874, 2005. (Cited on pages 4 and 125.)

J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003. (Cited on page 16.)

J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007. (Cited on pages 12, 15, 16, 36, 54, 60, 62, 78, 80, and 106.)

J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315, 2012. (Cited on pages 23, 78, and 106.)

J. Lines, A. Bagnall, P. Caiger-Smith, and S. Anderson. Classification of household devices by electricity usage profiles. In *Intelligent Data Engineering and Automated Learning-IDEAL 2011*, pages 403–412. Springer, 2011. (Cited on pages 114, 115, and 118.)

J. Lines, L. M. Davis, J. Hills, and A. 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. ACM, 2012. (Cited on page 26.)

J. Lines, S. Taylor, and A. Bagnall. Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 1041–1046. IEEE, 2016. (Cited on page 27.)

B. Luque, L. Lacasa, F. Ballesteros, and J. Luque. Horizontal visibility graphs: Exact results for random time series. *Physical Review E*, 80(4):046103, 2009. (Cited on page 81.)

P.-F. 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. (Cited on pages 23 and 106.)

R. Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research*, 15(1):3735–3739, 2014. (Cited on page 31.)

J. Morales Pedraza. *Electrical Energy Generation in Europe: The Current Situation and Perspectives in the Use of Renewable Energy Sources and Nuclear Power for Regional Electricity Generation*. Springer, 2015. (Cited on page 113.)

A. Mueen, E. Keogh, and N. Young. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1154–1162. ACM, 2011. (Cited on pages 26 and 115.)

A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10(3), 2001. (Cited on page 115.)

P. Nemenyi. Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263, 1962. (Cited on page 47.)

M. E. Newman and M. Girvan. Mixing patterns and community structure in networks. *Statistical mechanics of complex networks*, pages 66–87, 2003. (Cited on

pages 78 and 88.)

T. A. Nguyen and M. Aiello. Energy intelligent buildings based on user activity: A survey. *Energy and buildings*, 56:244–257, 2013. (Cited on pages 123 and 124.)

P. Norvig. Natural language corpus data. In T. Segaran and J. Hammerbacher, editors, *Beautiful data: the stories behind elegant data solutions*, pages 219–242. O'Reilly Media, Inc., 2009. (Cited on page 61.)

B. Okorn, X. Xiong, B. Akinci, and D. Huber. Toward automated modeling of floor plans. In *Proceedings of the symposium on 3D data processing, visualization and transmission*, volume 2, 2010. (Cited on page 126.)

N. D. Pham, Q. L. Le, and T. K. Dang. Two novel adaptive symbolic representations for similarity search in time series databases. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 181–187. IEEE, 2010. (Cited on page 17.)

A. Pizurica, A. M. Wink, E. Vansteenkiste, W. Philips, and B. J. Roerdink. A review of wavelet denoising in MRI and ultrasound brain imaging. *Current medical imaging reviews*, 2(2):247–260, 2006. (Cited on pages 36 and 39.)

J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281, 1998. (Cited on page 55.)

T. Rakthanmanon and E. Keogh. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *Proceedings of the thirteenth SIAM conference on data mining*, 2013. (Cited on pages 25, 26, 78, 100, 101, 107, and 115.)

T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012. (Cited on pages 5, 14, 35, 40, 47, 78, and 106.)

K. J. Åström. On the choice of sampling rates in parametric identification of time series. *Information Sciences*, 1(3):273–278, 1969. (Cited on pages 12 and 80.)

C. A. Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004. (Cited on page 3.)

C. A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *Proceedings of SIAM International Conference on Data Mining*, pages 506–510, 2005. (Cited on pages 106 and 115.)

P. Ribeiro, F. Silva, and L. Lopes. Efficient parallel subgraph counting using g-tries. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 217–226. IEEE, 2010. (Cited on page 84.)

J. J. Rodríguez and C. J. Alonso. Interval and dynamic time warping-based decision trees. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages

548–552. ACM, 2004. (Cited on page 115.)

J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006. (Cited on page 21.)

P. Royston et al. Multiple imputation of missing values. *Stata journal*, 4(3):227–41, 2004. (Cited on page 128.)

S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007. (Cited on page 40.)

K. Samiee, P. Kovacs, and M. Gabbouj. Epileptic seizure classification of eeg time-series using rational discrete short-time fourier transform. *IEEE transactions on Biomedical Engineering*, 62(2):541–552, 2015. (Cited on page 3.)

P. Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015. (Cited on pages 25, 27, and 106.)

P. Schäfer and M. Högqvist. Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 516–527. ACM, 2012. (Cited on page 17.)

G. Schrack and M. Choit. Optimized relative step size random searches. *Mathematical Programming*, 10(1):230–244, 1976. (Cited on page 30.)

P. Senin and S. Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *IEEE 13th International Conference on Data Mining*, pages 1175–1180. IEEE, 2013. (Cited on pages 24, 74, 78, 100, and 106.)

P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, S. Frankenstein, and M. Lerner. Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In *Machine Learning and Knowledge Discovery in Databases*, pages 468–472. Springer, 2014. (Cited on page 16.)

J. Serra and J. L. Arcos. A competitive measure to assess the similarity between two time series. In *Case-Based Reasoning Research and Development*, pages 414–427. Springer, 2012. (Cited on pages 23 and 106.)

J. Serra and J. L. Arcos. An empirical evaluation of similarity measures for time series classification. *Knowledge-Based Systems*, 67:305–314, 2014. (Cited on pages 23 and 40.)

J. Shieh and E. Keogh. iSAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008. (Cited on page 17.)

O. Shih and A. Rowe. Occupancy estimation using ultrasonic chirps. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 149–158. ACM, 2015. (Cited on page 127.)

H. Shin, Y. Chon, and H. Cha. Unsupervised construction of an indoor floor plan using a smartphone. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):889–898, 2012. (Cited on page 126.)

J. Sill, G. Takács, L. Mackey, and D. Lin. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460*, 2009. (Cited on page 96.)

A. A. Smith and M. Craven. Fast multisegment alignments for temporal expression profiles. In *Proceedings of the 7th Annual International Conference on Computational Systems Bioinformatics*, volume 7, pages 315–326. World Scientific, 2008. (Cited on pages 14 and 65.)

S. Supriya, S. Siuly, H. Wang, J. Cao, and Y. Zhang. Weighted visibility graph with complex network features in the detection of epilepsy. *IEEE Access*, 4:6554–6566, 2016. (Cited on pages 81 and 107.)

D. S. Taubman and M. W. Marcellin. JPEG2000: Standard for interactive imaging. *Proceedings of the IEEE*, 90(8):1336–1357, 2002. (Cited on page 36.)

S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic hpc cluster: The ul experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967. IEEE, July 2014. (Cited on pages 41, 83, and 116.)

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. (Cited on page 61.)

Q. Wang and V. Megalooikonomou. A dimensionality reduction technique for efficient time series similarity analysis. *Information systems*, 33(1):115–132, 2008. (Cited on page 36.)

X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013. (Cited on page 74.)

X. Wang, J. Lin, N. Patel, and M. Braun. A self-learning and online algorithm for time series anomaly detection, with application in cpu manufacturing. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1823–1832. ACM, 2016a. (Cited on page 3.)

X. Wang, J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. Rpm: Representative pattern mining for efficient time series classification. In *Proceedings of the 19th International Conference on Extending Database Technology*, 2016b. (Cited on pages 24, 70, 74, 78, 100, 101, and 106.)

Z. Wang and T. Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. (Cited on page 83.)

J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963. (Cited on page 125.)

T. K. Wijaya, J. Eberle, and K. Aberer. Symbolic representation of smart meter data. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 242–248. ACM, 2013. (Cited on page 53.)

M. Wojnowicz, G. Chisholm, B. Wallace, M. Wolff, X. Zhao, and J. Luan. Suspend: Determining software suspiciousness by non-stationary time series modeling of entropy signals. *Expert Systems with Applications*, 71:301–318, 2017. (Cited on pages 3 and 77.)

D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992. (Cited on page 96.)

D. F. Wong, L. S. Chao, X. Zeng, M.-I. Vai, and H.-L. Lam. Time series for blind biosignal classification model. *Computers in biology and medicine*, 54:32–36, 2014. (Cited on page 3.)

X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008. (Cited on page 20.)

Y. Wu and E. Y. Chang. Distance-function design and fusion for sequence data. In *Proceedings of the 13th ACM international conference on Information and knowledge management*, pages 324–333. ACM, 2004. (Cited on page 115.)

X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM, 2006. (Cited on pages 5, 53, and 74.)

X. Xu, J. Zhang, and M. Small. Superfamily phenomena and motifs of networks induced from time series. *Proceedings of the National Academy of Sciences*, 105(50): 19601–19605, 2008. (Cited on pages xviii, 86, and 107.)

J. Yang, M. N. Nguyen, P. P. San, X. Li, and S. Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *IJCAI*, pages 3995–4001, 2015. (Cited on page 107.)

L. Yang, K. Ting, and M. B. Srivastava. Inferring occupancy from opportunistically available sensor data. In *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*, pages 60–68. IEEE, 2014. (Cited on pages 126 and 127.)

L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956. ACM, 2009. (Cited on pages 23, 25, 78, 107, and 115.)

M. Zeifman and K. Roth. Nonintrusive appliance load monitoring: Review and outlook. *IEEE Transactions on Consumer Electronics*, pages 76–84, 2011. (Cited on pages 114 and 115.)

G. P. Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003. (Cited on page 4.)

Z.-N. Zhen, Q.-S. Jia, C. Song, and X. Guan. An indoor localization algorithm for lighting control using rfid. In *Energy 2030 Conference, 2008. ENERGY 2008. IEEE*, pages 1–6. IEEE, 2008. (Cited on page 126.)

S. Zhong, T. M. Khoshgoftaar, and N. Seliya. Clustering-based network intrusion detection. *International Journal of reliability, Quality and safety Engineering*, 14(02): 169–187, 2007. (Cited on page 4.)

K. Zhu, Y. San Wong, and G. S. Hong. Wavelet analysis of sensor signals for tool condition monitoring: a review and some new results. *International Journal of Machine Tools and Manufacture*, 49(7):537–553, 2009. (Cited on page 39.)

T. Zhu, S. Xiao, Q. Zhang, Y. Gu, P. Yi, and Y. Li. Emergent technologies in big data sensing: a survey. *International Journal of Distributed Sensor Networks*, 2015:8, 2015. (Cited on page 126.)

J.-P. Zimmermann, M. Evans, J. Griggs, N. King, L. Harding, P. Roberts, and C. Evans. Household electricity survey: A study of domestic electrical product usage. *Intertek Testing & Certification Ltd*, 2012. (Cited on page 117.)