APPROVAL SHEET

Title of Dissertation: Topological Time-series Classification

Name of Candidate: Joseph Robert Collins
Doctor of Philosophy, 2022

Dissertation and Abstract Approved: _____

Dr. David Chapman
Assistant Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

ABSTRACT

Title of dissertation: TOPOLOGICAL TIME-SERIES CLASSIFICATION

Joseph Robert Collins, Doctor of Philosophy, 2022

Dissertation directed by: David Chapman
Department of Computer Science and
Electrical Engineering

We establish a strong connection between Topological Data Analysis (TDA) and the field of time-series classification. This is accomplished via two novel contributions. The first is a method for extracting topological information from a time series in a manner which circumvents the expensive computation normally required. The second is an adaption of an existing topological summary to create an efficient topological time-series pooling operator. Taken together, these insights enable a new classifier, TopRocket (**Top**ological **R**and**o**m **C**onvolutional **Ke**rnel **T**ransform). Using standard comparison methods, TopRocket ranks second among all univariate time-series classifiers. Further, TopRocket is the best performing scalable classifier. As a final contribution, we provide the fastest available implementation of the algorithm for computing time-series sublevel set persistence. This provides a foundation for further investigation of TDA for time-series classification.

This work lies at the intersection of TDA and machine learning, where success is dependent on efficiently extracting topological information from data and

using that information with downstream models. For this purpose, we develop an efficient, data-based, topological summary, the Betti pooling operator. We rigorously demonstrate the utility of TopRocket by evaluating its performance against the University of California Riverside (UCR) Time Series Archive, a widely used benchmark collection of data sets. TopRocket is the first TDA-based classifier to obtain state-of-the art results on any similarly competitive problem. We strongly believe that TDA will play a central role in the future of machine learning.

TOPOLOGICAL TIME-SERIES CLASSIFICATION

by

Joseph Robert Collins

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Dr. David Chapman, Chair/Advisor
Professor Samuel Lomonaco
Dr. Rita Doerr
Dr. Christopher Marron
Dr. Don Engel
Professor Anindya Roy

Dedication

For my grandfather RCB Jr., a great American. I learned a few things about the wavy lines.

Acknowledgments

I owe the completion of this dissertation to many people. I am grateful for the researchers who have cleared a path, the teachers and mentors who have guided me, my family for their immense patience and support, and God for the continued blessings in my life.

This work would not have been possible without the open and welcoming environment fostered by members of both the Topological Data Analysis and time-series classification communities. The Applied Algebraic Topology Network allowed me to undertake novel work in TDA. The resources provided by the network explain the core concepts of TDA in approachable terms. I would like to thank all the organizers of AATRN, and specifically Henry Adams.

Time series classification has a similarly outstanding community. The maintainers of the UCR Time Series Archive made this work possible by striving to provide a fair benchmark. The archive maintainers are welcoming of dataset contributions, which may be the strongest way to support their work. I did not produce any datasets as part of this work, but if I did I would gladly share with the archive.

The lab of Dr. Geoff Webb at Monash University has consistently provided high quality results and open-source code for state-of-the-art classifiers. The next four classifiers after TopRocket are from his lab. I would specifically like to thank Angus Dempster and Chang Wei Tan for publicly providing open-source Rocket family classifiers. Their work directly enables TopRocket.

I have had many teachers and mentors support me throughout my education.

Amy Brunner and Anna DiGiulian helped to set me on the path towards computer science in high school, and I am very grateful to them.

At UMBC, I am thankful to all those who made the school what it is today. Specifically, Dr. Freeman Hrabowski, who created an environment of excellence by putting the needs of the students first.

Dr. Christopher Marron was my advisor for my M.S. and introduced me to TDA. I am very thankful he did! Dr. Samuel Lomonaco opened my eyes to the world of applied abstract algebra. His classes gave me a fresh perspective on computer science, and our discussions have helped me to think of TDA in a broader sense. I am also grateful to have been his teaching assistant for several years. My PhD advisor, Dr. David Chapman, gave me the freedom to explore my research interests, a rare gift for students. He also provided insights and guidance when needed. Dr. Rita Doerr has been an exceptional mentor, encouraging me throughout my studies and providing invaluable feedback.

None of this would have been possible without the continued support and patience of my family. My late grandfather, Bob, described the Fourier Transform to me when I was only 15 years old while we were watching waves at the beach. I did not really understand it at the time, but the idea captivated me, as he knew it would, and has ever since. From an early age, he impressed upon all of his grandchildren the value of higher education. College is an opportunity he was never afforded, but it did not stop him from achieving great things. As my formal education draws to a close, I plan to honor his memory by striving for the same.

I am also grateful to my grandmother, Angie, for always lending an ear and

encouraging me. She has consistently supported me and my education. She reminds me to relax a little and be patient, especially when I had to work after Sunday dinners (nothing should come before cards with the family).

My dad, Mike, has always taken an interest in my interests and has been there every step of the way. He taught me to solder, which led to my first robots, and the realization that, with enough patience, I can control all sorts of gadgets and machines!. He spent more than a few nights with my high school robotics team, helping in every way he could. His support and encouragement have kept me on track.

My mom, Lisa, was my first teacher and continues to be my best teacher. She taught me the value of hard work, patience, and remaining humble. Her kindness and love have made me the person I am today. She has been patient beyond measure, and is the reason I have been able to complete this dissertation. She has been present and involved throughout my education, always making sure I have all the tools I need. She is responsible for homeschooling my brother and me during middle school. It is an experience I am very grateful for, as it taught me the value of self-study at a young age. In high school, Mom encouraged my early interest in research by letting me skip class to attend talks at UMBC (if anyone asks, I had "headaches" those days). She would wait in the car while I learned about subjects which were way over my head at the time. It turned out to be the right call!

My brother, Michael, has pushed me to continually improve. His example helped to clear a path for me, both through my education and through life. I could not ask for anything more from a brother or, more importantly, a role model.

Most importantly, I am grateful to God for the blessings and opportunities in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Topological data analysis for time-series classification is not an immediately obvious or intuitive choice. The line of work presented here originated for the purposes of network analysis and cybersecurity applications. Specifically, it began as an investigation into the use of persistent homology over topological spaces formed by computer networks. That work, in turn, was motivated by promising network anomaly detection results over the Enron email dataset [6] [7]. The core idea behind that work is to identify major events in the Enron scandal time-line, based only on the email graph of who communicated with whom in a given week. The use of persistent homology over a derived topological space of the communication graph provides accurate results.

That motivated further investigation of using TDA for applications where the data is naturally represented as a graph. One such domain is cybersecurity, where such techniques may be able to detect various types of network failures or cyber attacks. The initial goal was to analyze network anomalies in a broad range of computer networks. However, in many environments it is not possible to have complete visibility into the full network topology. Therefore, the work was restricted to analyzing network traffic which passes through key ingress and egress points.

The data available at these points is not naturally represented as a graph, and more closely resembles a times series. Given the past success of TDA for the Enron problem, it was decided to also apply persistent homology to this resulting time-series classification problem. This led directly to highly competitive results for identifying internet of things (IOT) devices from encrypted traffic [5]. Specifically, the TDA-based model classifies solely based on the persistent homology of the inter-packet arrival time of packets passing through a router. Inter-packet arrival time is always available, regardless of encryption. The sequence of inter-packet arrival times forms a time series, which is then classified. This result was obtained by classifying the persistence images of the sub-level set persistence of the time-series using a convolutional neural network.

Given this initial success, it is a natural next step to investigate TDA for general time-series classification. Ideally, a time-series classifier should be domain agnostic. The datasets and statistical framework used for this task are provided by the University of California Riverside (UCR) time-series archive. This archive provides a common reference point against which to compare competing classifiers. This is the motivating problem for the work presented here, maximizing performance on the UCR archive.

1.1 Challenges

There are several key challenges which accompany this line of research. A few of these challenges are unique to TDA or time series classification, while others arise

at the intersection of both. TopRocket overcomes each of these challenges.

1.1.1 How Should Time-series Classifiers be Compared?

The cornerstone of modern time-series classification research is the UCR archive. There are two reasons this archive is useful. First, the common datasets ensure that competing classifiers are performing an identical task. Second, the common comparison methods ensure that classification results over the entire archive are fairly compared. The datasets contained in the archive and the accompanying comparison methods are explored in Section 2.1.

1.1.2 How Can Topological Information be Efficiently Extracted?

A fundamental tool of TDA is persistent homology, which intuitively gives a notion of shape across scales. Persistent homology has many convenient properties, explored in Section 2.3. In the general case, the naive algorithm for computing persistent homology runs in $O(N^3)$ time, which is too slow for time series classification. It is possible to compute features of the persistent homology of a time series, without explicitly computing the fully persistent homology. One such useful approach is presented in Section 3.2. For classifiers which require the full persistent homology of a time series, we present the fastest known algorithm implementation for computing it in Appendix A.2.

1.1.3 How Can Topological Information be Compared?

The most natural representation of persistent homology is as a multiset of tuples, representing the half-open intervals where homology classes are born and die, called a persistence diagram. Unfortunately, this representation is not easily compared. The Bottleneck and Wasserstein distances are the most intuitive methods for comparing diagrams, and are detailed in Section 2.3.4. These distances are inefficient to compute, due to a brute force bipartite graph matching problem which must be solved as part of computing the distance.

There are many ways to compare persistence diagrams without using these distances. Most methods involve transforming the persistence diagram in to a vectorization or representation which can be compared more efficiently. These transformations are covered in Section 2.3.5. Through a detailed understanding of how these transformations are computed, it is possible, in some cases, to directly compute the transformation from the data, without first finding the persistent homology.

1.1.4 How Can TDA be used to Augment Existing Classifiers?

Given a good vectorization of persistence diagrams, it is straightforward to create classifiers using kernel and clustering methods. Some vectorizations are especially well suited for neural networks. However, over the course of evaluating such approaches, two main issues arise. First, computing the persistent homology and vectorization is too slow. Second, these purely topological classifiers do not perform well, as shown in Section 3.1.

We address the first issue by optimizing tools from TDA for time series. Specifically, we provided the fastest implementation, to our knowledge, of the algorithm for finding the 0-dimensional sublevel-set persistence of a time series. We avoid the need to compute full persistence, when it can be avoided, by exploiting properties of a given persistence representation. We overcome the second issue by using topological information to augment existing classifiers designed for time-series classification. The competition over the UCR archive is fierce and the field is mature, so it is reasonable to make use of existing work.

1.2 Thesis Statement

This research will develop topological, univariate, time-series classification methods which are fast, accurate, reproducible, and rigorously evaluated.

1.3 Objectives

The primary goal of this work is to explain how and why TopRocket works. The secondary, and almost as important goal, is to explain the attempts which ultimately motivate the development of TopRocket. These attempts are guided by the overarching belief that topological features provide information which is orthogonal, in some sense, to the features used by existing classifiers. In other words, TDA can provide an edge.

The design of TopRocket is not arbitrary. In the search for topological features useful for general time-series classification, many attempts are made using a

wide variety of techniques. Ultimately, the Betti curves used in TopRocket perform exceptionally well, while the other methods do not outperform MultiRocket. As we will demonstrate, these other topological classifiers are not competitive on the UCR dataset. It is our belief that TDA has more to offer machine learning, especially in the context of time-series classification.

1.4 Contributions

The primary contribution of this work is TopRocket, which is the first TDA-based classifier to achieve state-of-the-art results on any similarly competitive problem. Two innovations enable TopRocket. The first is the insight that information about the persistent homology of a time series can be extracted without explicitly computing the persistent homology. Rocket classifiers are known for speed, so maintaining that speed is a primary constraint, behind maximizing accuracy. The second key novel idea is computing the Betti curve of a time series at fixed values derived from the underlying data. The Betti curve is a convenient representation of persistent homology, presented in Section 2.3.5.1. In past work, Betti curves are scaled according to the persistence information of the underlying data, but never explicitly compute a select set of data-based values. These two insights combine to make TopRocket possible.

TopRocket shortcuts the calculation of persistent homology by computing values of the Betti curve directly from a given time series. This improves accuracy while maintaining speed and providing a firm basis in Topological Data Analysis.

The Betti curve is a special case of a similar construction called the persistence curve, explained in Section 2.3.7. All other known persistence curves require calculation of the full persistent homology of a time series. In order to enable further research in this direction, we also present the fastest known implementation of the algorithm for computing the persistent homology of a time series.

1.5 Overview

We begin by presenting the necessary background in time-series classification and topological data analysis in Chapter 2. This includes a description of the time-series classification problem, the UCR archive, and the associated statistical machinery. Accuracy can be used to compare classifier performance over a single dataset. Over many datasets, comparison is more difficult and greater care must be taken.

We then provide an overview of the useful tools from TDA. These tools include persistent homology and its associated constructions. Additionally, the relevant persistence diagram vectorizations and representations are defined. The persistence curve framework is presented, as it is a generalization of the Betti curve used in TopRocket.

With the proper foundation in place, we build upon the existing research in Chapter 3. The novel techniques attempted in this section are first motivated by experiments over a wide array of existing topological methods. These experiments, failing to provide sufficient accuracy, provide the motivation for hybrid topological

methods. Properly motivated, hybrid topological Rocket classifiers are presented in Section 3.2. Most of these classifiers fail to provide the desired boost in accuracy. Still, the creation of these classifiers involves the development of algorithms which may be of some interest. Additionally, the results are useful for attempting a similar line of investigation. Chapter 3 concludes with a description of TopRocket and its variations.

We finish with Chapter 4, which aims to accomplish two things. First, we present a call to action for TDA and time-series classification researchers alike. On the TDA side, techniques from time-series related papers are not compared using the rigorous methods developed for the UCR archive. While the existing body of TDA work related in this domain lays the ground work for TopRocket, future research should aim to meet the standards of the UCR archive. On the time-series classification side, TDA provides a rich and mostly untapped set of tools which appear to be especially useful in this domain. Second, several of the research paths remain unexplored, but would absolutely be worthy of further investigation.

Chapter 2: Background and Related Work

We begin by reviewing the UCR archive and the associated comparison techniques. The archive sets the standard of rigor in the field of time-series classification. We then review the significant classifiers in the field. These include the baseline and the state-of-the-art classifiers. The selected classifiers are chosen as a basis for later discussion and experiments. Additionally, the set of classifiers selected forms a good foundation for further investigation.

We conclude this section with a review of TDA. Our goal is to provide an intuition for the tools of TDA, and motivate why the tools might be useful for time-series classification. TDA is deeply rooted in mathematics, with a rich body of highly active research. We review only what is necessary for understanding the topological classifiers in Chapter 3.

2.1 UCR Time Series Archive

The current univariate UCR archive contains 128 datasets from a wide variety of domains [8]. All datasets, and information regarding the datasets, are publicly available [9]. The goal of the archive is to provide a common benchmark against which to compare classifiers. There is an open solicitation from the maintainers of

the archive for additional dataset contributions.

The original archive is much smaller, but has grown over the years to its current size. As the archive changes, the variety and nature of the datasets change as well. Most significantly, in the early iterations, all time series are z-normalized. This is properly motivated in most contexts, but is still a pre-processing decision independent of any classifier operations. The latest additions to the archive are not z-normalized. For the classifiers presented here, it can be assumed that the time series are processed as is, without any additional pre-processing, unless otherwise noted.

2.1.1 Methods of Comparison

Given a single dataset, it is straightforward to compare different classifiers using accuracy or related statistics. However, over a large number of datasets the methods of comparison and evaluation are less clear. The following methods assess the performance of classifiers over the entire archive.

2.1.1.1 Critical Difference Diagrams

A critical difference diagram compares a set of classifiers across the entire archive. The construction of the critical difference diagram is carried out in two main steps. First, the average rank of each classifier is calculated. For each dataset, the rank is calculated by ordering classifiers according to accuracy with lower being better (i.e. 1.0 is first place). Ties are broken with fractional ranks, so two classifiers

tied for second place would both be ranked 2.5. For each classifier, the average is taken over all datasets to obtain the average rank. This average rank is plotted along a horizontal scale, as shown in Figure 2.1.

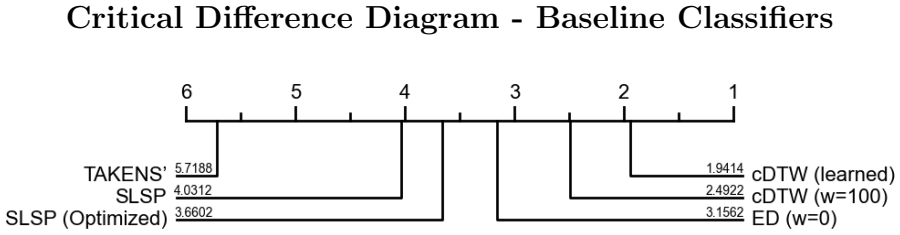

Figure 2.1: The baseline UCR archive classifiers are compared. There are no black bars connecting the classifiers, indicating that each classifier passed all pairwise significance tests.

Next, the statistical significance of the ranking is calculated. It is possible that one classifier might have a better rank than another, without significantly out performing it in terms of accuracy on any dataset. Significance is tested pairwise using the Wilcoxon Signed Rank test with Holm correction and $\alpha = .05$ [10, 11]. Visualizing the classifiers as nodes, classifiers are connected by an edge when the test fails. This leads to the notion of statistically similar cliques, which are represented as black bars in the critical difference diagram.

The black bars indicate that the connected classifiers are not statistically significantly different. While convenient for comparison at a glance, the bars require a nuanced interpretation in some cases when there is significant overlap. Additionally, the ranking is dependent on the classifiers included in the analysis. Specifically, it is possible that cliques do not respect the average rank order, in which case bars will cross over classifiers not included the clique. This situation is always made clear by

the additional bars in the diagram, and will always be accompanied by additional explanation. All critical difference diagrams presented here are created using an eponymous standard open-source library [12].

2.1.1.2 Win/Tie/Loss Graph

In many cases, it is useful to directly compare the performance of two classifiers in a more detailed way. This presents difficulty in comparing performance across all datasets. For this comparison, the Win/Tie/Loss graph is the standard method. Points are plotted for each data set, with accuracy for the first and second competing classifiers along the horizontal and vertical axis, respectively. Additionally, the Win/Tie/Loss count with respect to the first classifier is included as the title.

An example of this method is shown in Figure 3.7. Any points below the diagonal indicate datasets in which the first classifier performed better and points above indicate the second classifier is better. Additionally, lines are included $y = x - .05$ and $y = x + .05$ to make it clear when there is a difference in accuracy of over 5%.

2.1.1.3 Texas Sharpshooter Plot

This plot is particularly relevant for new time-series distances used in conjunction with k-nearest neighbor (k-NN) classifiers. It takes its name from the Texas Sharpshooter Fallacy, in which a sharpshooter shoots randomly at a blank wall and then draws targets around each hole after the fact. By analogy, not formulating a

hypothesis, prior to testing, calls the results into question.

In the context of time-series classification, it is not enough that a new distance performs better, it must also be known that it will perform better before evaluation over the test set. Knowing when a classifier will perform well is critical for applications to new datasets. This is one method of preventing overfitting to the archive. Another method, discussed in Section 2.2.3.1, is to only develop the classifiers over a subset of the archive.

For time-series classifiers, the hypothesis is taken as the expected accuracy gain over a baseline classifier. Expected accuracy is the leave-one-out accuracy of the classifier over the training set. Actual accuracy is the classifier accuracy over the test set. Accuracy gain is the ratio of the given accuracy (expected or actual) to the baseline actual accuracy.

Definition 2.1 (Accuracy Gain) $G = A_{new}/A_{baseline}$

The Texas Sharpshooter Plot is formed by plotting the actual accuracy gain against the expected accuracy gain, for each dataset. Ideally, all points would fall in the blue quadrants shown in Figure 3.2, indicating that the performance of the new classifier is predictable.

2.2 Time-series classification

The field of time-series classification is highly competitive with new state-of-the-art classifiers being published on a regular basis. In the past, it was exceptionally difficult to evaluate the general performance of different classifiers, as datasets and

classifier implementations are not publicly shared. The UCR archive addresses this issue, and it is now possible to rigorously compare different time-series classifiers.

2.2.1 Motivation

Time-series classification is a problem arising across domains. There is a time-series classification problem associated with almost every sensor. A wide range of domains and tasks are represented by the UCR archive.

Creating fast and effective classifiers for these problems has a high impact. Identification of classes in some noteworthy domains within the UCR archive include:

- Medical
  - Electrooculography (EOG) [13]
  - Electrocardiography (ECG) [14]
- Environmental Science and Agriculture
  - Image-based leaf species [15]
  - Audio-based insect species [16]
  - Satellite image-based crop identification [17]
  - Spectrographs of food samples for quality control [18] [19]
- Industrial Diagnostics
  - Semiconductor wafer production [14]
  - Electricity usage [20]

These are just a select few of the many applications for which time-series classification can be used. For each domain and application, there are unique features which may be useful for aiding classification. The goal of classifying over the entire archive is to create methods which extract useful features across domains. In order to ensure applicability to new tasks, time-series classifiers should be domain agnostic. Additionally, it is always possible to incorporate domain specific knowledge down stream of a general purpose classifier.

Several criticisms have been levied against the UCR archive. Importantly, the datasets in the archive are relatively small, which may impact applicability for real-world problems. The largest dataset, StarLightCurves, only has 9236 time series. Contrast this with the popular image dataset ImageNet, which contains approximately 1.2 million images [21]. Even with its shortcomings, the UCR archive is the only collection of its kind, and continues to facilitate the rapid growth in time-series classification.

2.2.2 Baseline Classifiers

There are three baseline classifiers provided with the UCR archive. Euclidean Distance (ED), constrained Dynamic Time Warping (cDTW) (with a window size of 100), and learned cDTW are used as time-series distances for 1-nearest neighbor (1-NN) classifiers. These baseline methods are used in Section 3.1 to evaluate the effectiveness of existing topological approaches. These classifiers are fast, intuitive, and difficult to outperform using a naive strategy.

Assume we have two time series, x and y, of length n. Euclidean Distance and cDTW both provide a distance which can be used to assess how similar x is to y. Euclidean Distance between two time series is also the most constrained version of dynamic time warping. The Euclidean Distance between two time series is simply the absolute value of the pairwise difference of the aligned points. The time series must be the same length in order to use Euclidean Distance.

Definition 2.2 (Time-series Euclidean Distance) $ED(x, y) = \sqrt{\sum_i^n (x[i] - y[i])^2}$

Euclidean Distance is very fast to compute, but is the least accurate of the baseline classifiers. It is overly sensitive to small shifts in otherwise similar time series. This issue is addressed by DTW, which allows for a dynamic pairing of points, instead of a fixed pairing as used with Euclidean Distance. Instead of a one-to-one mapping, DTW utilizes a one-to-many mapping.

Let D be the distance matrix between x and y, such that $D_{i,j} = |x[i] - y[j]|$. Let W be a contiguous path from $D_{0,0}$ to $D_{n-1,n-1}$, where the k^{th} element of W is $W[k] = |x[i] - y[k]|$. The total cost of the optimal path $W_{\min}$ is the DTW distance.

Definition 2.3 (Time Series Dynamic Time Warping Distance) $DTW(P, Q) = \sum_i^n W_{\min}[i]$

This is the unconstrained version of DTW, as $W_{\min}$ can be any contiguous path through D. Unconstrained DTW places ahead of Euclidean Distance, ranking second place among the three baseline classifiers. The third and final baseline classifier is constrained DTW (cDTW).

Computing cDTW works identically to DTW, with the one difference being that the path W is constrained. Specifically, a window parameter ω specifies the degree of warping the path W is allowed to apply. In practice, this means that W cannot deviate from the diagonal of D by more than ω cells. Unconstrained DTW uses $\omega = \infty$. Figure 2.2 shows a comparison of Euclidean distance and cDTW.

The choice of ω significantly impacts classification accuracy. The baseline cDTW classifier chose ω on a per dataset basis, using cross-validation over the training set. This is the best performing baseline method.

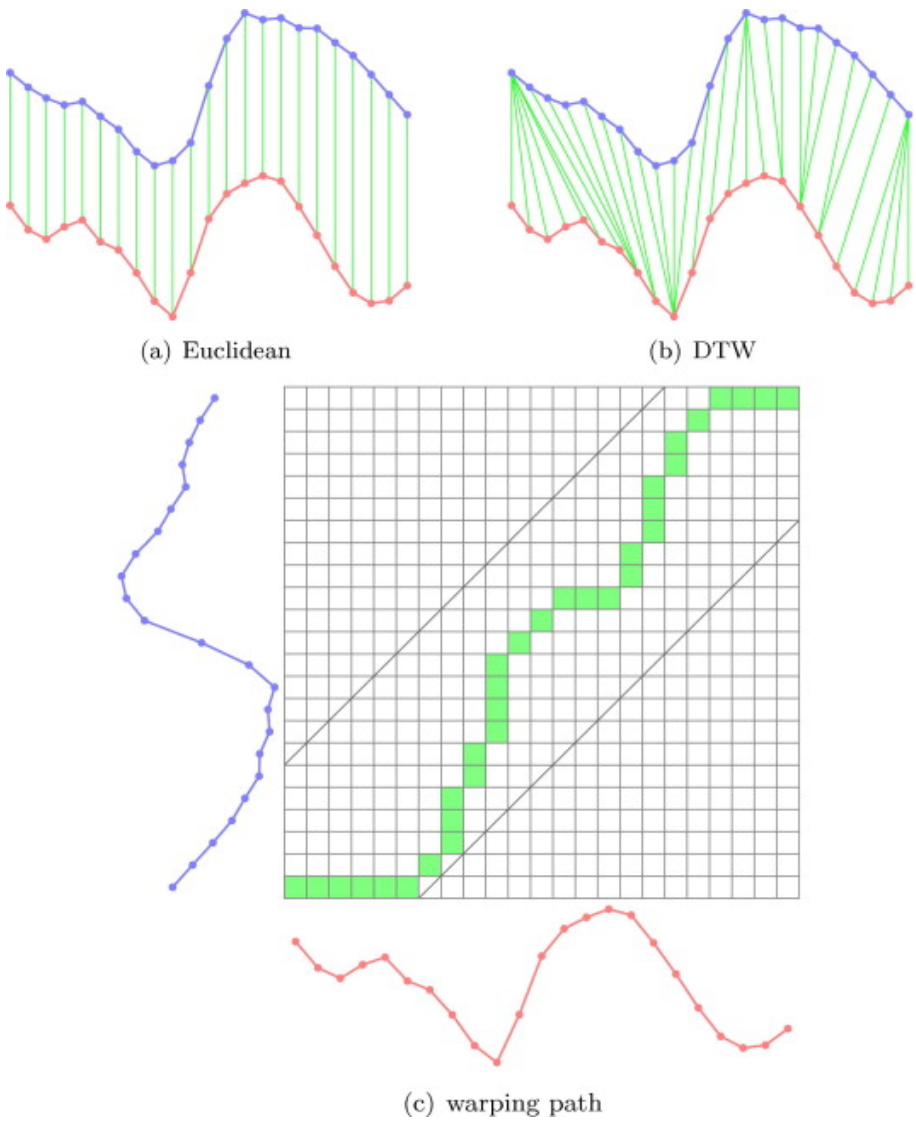

Figure 2.2: Note the warping between (a) and (b). The diagonal lines in (c) indicate a constraint on the path W [1].

2.2.3 State-of-the-art Classifiers

There are many types of time-series classifiers, with more being released on a consistent basis. With this in mind, we restrict our coverage of additional classi-

fiers to the top five (5) current state-of-the-art classifiers, and their close variants. Additionally, because TopRocket is a Rocket family classifier, we cover the Rocket methodology in detail. While interesting in their own right, HIVE-COTE 2.0, InceptionTime, and TS-CHIEF are presented only to facilitate comparison to Rocket family classifiers. This list is based on the ranking used in MultiRocket [22]. As we will show, TopRocket replaces MultiRocket as the top ranked scalable classifier, claiming second place in the ranking. Excluding TopRocket, the current ranking is:

1. HIVE-COTE 2.0
2. MultiRocket
3. InceptionTime
4. TS-CHIEF
5. MiniRocket

2.2.3.1 Rocket Classifiers

The Rocket (**R**and**o**m **C**onvolutional **Ke**rnel **T**ransform) family of classifiers consists of three sequentially published methods, each improving on its predecessor. The Rocket family is known for being very fast and accurate. These classifiers extract a large number of simple features from the time series. These features are then used for classifying the time series, usually using a linear regression model. Specifically, all of the Rocket classifiers presented here utilize a ridge regression

classifier, with the regularization parameter selected by cross-validation over the training set [23].

All Rocket methods first convolve the time series with a large set of kernels. This results in a large number of transformed time series. Very fast pooling operators are used to extract features from these transformed time series. These pooled values are concatenated into a (high dimensional) vector and used for classification. The classifiers primarily differ in how the convolution is carried out and which pooling operators are applied. TopRocket is an extension of this family, which uses a topological pooling operator in conjunction with existing conventional pooling operators.

Notably, all Rocket family classifiers are publicly available and open source. This keeps with the spirit of the UCR archive, and is instrumental in the development of TopRocket. Over the long term, the online locations of these classifiers may be subject to change. The interested reader is referred to the relevant, original Rocket papers for the latest location of the classifiers.

With all Rocket classifiers, it is possible to specify the desired dimension of the output vector. The number of kernels and dilations used are automatically changed to the feature vector length. This means that the methods can be easily scaled to accommodate different hardware constraints. We summarize the key points of the three primary Rocket family classifiers, as follows.

Rocket The original classifier in this family, Rocket was the best performing classifier when published [24]. There are two sources of randomness in the model. The

first is in the choice of kernels, the second is in the choice of bias values used to apply the pooling operations. The two pooling operations applied to each transformed time series are proportion positive value (PPV) and maximum value. Each transformed time series is the result of the following five defining features: kernel length, kernel weights, kernel dilation, kernel padding, and biases.

Kernel length is uniformly randomly selected from the set {7,9,11}. The lengths are all odd, because by default the kernels are centered. The short length of the kernels helps to ensure that Rocket is scalable for large time series. The random choice proves to have a limited impact on accuracy, so kernels are restricted to length 9 in later Rocket variants.

The weights of each kernel are randomly chosen from the standard normal distribution. The randomization of the weights ensures each kernel extracts distinct information from the time series. However, due to the random weights, it is difficult to optimize the convolution step. MiniRocket addresses this by restricting the possible weights.

Kernel dilation specifies how much the convolution is "spread out". Convolution as conventionally defined has an implicit dilation of 1. A dilation of 2 means that the weights of the kernel are convolved with every second element of the input time series. This dilation is applied locally, as the kernel slides past each point in the time series. Dilation allows for the same, or similar kernels, to be applied multiple times at different scales. Dilations are randomly sampled on an exponential scale.

Definition 2.4 (Rocket Dilation Sampling) $d = \lfloor 2^x \rfloor, x \sim U(0, max)$, *where* $max = log_2 \frac{l_{input}-1}{l_{kernel}-1}$.

Padding provides a similar increase in the variety of kernels. The choice of padding or no padding for a given kernel is uniformly randomly chosen. If padding is used, 0's are appended to the start and end of the base time series. Enough 0's are used relative to the kernel length such that each element of the base time series is centered for a convolution. If the kernel is length l, then a padding length of $\lfloor ((l-1) * d)/2 \rfloor$ is used, where d is the dilation. Without padding, the kernel is never centered over the first and last $\lfloor l/2 \rfloor$ points of the time series.

The biases are randomly selected from $U(-1, 1)$, but only positive biases are used. Each kernel is paired with a single or multiple biases, depending on the model parameters. The number of biases and dilations can be changed in order to obtain the desired length feature vector.

After the convolution is carried out, the biases are added to the transformed time series before the pooling operators are applied. These biases have the effect of extracting multiple useful features from the same transformed time series. For the result of each convolution, the maximum value and the PPV at each bias is computed and appended to the feature vector.

The resulting feature vector of pooled features is used to classify the time series. For Rocket, the parameters are set such that $20,000$ pooled features are extracted from each time series. While it is possible to use a wide variety of classification methods, ridge regression provides fast and accurate results. This classifier is used

for all Rocket family classifiers over the UCR dataset. Logistic regression is well suited to exceptionally large datasets, larger than any included in the UCR archive.

MiniRocket MiniRocket (**MINI**mally **R**and**o**m **C**onvolutional **Ke**rnel **T**ransform) is a faster and more accurate version of Rocket [25]. Each transform has the same five defining features, but they vary in the following ways, improving speed and accuracy. The randomness is removed from the construction of the kernels. Kernels are set to a fixed length of nine elements, with values either -1 or 2. Specifically, three elements in the kernel are chosen to be 2 and the remaining elements are set a to -1. There are 84 such unique kernels. Due to the kernel construction, it is possible to efficiently compute the convolution using scalar multiplication and matrix addition. This is the primary reason MiniRocket is faster than Rocket.

Instead of completely random biases, the biases are derived from the training set. Specifically, for each kernel and dilation combination, a training sample is randomly chosen and convolved. Biases for the kernel and dilation pair are then set as the quantiles of the output. Aside from the random selection of a training sample, this is an entirely deterministic process.

Similarly, the dilations are also selected deterministically, spaced evenly on an exponential scale. This allows for additional optimizations, contributing to the speed of MiniRocket.

Definition 2.5 (MiniRocket Dilation Sampling) $d = \{\lfloor 2^0 \rfloor, ..., \lfloor 2^i \rfloor, \lfloor 2^{max} \rfloor\}$, *where* $max = log_2 \frac{l_{input}-1}{l_{kernel}-1}$, *and the exponents* i *are spaced evenly from* 0 *to* max.

Due to the exponential nature of the construction, there are more smaller

dilations than larger dilations, so small dilations are responsible for most of the features. To prevent excessive computation for longer time series, the constraint $max \leq 32$ is applied. This constraint, as well as the length of the time series, places a limit on the number of features that can be added by increasing the number of dilations. If additional features are required, the difference is made up by computing additional bias values. Padding is also deterministic, with every other kernel and dilation combination using padding.

Under this construction, the use of two pooling operations does not improve accuracy. For this reason, only PPV is used as a pooling operator. Unlike Rocket, in which an arbitrary output dimension can be chosen, MiniRocket is restricted to an output dimension which is a multiple of 84, the number of kernels used. Using just under 10000 features ($9996 = 84 * 119$), MiniRocket outperforms Rocket and is significantly faster. MiniRocket ranked in third place when it was released.

MultiRocket MultiRocket is very similar to MiniRocket, but is optimized to increase accuracy at the expense of speed. Prior to TopRocket, MultiRocket was the most accurate scalable classifier, behind HIVE-COTE 2.0.

There are two primary differences which make MultiRocket more accurate than MiniRocket. Features are extracted from the first order difference of the time series, as well as the base time series. Additionally, three new pooling operators are used, bringing the total to four. The four pooling operators are: PPV, length of the longest positive stretch, mean positive index, and mean of positive values. These pooling operators are computed relative to the biases. Instead of $10,000$ features,

MultiRocket uses $50,000$ features.

Aside from these changes, MultiRocket proceeds identically to MiniRocket. MultiRocket is about an order of magnitude slower, but significantly more accurate than MiniRocket. TopRocket continues in this direction, with the addition of a topological pooling operator, significantly improving accuracy.

2.2.3.2 HIVE-COTE 2.0

The latest HIVE-COTE version, HIVE-COTE 2.0 (HC2) is the current reigning champion of the UCR archive, in terms of accuracy [26] [27]. HIVE-COTE is an acronym for **Hi**erarchical **V**ot**e** **Co**llective of **T**ransform-based **E**nsembles. The goal of this family of classifiers is to maximize accuracy over the archive, regardless of computational cost. HC2 provides improvement in both speed and accuracy over HIVE-COTE.

As meta-ensemble, HC2 incorporates four classifiers, two of which are novel, and two of which are derived from existing classifiers. Each classifier is used to create a constituent ensemble, each of which is trained separately from one another. The results of each ensemble are combined using a bagging approach, providing excellent accuracy. Significant effort in the design of HC2 is directed toward ensuring bounded training time. Specifically, a time contract mechanism places limits on how long each ensemble can be trained.

One of the notable improvements present in HC2 is Arsenal, a novel Rocket ensemble. This approach means it is possible to incorporate new Rocket family

classifiers into future iterations of HIVE-COTE. With this in mind, it is not unreasonable to expect that TopRocket might be included in the next iteration of HIVE-COTE.

2.2.3.3 InceptionTime

Neural networks have come to dominate many areas of machine learning. Notably, time-series classification, and specifically the UCR archive is not one of these domains. Even still, it is possible to obtain impressive results over the archive using deep learning methods, as demonstrated by InceptionTime [28].

One of the most popular neural network architectures, Inception networks provided state-of-the-art image classification when released [29]. InceptionTime modifies and extends the Inception architecture to work with time-series data. This runs counter to the widely-held intuition that recurrent neural networks, not convolutional neural networks, are better for sequential data. InceptionTime uses an ensemble of Inception-based networks.

The intersection of TDA and neural networks is a very active area of research. One such promising TDA tool is PersLay, a neural network layer designed to accept persistence diagrams, which are a foundational object of study in TDA [30]. Learned neural persistence representations are another potential option [31]. Given the success of TopRocket, it may be possible to incorporate tools from this area with InceptionTime, or similar neural classifiers.

2.2.3.4 TS-CHIEF

Another ensemble method and the current fourth ranked classifier, TS-CHIEF (**T**ime **S**eries **C**ombination of **H**eterogeneous and **I**ntegrated **E**mbedding **F**orest) is similar in many ways to HC2 [32]. As the name suggests, TS-CHIEF utilizes a decision-tree variant known as a Proximity Forest (PF) over a large number of time-series embeddings [33]. At a high level, this method utilizes a decision forest of k trees. Tree construction begins at the root node and proceeds recursively to the leaf nodes, just as with a decision tree.

As it moves from the root down to the leaves, the dataset is split at each node according to a splitting function. For each node, the splitting criteria is based on the distance of the sample to a fixed randomly selected training sample, as measured by several time-series distances. The leaf nodes are used for voting classification. The innovation of TS-CHIEF is to use a set of candidate splitting functions at each node, all of which are specifically designed for time series. The classifier uses many candidate splitting functions derived from existing classifiers, giving a notion of time-series "proximity". The best splitting function is selected at each node based on a split purity measure, called the Gini index, which is similar to information gain used in classical decision trees.

2.3 Topological Data Analysis

Topology is the branch of mathematics which focuses on the properties of an object which are preserved under continuous deformations. Sometimes called

"rubber sheet geometry," topology allows for manipulating an object through "nice" continuous operations, such as stretching, shrinking, and twisting. Operations such as gluing, tearing, and forming new holes are not allowed. The motivating question underlying topology is: What describes the properties which remain under these allowed operations? Intuitively, the two descriptive properties are how many holes there are in an object, and how it is connected. The properties are formalized to varying degrees using the ideas of homeomorphism, homotopy, homology, and many other related concepts.

The goal of TDA is to leverage the vast array of tools from topology to better understand data. There are several key areas of investigation with regard to TDA, which can be summarized as follows:

1. How can data be transformed into a topological space?
2. How can useful information be extracted from the resulting topological space?
3. What are the properties of the extracted information?
4. How can the information be used for downstream analysis?

Each of these questions will be answered in turn. Additionally, we show how the answers to the above questions can be applied to time-series data. This section concludes with a discussion of persistence curves, a recently developed framework with a strong connection to TopRocket.

TDA is a rich and rapidly developing field with a wide array of techniques. We restrict our discussion to a particularly useful topological property: persistent

homology. Further, we are only concerned with the persistent homology of a computationally convenient type of topological space called a simplicial complex. Even with this seemingly narrow view, it is possible to explore a wide area of interesting applications in TDA.

It should also be noted that TDA is deeply rooted in algebraic topology. This connection will likely bear additional fruit in the future, especially in the domain of time-series classification. However, a full description of the foundations of TDA is not required to understand the methods presented here. As much as possible, we eschew the fundamental, yet abstract, mathematics underlying TDA. Instead, we favor intuitive explanations, which explain how the tools of TDA work and why those tools are useful for time-series classification. Several of the following basic definitions are taken, at least in part, from the excellent introductory text by Eldelsbrunner and Harrer [34].

2.3.1 Motivation

The seminal work in TDA is primarily concerned with the properties of point clouds [35]. There are an ever-increasing number of data streams, and often times the data is very high dimensional. Additionally, data is often naturally represented as a point cloud in high dimensional space. The initial working assumption is that the topological properties of point clouds, which occur across scales, provide useful information about process underlying the point cloud.

The core idea is to construct topological spaces at different scales, and analyze

how the topological properties of the spaces change as the scale varies. Homology is an especially useful topological property. While not perfect, in a rigorous sense as we will show, homology is relatively fast to compute and yields a good representation of the "holes" in a space. Tracking how homology changes across scales directly leads to the idea of persistent homology.

2.3.2 Topological Spaces From Data - Simplicial Complexes

The simplicial complex is a central idea in applied topology. This is the primary object studied using persistent homology. As a baseline intuition, it is useful to think of a simplicial complex as a higher dimension analog of a graph. In addition to vertices and edges, a simplicial complex can have 2-dimensional faces, 3-dimensional filled spaces, and n-dimensional hyper-faces. We proceed by defining simplices, simplicial complexes, and abstract simplicial complexes (ASCs).

Let $U = u_0, ..., u_1, ..., u_k$ be a set of points in $\mathbb{R}^d$. A convex combination over U is $x = \sum \lambda_i u_i$ such that $\sum \lambda_i = 1$ and $\lambda_i \geq 0 \forall i$. The convex hull of U is the set of all such combinations.

Definition 2.6 (k-Simplex) *The convex hull of $k+1$ affinely independent points is a k-simplex.*

A 0-simplex, 1-simplex, 2-simplex, and 3-simplex correspond to a vertex, line segment, filled triangle, and tetrahedron, respectively. A face of simplex is the convex hull of a non-empty subset of U, and is also a simplex. A proper face is the convex hull of a proper non-empty subset of U. If τ is a face of σ, we write $\tau \leq \sigma$,

or $\tau < \sigma$ if τ is a proper face. The boundary of a simplex is the union of its proper faces. Each simplex with $k \geq 1$ is bounded by $k-1$ simplices. The set of vertices of a k simplex has $2^{k+1}-1$ faces. A simplicial complex is a collection of simplices which obey the rules of a topological space, as seen in Figure 2.3.

Definition 2.7 (Simplicial Complex) *Let K be a collection of simplices. K is simplicial complex if:*

1. *For simplices $\sigma \in K$ and τ, if $\tau \leq \sigma$, then $\tau \in K$.*

2. *If simplex $\sigma, \sigma_0 \in K$, then $\sigma \cap \sigma_0$ is either empty, or a face of both σ and σ_0.*

A simplicial complex is inherently tied to an underlying space, in this case R^n. For computational purposes, it is better to describe a simplicial complex in purely combinatorial terms. This is the role of the abstract simplicial complex.

Definition 2.8 (Abstract Simplicial Complex) *A finite collection of sets, A, is an abstract simplicial complex if $\alpha \in A$ and $\beta \subseteq \alpha$ then $\beta \in A$.*

There are several methods for constructing an ASC from point-cloud data. We focus on the Vietoris-Rips (VR) complex, because it is efficient to compute. The Vietoris-Rips is based on the idea of placing a ball of radius r around each vertex in a set S. Two vertices are connected by an edge if the radius r ball of each intersect each other. The cliques of the graph are included in the abstract simplicial complex as higher dimensional simplices.

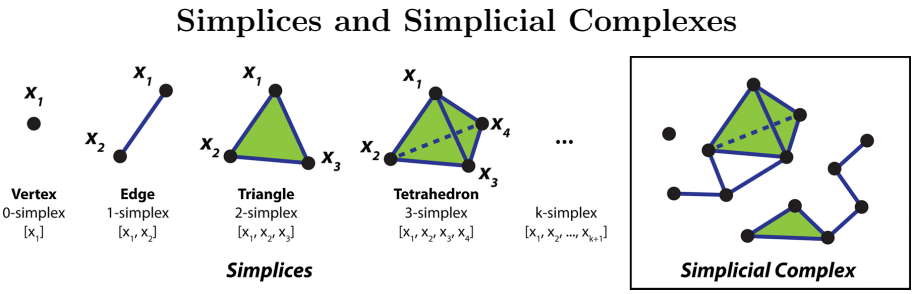

Figure 2.3: An example of low-dimensional simplices and how they can combine to form a simplicial complex [2]

Definition 2.9 (Vietoris-Rips Complex) $\mathrm{VR}_r(S) = \{\sigma \subseteq S | d(u,v) \leq r, \forall u \neq v \in \sigma\}$

The Vietoris-Rips complex allows us to construct a topological space from a point cloud of data, but the choice of the radius parameter is often not obvious. In fact, for many applications, there may be no clear choice of radius. Instead, we are interested in how the ASC changes as r changes. This is the purpose of persistent homology.

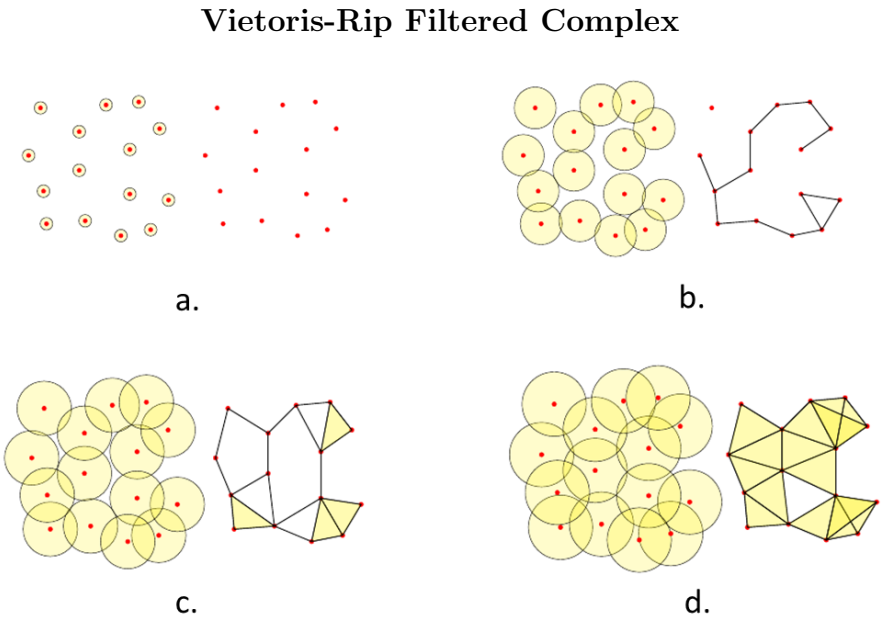

Figure 2.4: A sequence of Vietoris-Rip complexes formed from an underlying 2-dimensional point cloud [3]

2.3.3 Extracting Useful Information - Persistent Homology

Intuitively, persistent homology tracks how "holes" form and close in a topological space across scales. It is one of the primary tools of TDA. We make this definition more rigorous by first defining homology, and then persistence. We are only concerned with the persistent homology of ASCs, although the idea applies more generally.

2.3.3.1 Homology

The homology of an ASC, S, is defined in terms of boundaries and cycles within S. A homological hole is a cycle in S which is not the boundary of some higher

dimensional simplex in S. S can contain holes of different dimensions, depending on the dimension of the cycles and boundaries involved.

We begin by formally defining cycles and boundaries of an ASC. In order to detect cycles and boundaries, we need a way of combining simplices within S. This is accomplished by constructing chains over S.

Definition 2.10 (p-Chain) *The formal sum of the p-simplices in S, $c = \sum \alpha_i \sigma_i$, with coefficients $\alpha_i \in \mathbb{F}_2$, is a p-chain of S.*

The coefficients indicate whether a given simplex is included in the chain, with $\alpha_i = 1$ indicating membership. In general, coefficients can be taken from any field, but for this work, and computational topology in general, $\mathbb{Z}^2$ is used to simplify computation. Component-wise addition is used to add two p-chains. If $c = \sum \alpha_i \sigma$ and $c_1 = \sum \beta_i \sigma_i$, then $c + c_1 = \sum(\alpha + \beta_i)\sigma_i$. Together with this addition operator, the p-chains of S form an abelian group $C_p(S)$ or just C_p when the ASC is clear from context. The boundary of a p-simplex, σ, is the chain of its $(p-1)$-dimensional faces.

Definition 2.11 (Simplex Boundary) $\partial_p \sigma = \sum_{i=0}^{p} [u_0, ..., \hat{u}_j, ..., u_p]$ *is boundary of the p-simplex σ.*

The simplex $[u_0, ..., \hat{u}_j, ..., u_p]$ is the $(p-1)$-simplex formed from σ by removing vertex u_j. The boundary operator ∂_p is a homomorphism from C_p to C_{p-1}. The sequence of chain groups of S connected by the boundary homomorphism is the chain complex of S.

It is now possible to define cycles and boundaries in terms of chains. A p-cycle is a p-chain with a boundary of 0. The p-cycles form an abelian subgroup $Z_p \leq C_p$. A p-chain is a p-boundary if it is the boundary of a chain in C_{p+1}. All p-boundaries are p-cycles. The p-boundaries also form an abelian subgroup $B_p \leq Z_p \leq C_p$. The boundary group B_p is the image of boundary operator B_{p+1}.

Definition 2.12 (The Fundamental Lemma of Homology) $\partial_p \partial_{p+1} d = 0$ *for every integer p and every $(p+1)$-chain d.*

As B_p is a subgroup of Z_p, every element is a cycle, meaning the boundary of every element is 0. Now that we have a formal definition of cycles and boundaries, we recall the intuitive definition of "holes" as cycles which are not the boundaries of some higher dimensional simplex. This notion is made precise by the definition of homology.

Definition 2.13 (Simplicial Homology) *The p^{th} homology group is the p^{th} cycle group modulo the p^{th} boundary group, $H_p = Z_p/B_p$, and is abelian.*

The p^{th} Betti number is the rank of H_p, and corresponds to the number of p-dimensional "holes." We now have a way to count the number of holes in an ASC. The 1^{st} Betti number corresponds to holes bounded by edges, the 2^{nd} to voids bounded by triangles, and so on. Somewhat counter-intuitively, the 0^{th} Betti number is the number of connected components in the ASC. The number of connected components of an ASC is particularly relevant for time-series applications, as will be shown.

Computing the p^{th} Betti number of an ASC is accomplished via a simple matrix reduction algorithm. The p-boundary matrix is a binary matrix which completely describes the boundary operator ∂_p over ASC. The matrix D_p consists of rows corresponding to $(p-1)$-chains and columns corresponding to (p)-chains, where $D[i,j] = 1$ indicates the i^{th} $(p-1)$-chain is in the boundary of the j^{th} p-chain. This reduction algorithm transforms the boundary into a form from which the Betti numbers can be read. This process is extended in the following section to capture how the Betti numbers change across scales.

2.3.3.2 Persistence

As shown in Section 2.3.2, constructing an ASC from data often involves a parameterization. In the case of the VR complex, the important parameter is the radius r. This naturally leads to the question: How does the homology of a VR complex change as r changes? Persistent homology seeks to answer this question.

Instead of working with a single ASC, we now turn our attention to a sequence of complexes, called a filtered abstract simplicial complex. The corresponding sequence of homology groups is a filtration, and the object which persistent homology describes. Let f be a monotonic real-valued function over the simplices of an ASC, $f : S \to \mathbb{R}$. In this case f is monotonic with respect to the order of the faces in S, meaning if $\sigma \leq \tau$ then $f(\sigma) \leq f(\tau)$. Let $S_a = f^{-1}(-\infty, a]$, the sub-complex of all $\sigma \in S$ such that $f(\sigma) \leq a$.

Definition 2.14 (Filtered Abstract Simplicial Complex) *The sequence of in-*

creasing sub-complexes of S, $\emptyset = S_0 \subseteq S_{a_1} \subseteq ... \subseteq S_{a_n} = S$, *where* $0 \leq a_1 \leq ... \leq a_n$, *is a filtered ASC.*

For every $a_i \leq a_j$, there is an inclusion map from S_{a_i} to S_{a_j}. These inclusion maps induce a homomorphism over the filtered homology groups, $f_p^{a_i,a_j} : H_p(S_{a_i}) \rightarrow H_p(S_{a_j}) \forall p$. The filtration of S with respect to f is the resulting sequence of homology groups: $0 = H_p(S_0) \rightarrow H_p(S_{a_1}) \rightarrow ... \rightarrow H_p(S_{a_n}) = H_p(S)$, where $0 \leq a_1 \leq ... \leq a_n$.

Definition 2.15 (Persistent Homology) *The* p^{th} *persistent homology groups are the images of the induced homomorphisms,* $H_p^{a_i,a_j} = \text{im } f_p^{a_i,a_j}$, $0 \leq a_i \leq a_j \leq a_n$.

Betti numbers are similarly extended to persistent Betti numbers, $B_p^{a_i,a_j} = \text{rank} H_p^{a_i,a_j}$. The equivalence classes of $H_p^{a_i,a_j}$ represent the p-dimensional holes of S_{a_i} which still exist in S_{a_j}. Now, it is possible to track when holes are born and die over the course of the filtration. The persistent Betti numbers can be used to provide precisely this information. Let $\mu_p^{a_i,a_j} = (B_p^{a_i,a_{(j-1)}} - B_p^{a_{(i-1)},a_{(j-1)}}) - (B_p^{a_{(i-1)},a_{(j-1)}} - B_p^{a_{(i-1)},a_j})$, where $0 \leq a_i \leq a_j \leq a_n$. Each μ^{a_i,a_j} is the number of holes which are born in complex S_{a_i} and died entering S_{a_j}.

Definition 2.16 (Persistence Diagram) *The multiset of all points* (a_i, a_j) *with multiplicity* μ^{a_i,a_j}, *for fixed dimension* p, *is the* p^{th} *persistence diagram of the filtration, denoted* $\text{Dgm}_p(f)$.

By the Fundamental Lemma of Persistent Homology, the $\text{Dgm}_p(f)$ contains all information about the p^{th} persistent homology groups [34]. While persistent homology may appear significantly more complicated than homology, it is nearly

just as straightforward to compute. There are two major differences in the construction of the persistent boundary matrix. First, the matrix includes all simplices of S represented by both the rows and columns. Second, the simplices are ordered (monotonically increasing) by both simplex order (coface relationship) and filtration order. Applying a simple reduction algorithm and subsequent read off procedure to this matrix yields the persistence diagram. The algorithm is detailed in Appendix A.1.

A filtration of a VR complex can be created by using the radius r in the filtering function, as shown in Figure 2.4. Specifically, for a VR filtration, $f(\sigma) = r_{min}$, where r_{min} is the smallest radius such that $\sigma \in \mathrm{VR_{rmin}(S)}$. While not always fast, the VR filtration tends to at least be tractable over reasonably large point clouds. Significant effort is dedicated to speeding up the computation of persistent homology over VR filtrations [36].

Persistence diagrams are the primary focus of much of TDA, especially at the intersection with machine learning. Given that the diagram is a multiset of points, it is not immediately clear how it can be used with downstream machine learning models. In order to do this, we first explore the properties of persistence diagrams and methods of comparison.

2.3.4 Properties of Persistence - Distances and Stability

With the heavy lifting of computing the persistent homology out of the way, we must now make sense of the resulting information. We first review the two basic

distances for comparing persistence diagrams, and then present the idea of stability, which guarantees the distances are representative of the underlying data. This section serves to motivate other methods for extracting and comparing persistence information, as the basic distances are very costly to compute.

2.3.4.1 Wasserstein and Bottleneck Distances

The Bottleneck and Wasserstein distances between persistence diagrams are closely related. Both involve taking the l_∞ norm between pairings of points of two diagrams. The Bottleneck distance makes use of this norm without any additional steps. The Wasserstein is a generalization of the Bottleneck distance.

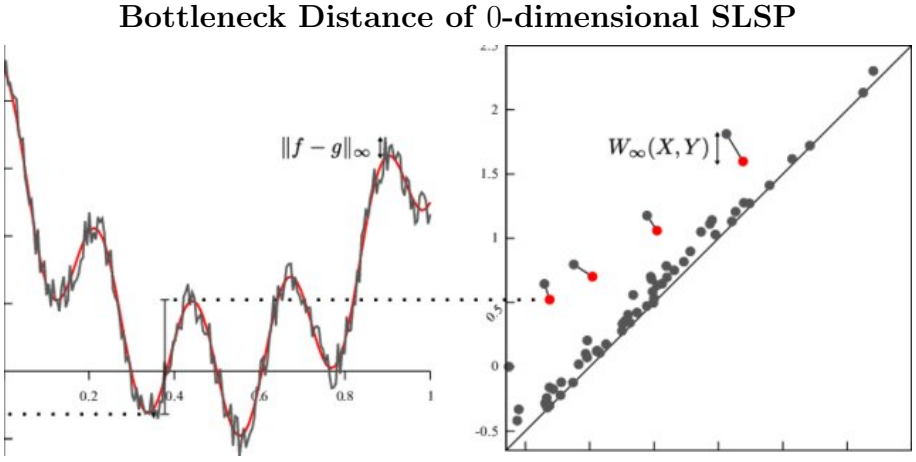

Figure 2.5: Two curves and persistence diagrams are shown, colored red and grey. The red curve contains fewer peaks and valleys, and correspondingly fewer persistence points than the grey curve. All red points are paired, while most of the gray points are noise induced and paired with the diagonal. This provides an intuition for stability. [4]

Assume X and Y are two persistence diagrams. Let these diagrams have points $X = (x_{1_1}, x_{1_2}), ..., (x_{m_1}, x_{m_2})$ and $Y = (y_{1_1}, y_{1_2}), ..., (y_{n_1}, y_{n_2})$. The first coordinate of each point corresponds to when a persistent homology class (a hole) was born, and the second coordinate refers to when it died. It is convenient to use $x_i = (x_{i_1}, x_{i_2})$. Computing the distance between the diagrams is accomplished by finding a minimum unique pairing, or more formally a bijection, from the points of X to the points of Y. The Manhattan distance between the pair of most separated points, within the overall minimal total pairing, is the Bottleneck distance. Let $\eta : X \to Y$ be a bijection between X and Y, where $\eta(x) \in Y$ is the point paired with $x \in X$. This pairing process for the 0-dimensional persistence of a time series is shown in Figure 2.5.

Definition 2.17 (Bottleneck Distance) *Given persistence diagrams X and Y, $W_\infty(X, Y) = \inf_{\eta: X \to Y} \sup_{x \in X} \|x - \eta(x)\|_\infty$ is the Bottleneck distance between X and Y.*

As noted above, the number of points in X and Y might be different. As a computational trick, it is assumed that there are an infinite number of points along the diagonal. Any unpaired points for a given bijection are paired to the diagonal to compute the overall distance. It is possible to compute the Bottleneck distance without these points, but in practice it simplifies the computation, so the extra points are always included. The Bottleneck distance is somewhat constrained and insensitive because the final distance is ultimately just the distance between a single selected pair of points. The Wasserstein distance addresses this by incorporating the distance between all pairs in the minimal bijection. As q goes to infinity, the

q-Wasserstein distance approaches the Bottleneck distance.

Definition 2.18 (Wasserstein Distance) $W_q(X,Y) = \left[\inf_{\eta:X\to Y} \sum_{x\in X} \|x-\eta(x)\|_\infty^q\right]^{1/q}$, *is the Wasserstein distance between persistence diagrams* X *and* Y*.*

Given two similar filtering functions f and g over an ASC, S, it is reasonable to assume the $\text{Dgm}_p(f)$ would be similar to $\text{Dgm}_p(g)$. This desirable property is called stability, and is formalized using the Bottleneck and Wasserstein distances.

Definition 2.19 (Bottleneck Stability) *Let* S *be a simplicial complex with two monotonic filtering functions* f *and* g*. For each* p*, the* l_∞ *distance between* f *and* g *bounds the distance between* $X = Dgm_p(f)$ *and* $Y = Dgm_p(g)$*.* $W_\infty(X,Y) \leq \|f-g\|_\infty$*.*

Stability is an important property in TDA, because for constructions such as the VR filtration, small changes in the underlying point cloud are equivalent to small changes in the filtering function. In effect, stability establishes a useful connection between data and the topological features extracted from it. The q-Wasserstein distance is similarly stable, given a few mild constraints on the filtering functions and the values of q.

The stability of these distances suggests uses in downstream machine learning, such as a k-NN classifier, or similar. For larger diagrams, however, these distances are not very useful in practice. This is due to a costly problem hidden in the definition of both distances. Namely, in order to find the optimal bijection, the bipartite graph matching between the points in X and Y must be brute forced. Even

with optimizations, this is too slow for many machine learning applications. This issue motivates the development of vectorizations and representations of persistence diagrams, which allow computationally more efficient comparison.

2.3.5 Utilizing the Information - Vectorizations and Representations

The following methods are used in the initial investigation of TDA for time-series classification. Each vectorization is utilized in conjunction with a 1-NN classifier in an attempt to outperform DTW. As will be shown in Section 3.1, none of these representations succeeded in beating DTW. These representations are chosen as a selection of the most commonly available representations available to TDA practitioners. Specifically, each representation described is implemented in the widely used Python TDA library Giotto [37].

When designing a new vectorization, there are two primary considerations. First, are the computational concerns, such as ensuring it is tractable to compute and compare the representation. In order to accommodate different computing trade-offs, all of the following methods include parameters to scale the size of the representation. The second consideration is stability. With the exception of Betti curves, each representation is stable with respect to either the Bottleneck or Wasserstein distance or, in some cases, both. This means that the distance between the representations of the persistence diagrams is bounded by either the Bottleneck or Wasserstein distances. As the Bottleneck and Wasserstein distances are also stable, this serves to reinforce the connection between the underlying data and the

persistence representation.

For the following examples, we refer to the points in an example persistence diagram D as (b, d) or (b_i, d_i). This is to emphasize that the persistence points refer to the **b**irth and **d**eath of holes in the filtration, which provides intuition for the different representations. In the following representations, t refers to a "time" in the filtration, for instance a specific radius in a VR filtration. This matches the intuition about the birth and death of holes. Each representation can be compared using the Minkowski distance over its discretized elements.

2.3.5.1 Betti Curves

The Betti curve over a persistence diagram is the most basic form of a persistence curve [38]. Betti curves, and by extension persistence curves, form the foundation of TopRocket, and are covered in more detail in Section 2.3.7.

Definition 2.20 (Betti Curve) $\beta(t) = \sum_{(b,d)\in D_t} 1, b < t < d.$

In practice, values of t are discretized into n bins from the minimum birth to the maximum death in the diagram. The distance between discretized curves $B(t)_n \in \mathbb{R}^n$ is taken using the Minkowski distance. TopRocket takes a different approach which, to our knowledge, is novel. TopRocket implicitly calculates $\beta(t)$ for a discrete values of t, without having to first calculate the persistence diagram.

2.3.5.2 Persistence Images

This technique results in a 2-dimensional discrete vectorization of the input persistence diagram [39]. The diagram is first converted to birth-life coordinates, $(b, d) \mapsto (b, d-b)$, so that all points lie in the first quadrant. This transformation of a persistence diagram contains the same information, but is better suited to downstream machine learning tasks. In birth-death coordinates, all of the points are above the diagonal (a hole must be born before it dies). Converting a birth-death plot to a square image means that half of the pixels will always be blank. Birth-life coordinates fix this problem by spreading the persistence points across the entire quadrant.

A differentiable probability distribution is centered at each point in the diagram, creating a persistence surface, which is then discretized to create a persistence image. It is possible to weight the persistence surface before discretizing under some mild conditions, but only the trivial weighting function is used in this work. Given a few mild constraints on the distribution used, persistence images are stable with respect to both the Bottleneck and Wasserstein distance. This means that when the same filtration method is applied to similar data, the resulting persistence images will be similar, as shown in Figure 2.6.

For the experiments, as is most often done in practice, a 2-dimensional normalized symmetric Gaussian with standard deviation σ is used and the surface is discretized into a square grid. The *bins* parameter specifies the side length of the square image. For $bins = n$, the image will have n^2 pixels.

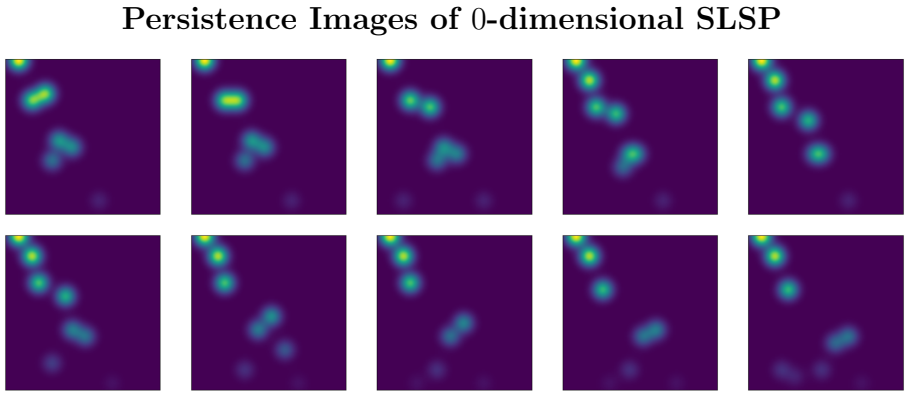

Figure 2.6: A collection of persistence images from a set of closely related time series [5].

2.3.5.3 Heat Kernels

In many ways the heat kernel is similar to a persistence image. This representation creates a vectorization in the same way using a square grid, but differs in underlying surface. The intuitive idea behind a heat kernel is to treat the persistence points as the initial condition for a heat diffusion problem. The underlying surface used for the kernel is then the solution to the heat equation at time $t > 0$ [40].

In practice, the heat kernel is computed much the same way as a persistence image, but the points are left in birth-death coordinates. The difference of the surface of the persistence diagram, and the surface of the persistence diagram reflected along the diagonal yields a surface which is discretized to give the heat kernel vectorization. Both surfaces are created using a 2-dimensional symmetric Gaussian with standard deviation $\sigma = \sqrt{2t}$.

2.3.5.4 Persistence Landscapes

A persistence landscape is a set of functions $\{\lambda_k\}, k \in \mathbb{N}$, where $\lambda_k : \mathbb{R} \to \bar{\mathbb{R}}$, where $\lambda_k(t)$ is the k^{th} largest value of $\{\Lambda_i(t)\}$.

$$\Lambda_i(t) = max\{0, \min\{t - b_i, d_i - t\}\} \tag{2.1}$$

Each λ_k is considered a layer in the landscape. The layers are discretized to yield a vector representation [41].

2.3.5.5 Silhouettes

This representation is similar to a persistence landscape. A weighted silhouette is effectively the weighted first layer of the corresponding persistence landscape [42]. Let the weighting function be $w_i = |d_i - b_i|^\rho$, where $\rho \in (0, \infty]$ is distinct from the distance parameter p shown in Table 3.1. The ρ-power weighted silhouette is:

$$\phi(t) = \frac{\sum_{i \in I} w_i \Lambda_i(t)}{\sum_{i \in I} w_i} \tag{2.2}$$

2.3.6 Persistent Homology of Time Series

We have shown how it is possible to construct a filtration from a point cloud using a sequence of VR complexes. In this section, we show how persistent homology can be applied to time-series data. There are several methods for accomplishing this, two of which are presented here.

First, applying a sliding window embedding to the time series transforms it into a point cloud. Persistent homology (of any dimension) can then be applied using a VR filtration, or similar. The second is to directly construct a simplicial complex from the time series, and then use a sublevel set filtration. This approach results in the sublevel set persistence (SLSP) of the time series. As will be shown, SLSP bears a striking resemblance to the methods used in Rocket family classifiers.

Let x be a time series of length n, $x := [x_1, ..., x_n]$. A point cloud can be created from x using Takens' embedding [43]. Takens' embedding was initially developed for analysing dynamical systems. It requires three parameters: a lag τ , a dimension d, and a stride s.

Definition 2.21 (Takens' Embedding) *The set of points in* $\mathbb{R}^d$, $\chi(x, \tau, d, s) = [x_i, x_{i+\tau}, \ldots, x_{i+(d-1)\tau}]$, *with* $i \in (0, 2s, 3s, \ldots)$ *as the stride, is Takens' embedding of* x, *also called the sliding window embedding.*

The resulting point cloud is then processed using any common filtration method, most often a VR filtration. The p-dimensional persistent homology of the resulting filtration can be calculated, for any dimension p.

Takens' embedding is a fundamental technique in TDA [44]. There are several drawbacks, however, namely computation cost (due to the VR filtration) and choice of parameters. There are well established methods for choosing the embedding parameters based on properties of the data. For a fixed stride, the stride is optimized through a process which selects a value that minimizes the time-delayed mutual information within samples [45]. Fixing the stride and the lag, a similar

nearest-neighbor test is used to select embedding dimension [46]. Optimizing these parameters and computing the VR filtration introduces significant computational cost. Furthermore, most often in practice, only the 0 or 1-dimensional persistent homology is utilized, as including higher dimensions classification tasks tends to have diminishing returns.

SLSP obviates the need for a point cloud embedding by directly computing the persistence of the time series. Specifically, a time series can be interpreted as a piecewise linear function, with lines between sample points. The lines between samples become 1-simplices (edges) and the samples points as 0-simplices (vertices), yields a simplicial complex. The ASC corresponding to this graph is very similar to a path graph.

Definition 2.22 *Let $T(x)$ be the ASC constructed from x, a time series of length n. $T(x) = \{\{v_1\}, ...\{v_n\}, \{v_1, v_2\}, ..., \{v_{n-1}, v_n\}\}$.*

A sequence of complexes can be created by adding in simplices from least to greatest, in order of the time series. In this case, the edges are added to the complex once both vertices are present. Recall that 0-dimensional persistent homology correspond to the number of connected components in an ASC. Starting with the empty ASC $T(X)_0$, the number of connected components will change as more simplices are added. Formally, the filtration is taken with respect to a sublevel set filtering function.

Definition 2.23 (Sublevel Set Filtration of Time Series) *$f(T(X)) = max(x(\tau))\forall i \leq \sigma \forall \sigma \in T(X)$, where τ are vertices of the ASC, and $x(\tau)$ is the value of the corre-*

sponding point of the time series.

This resulting 0-dimensional persistent homology of the time series is the SLSP. Intuitively, this process corresponds to scanning a horizontal line from the bottom of the time series to the top, and counting the number of connected components below the line. Due to the nature of the Rocket pooling operators, TopRocket actually computes the partial SLSP of the negative of the time series, which is the same as filtering from the top down instead. Computing the SLSP of a time series is more straightforward to optimize than the general persistent homology.

An algorithmically fast queue-based method is available through the Teaspoon library [47] [48]. In practice, the implementation of this algorithm leaves room for improvement. For all of the experiments presented in Section 3.1, the standard persistence algorithm provided by the Dionysus 2 library is used [49]. Additionally, in Appendix A.2 we include an exceptionally fast implementation of the standard persistence algorithm which is heavily optimized for time series. The algorithm makes use of the just-in-time Python compiler Numba, as well as Python hash maps, to exploit the constrained nature of SLSP [50]. To our knowledge, this is the fastest method for computing the 0-dimensional SLSP of a time series.

2.3.7 Persistence Curves

The primary novel contribution of TopRocket is the computing of the Betti curve of a time-series SLSP, without computing the full SLSP. This ensures that the classifiers remains fast, and provides a natural way to incorporate persistence

information. The Betti curve is a special case of a persistence curve [38]. Existing work on persistence curves requires a complete persistence diagram, and bins the curve into a histogram. Our method is different in that we do not need to compute the persistence diagram, and we directly use the value of the Betti curve at a certain fixed set of filtration values.

The only other TDA-based classifier run against the UCR archive utilizes a hybrid DTW/persistence curve method [51]. Using this approach, the authors are able to outperform learned DTW on 64 out of 128 datasets. The full results and the critical difference diagrams of these experiments are not published, so it is difficult to make a fair assessment of the technique. It should be noted that DTW is significantly less accurate than even the most basic Rocket classifier.

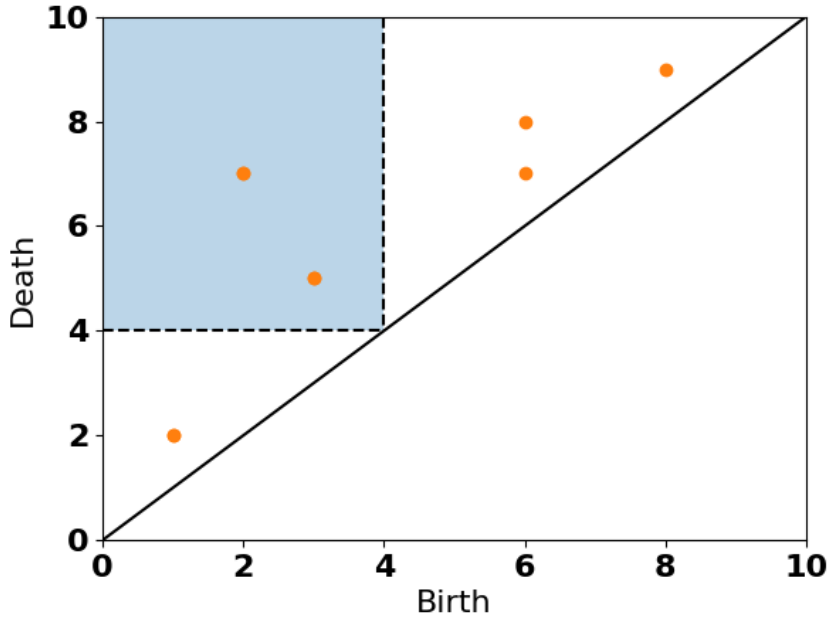

Figure 2.7: An example persistence diagram with sub-diagram D_4 shaded in blue. There two persistence points within the sub-diagram, so $\beta(4) = 2$.

The persistence curve is based on "cuts" of different filtering function values for t. Specifically, a sub-diagram D_t can be cut from a diagram D by taking all points which are already born and have not died before t, as shown in Figure 2.7. In other words, D_t contains all of the points which are "alive" at time t.

Definition 2.24 (Persistence Diagram Cut) *The cut of persistence diagram D at time t is the multiset $D_t = \{(b, d) \in D | b \leq t < d\}$.*

In the persistence curve framework, the Betti curve can be defined as the magnitude of D_t as t varies. Critically, computing values of the Betti curve does not necessarily require the birth/death pairs provided by a persistence diagram. This

fact is implicitly stated in the literature, but is not utilized. Instead, the past focus has always been on computing the Betti curve over an entire persistence diagram, or in some cases a truncated range.

Definition 2.25 (Betti Curve in Persistence Curve Framework) $\beta(t) = \#D_t = \sum_{(b,d)\in D_t} 1.$

There are two operations in the definition of the Betti curve which can be generalized. The first is the sum over the persistence pairs, which is denoted T. The second is an implicit function, denoted ψ, which maps every persistence pair in the sum to 1. Specifically, for the Betti curve $\psi(b, d) = 1$. Generalizing these operations yields the persistence curve. The choice of T and ψ specifies the type of persistence curve.

Definition 2.26 (Persistence Curve) $P(D, \psi, T) = T([\psi(D; b, d, t)|(b, d) \in D_t]).$

While not explored in this work, it is likely that other persistence curve features can be used in conjunction with Rocket family, or other, classifiers. As presented in the original work, there are currently 11 different types of persistence curves. Of these, the Betti curve, life entropy curve, persistence diagram threshold curve, and persistence landscape already existed, but are subsumed by the framework [52] [53] [41]. The remaining seven curves are introduced along with the framework. Some of the curves, such as persistence landscapes, are stable. Others are conditionally stable, have as of yet unknown stability, or are not stable. Betti curves are not stable. However, Betti curves are the only curves which can be computed without knowing the pairing of the birth and death times at time t.

This computational trick is the property currently exploited by TopRocket. Given our exceptionally fast algorithm implementation for computing SLSP of a time series, it may be possible to sacrifice a small amount of speed in favor of potential accuracy gains from using other curves. There are currently 11 curves, but there is no reason in the context of time-series classification to limit the search to just those curves. This presents an exceptionally large solution space, which we will endeavor to explore in future work, and encourage others to do the same.

Chapter 3: TDA for Time-series Classification

Having provided the proper foundation, we now explore the application of TDA for time-series classification. We begin this section by reviewing TDA based 1-NN classifiers designed to outperform DTW. The goal of these experiments is to assess the current state of TDA-based time-series classification. As will be shown, none of these attempted methods outperform learned cDTW.

Having assessed the current available options, we shift our attention to improving existing classifiers using TDA. MultiRocket is a clear choice for two reasons. First, the way bias values are used in conjunction with the pooling operators is evocative of sublevel set filtration. Second, MultiRocket is (or rather was, before TopRocket) the state-of-the-art scalable classifier over the UCR archive, so improving it empirically proves TDA can provide an edge for difficult machine learning problems.

3.1 Motivation for New Techniques

We begin the investigation of TDA for time-series classification by designing experiments around the persistence representations presented in Section 2.3.5. Specifically, we aim to create a TDA-based 1-NN classifier which outperforms learned

cDTW over the full UCR archive. For pre-processing, all time series are first z-normalized. In the very limited instances where a time series is constant, z-normalization is invalid, so a zero-fill time series of the same length is used instead. Importantly, z-normalization is idempotent, so this step has no effect on the datasets which are already normalized. For these experiments, all computations are carried out with Giotto, with the exception of SLSP, which is done with Dionysus 2 (our SLSP implementation would have been a faster choice, but it did not exist at the time these experiments were carried out).

Before choosing from the previously listed five representations, a filtration must first be chosen. Specifically, we must choose between a VR filtration over Takens' embedding or SLSP. In order to do this, we computed the persistence diagrams of the entire archive using both methods.

Attempting to compute Takens' embedding over the entire archive consumes significant resources and time. As a result, given current hardware and time constraints, it is not feasible to compute the VR filtration of Takens' embedding as previously described. With this in mind, we place several mild constraints in order to reduce the computational load. A stride of $s = 1$ is used, unless $\chi(x, \tau = 1, d = 2, s = 1) > 1000$, in which case the stride is increased until the number of points is less than 1000. This is done to ensure the VR filtration for each sample, in each dataset, can be computed in a reasonable amount of time. Once the stride is selected, τ is selected by minimizing time-delayed mutual information, and d is selected using the false nearest neighbors heuristic [45,46]. To further ensure that the filtration can be quickly computed, we utilize principal component analysis (PCA)

to create a 3-dimensional point cloud. The 1-dimensional persistent homology is then computed over the resulting point cloud for each sample.

SLSP is fast to calculate and non-parametric. It is therefore computed over the entire archive as previously described using Dionysus 2. In order to select between the two, we must choose a representation. The Betti curve is the fastest and simplest representation we have at our disposal, and is therefore used to create 1-NN classifiers from the VR and SLSP persistence diagrams. Specifically, we use a discretized Betti curve with 100 bins. The result is treated as a 100-dimensional vector, and classified using the 1-NN, with distance measured as the $p = 2$ Minkowski distance. SLSP significantly outperforms Takens' embedding, as shown in Figure 3.1.

In order to improve accuracy, we now optimize the persistence representation used in conjunction with SLSP. This is done by choosing the representation and parameters which are most accurate on the training set, for each data set. The range of parameters for each representation are shown in Table 3.1. This leads to 134 unique 1-NN classifiers for each of the 128 datasets. These parameters were chosen in order to fairly assess each representation while also balancing resource constraints. It is always possible to conduct a more thorough grid search of the parameters, but we believe any expansion will have diminishing returns.

The results of the initial SLSP and Taken's classifiers, the optimized SLSP classifier, and the baseline UCR classifiers are compared via a critical difference diagram in Figure 3.1. The diagram clearly shows that, even with an optimized representation, SLSP cannot compete with the worst performing baseline approach.

Table 3.1: **Grid Search Parameters.** Parameter p refers to the Minkowski distance

Parameters	
	Betti Curve
p	1,2
$bins$	10,50,100,250,500
	Persistence Image
p	1,2
bins	4,8,16,32
σ	.001, .01, .1, 1.0, 10
	Heat
p	1,2
bins	4,8,16,32
σ	.001, .01, .1, 1.0, 10
	Landscape
p	1,2
bins	10,50,100
layers	1,2
	Silhouette
p	1,2
bins	10,50,100
power	1,2

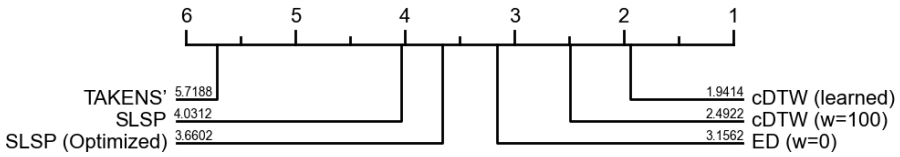

Figure 3.1: Purely topological methods compared to the three baseline methods of UCR archive. SLSP performs better than Takens' embeddings, and is improved by optimization of the representation parameters. Even optimized, it falls short of the baseline methods.

This would not necessarily be an issue if SLSP performed in a highly predictable way. Unfortunately, as shown in Figure 3.2, for a given dataset it cannot be predicted from the training set if optimized SLSP will perform better than the baseline SLSP. This indicates that SLSP-based distances are not well suited for use on the UCR archive. Even after a thorough search, the performance is lacking and unpredictable.

While it may be possible to create better TDA-based distances to use with a 1-NN classifier, it is not immediately obvious how that might be done. The presented experiments use the most commonly available tools from TDA. Seeking to address this lack of performance, we instead focus our attention on Rocket family classifiers, which in some ways mirror SLSP.

3.2 TopRocket

TopRocket is an extension of MultiRocket, adding a new topological pooling operator to the existing set of four operators. Recall that for each combination of kernel/bias/dilation/padding, MultiRocket extracts four features from a given time series. These features are: PPV, the length of the longest stretch above the bias,

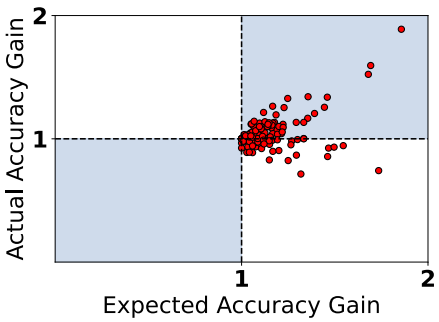

Figure 3.2: Optimized SLSP compared to baseline SLSP. For all points $x >= 1$, which indicates that the training dataset performance for optimized SLSP is at least as good as the baseline SLSP. The points, however, are almost evenly split by $y = 1$, indicating performance over training data does a poor job of predicting test performances.

the average value of the points above the bias, and the mean index of the points above the bias. The same features are extracted from the transformation of the first order difference of the time series, yielding an additional four features.

Recall further, that for each combination of kernel/dilation/padding, multiple bias values are computed from the quantiles of a random training set sample. Each bias is effectively a horizontal line which cuts through the transformed time series. The four MultiRocket pooling operators solely evaluate the portion of the transformed time series above the line. This process is highly evocative of the 0-dimensional persistent homology provided by SLSP, in which the number of connected components below a horizontal line is tracked, as the line sweeps from the bottom to the top of the time series. However, there is not an efficient or an intuitive way to incorporate the full SLSP process into MultiRocket.

Instead, we compute the number of connected components above the horizontal line specified by a bias value and include this as the fifth feature: the Betti

pooling operator. This is precisely the value of the Betti curve of the negative of the transformed time series taken at the negative bias value. Using the negative bias value and negative time series is a small adjustment in order to count the number of components above the horizontal line specified by b, instead of below it as with normal SLSP.

Definition 3.1 (Betti Pooling Operator) *Let x be a time series and $x_{min} \leq b \leq x_{max}$ a bias value. $B(-b) = \#D_{-b}$ is the Betti pooling operator, where D is the 0-dimensional persistence diagram of ASC, T(-x).*

In practice this value is very efficient to compute, only requiring a small change to MultiRocket, while significantly improving accuracy. Specifically, to extract the four preexisting features, MultiRocket already loops over the values of each transformed time series for each bias value. This is the inner most loop of the algorithm used. In order to create TopRocket, we simply add an additional condition to this loop to check how many times the transformed time series crosses the bias. It should also be noted that many pooling operators and combinations of operators were attempted when initially designing MultiRocket, so is difficult to find a new operator which does not impede performance.

This is a fast, linear-time process which integrates seamlessly with MultiRocket. In fact, because TopRocket requires fewer transformed time-series than MultiRocket, it is about that 5% faster. This speedup is verified on a test system using an AMD Ryzen Threadripper 1920X, with 12 parallel Numba threads, by averaging the time each classifier takes to run over all 30 splits. MultiRocket takes

an average of 346.9 seconds, while TopRocket only takes 326.7 seconds. This means TopRocket provides better results in less time.

We now present several results to support our claim that TopRocket is state-of-the-art and an improvement over MultiRocket. For this, we use the same experiment design as MultiRocket. Specifically, we run TopRocket over 30 different splits of 109 of the UCR datasets, and average the accuracy over all 30 runs. The results for the same experiment for the current top five classifiers are published as part of MultiRocket. These results are publicly available, and hosted with the code for the MultiRocket. Additionally, the selection of 109 datasets and 30 splits are also publicly available. Only 109 datasets are evaluated due to various constraints of each of the top five classifiers. For instance, MultiRocket only works with datasets that have fixed length time series.

We also adhere to the development practices used when designing the previous Rocket family classifiers, and only use 40 of the datasets as "development" datasets, used to tune the hyperparameters of the model. This is done in order to avoid overfitting to the UCR archive, which reduces its overall utility in the long run. The only hyperparameter to tune, with respect to TopRocket, is the choice of pooling operators. For all experiments we follow MultiRocket and fix the length of the feature vector as $50,000$. As we use five pooling operators, we only require $5,000$ $(50000/(2*5))$ kernel/bias/dilation/padding combinations, where as MultiRocket requires 6250 $(50000/(2*4))$.

As we have already revealed, TopRocket uses all of the original MultiRocket pooling operators. We verify this is the right choice by running all combinations of

the existing features with the Betti pooling operator over the development set. The resulting ranking shown in Table 3.2 shows that using all four MultiRocket features plus the Betti pooling operator is the best choice.

Table 3.2: **TopRocket Feature Choice Ranking:** All classifiers use the Betti pooling operator. The number represents which MultiRocket features are used: PPV(1), length of max positive stretch (2), mean positive value (3), and mean index (4). Ranking is taken over the standard 40 development sets.

1	TopRocket - 1234	6.2125
2	TopRocket - 134	6.3375
3	TopRocket - 124	6.9500
4	TopRocket - 234	7.1375
5	TopRocket - 14	7.4500
6	TopRocket - 123	7.7375
7	TopRocket - 13	7.7875
8	TopRocket - 1	8.4500
9	TopRocket - 34	8.6625
10	TopRocket - 12	8.8500
11	TopRocket - 3	9.0125
12	TopRocket - 23	9.4125
13	TopRocket - 24	9.7000
14	TopRocket - 4	10.0875
15	TopRocket - Betti Only	10.9625
16	TopRocket - 2	11.2500

Now we present the core result, the critical difference diagrams of the state-of-the-art classifiers, including TopRocket. Figure 3.3 shows that TopRocket displaces MultiRocket in the rankings. The critical difference diagram requires special attention to read, as the black bars surrounding MultiRocket, TopRocket, and HC2

may be confusing. Specifically, there is a bar between TopRocket and HC2 indicating that the two are not statistically significantly different. There is also a bar between MultiRocket and HC2, indicating the same. However, there is not a bar between TopRocket and MultiRocket, indicating TopRocket is significantly better. The bar connecting MultiRocket to HC2 may appear to indicate a clique including all three classifiers in question, but it does not. This is evidenced by the need for the clarifying bar between TopRocket and HC2, and can be independently verified by examining the pairwise statistical tests using the results provided.

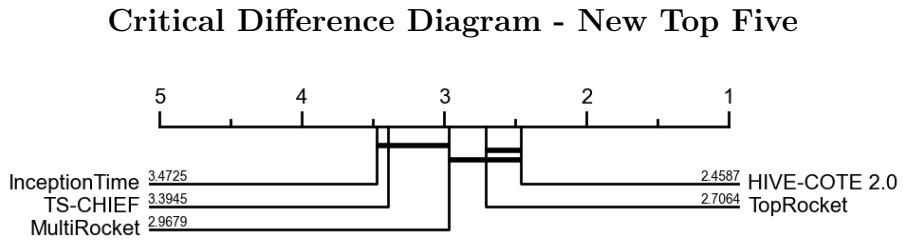

Figure 3.3: TopRocket outperforms MultiRocket, removing MiniRocket from the top five. TopRocket does **not** form a clique with MultiRocket.

The added performance of TopRocket is made even more clear when MultiRocket is replaced with TopRocket. Figure 3.4 shows the original critical difference diagram of the top five classifiers. Figure 3.5 shows the same diagram, but with TopRocket replacing MultiRocket. Notice that the bar spanning from InceptionTime to MultiRocket is not present in the new diagram.

Additionally, TopRocket has superior performance when competing directly against HC2, winning on 45 datasets, more than MultiRocket's 43 wins for the same task. When put head-to-head, TopRocket wins against MultiRocket on 63 datasets. These results are visualized in the Win/Tie/Loss graph in Figures 3.6 and

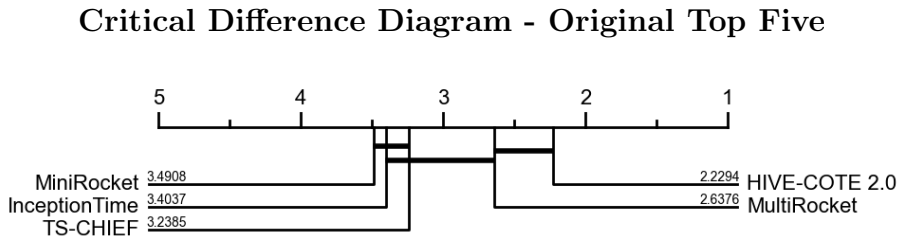

Figure 3.4: Note the clique formed between Inception Time and MultiRocket.

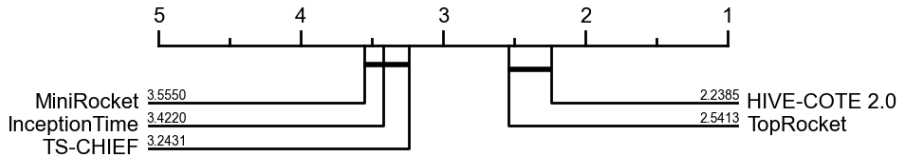

Figure 3.5: TopRocket only forms a clique with HC2.

3.7

These results rigorously show that TopRocket is now the best performing scalable time-series classifier over the UCR archive. Additionally, TopRocket empirically proves that TDA-based features can be used to provide an edge on highly competitive machine learning tasks. Additionally, we demonstrate that this benefit can be obtained without incurring the high computational costs typically associated with methods from TDA.

TopRocket vs. HIVE-COTE 2.0 Win/Tie/Loss: 45/6/58

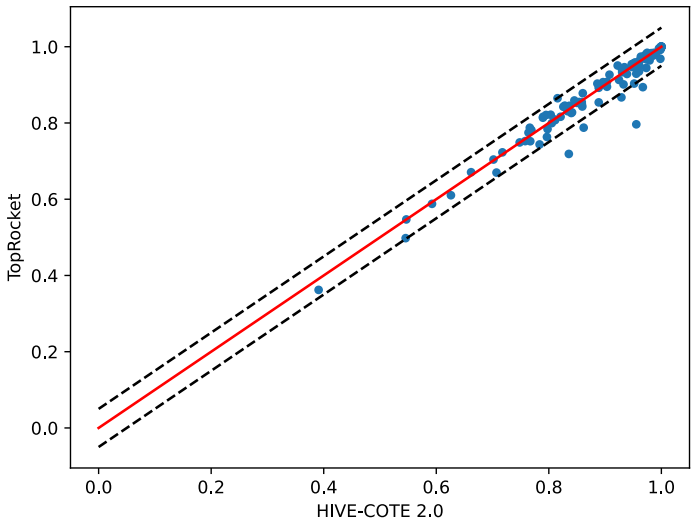

Figure 3.6: TopRocket does not outperform HC2, but it is the closest competitor. Points outside the dotted lines indicate a greater than 5% accuracy difference.

TopRocket vs. MultiRocket Win/Tie/Loss: 63/8/38

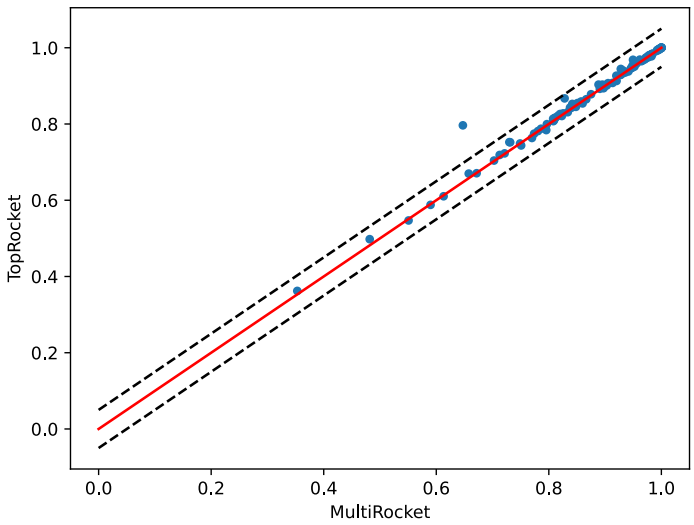

Figure 3.7: TopRocket outperforms MultiRocket by a significant margin. Each point is a data set, plotted by the accuracy of the respective classifiers. There are 63 points above the diagonal, indicating TopRocket wins on 63 of the 109 datasets.

Chapter 4: Conclusion

We set out to design and rigorously evaluate a fast, accurate, and reproducible TDA-based time-series classification method. With TopRocket, we accomplish this goal. Furthermore, we establish a clear path for further work in this direction. The connection of TopRocket, and specifically the Betti pooling operator, to the persistence curve framework serves to firmly connect the fields of TDA and time series classification.

While the connection between persistence curves and time series classification already existed via a hybrid DTW distance, we provide several key novel additions which reinforce this connection. The first is a method for extracting information about the Betti curve of a time series without having to compute persistent homology. The second is the method of incorporating this Betti curve SLSP information into an existing classifier framework. Taken together, these contributions provide state-of-the-art results over the UCR archive. Additionally, we provide a very fast implementation of the SLSP algorithm, which can be used in future work to incorporate additional persistence curves.

With this, we conclude with a call to action for TDA practitioners. The field of time-series classification will benefit from the additional attention of TDA

experts. There is significant work in TDA dedicated towards understanding SLSP and Takens' embedding of time series. Persistent development and application of this work, using the UCR archive as a rigorous benchmark, is likely to yield wholly new and useful results.

Appendix A: Algorithms

A.1 Persistent Homology Algorithm

Let ∂ be the filtered boundary matrix of a filtered simplicial complex over an ASC K. ∂ is an $n \times n$ square matrix, where n is the number of simplices in K. The rows and columns of *partial* correspond to the simplices sorted ascending by filtration order, and subsequently coface relation if needed for simplices added at the same filtration time. $\partial[i, j] = 1$ if simplex i is a proper coface of one dimension lower of simplex j, 0 otherwise.

Algorithm 1 Persistent Homology Reduction

Ensure: $x \geq 0$

```
1: procedure ComputePersistentHomology(∂)
2:     function Low(j)                ▷ Last coface of j to enter the filtration
         return Row index of lowest entry containing a 1 of column j.
3:     end function
4:     function Reduce(∂)
5:         R ← ∂
6:         for j=1 dom
7:             while there exists j_0 < j with Low(j_0) = Low(j) do
8:                 add column j_0 to column j
9:             end while
10:        end for
11:        return R
12:    end function
13: end procedure
```

For each non-zero column of R, j, if $i = low(j) \neq 0$ and σ_i is a $(p-1)$-simplex,

where σ_j is a p-simplex, there is a persistent homology class born at time a_i which dies entering time a_j. In other words, given some mild conditions, each non-zero column R describes a chain which bounds a whole. The last simplex added to that chain, σ_i, is the simplex that formed the hole. Meaning the hole formed at the time σ_i was added, a_i.

A.2 Fast SLSP Implementation

The speed of this implementation is mainly attributed to bypassing the matrix representation of the filtration boundary. In order to understand how this is done, we must first understand the boundary matrix, ∂, a time series. Each edge is bound by two vertices. Furthermore, the vertices have no boundary, so the corresponding columns in ∂ will always be 0 and have no impact on the calculation, so can be removed. Likewise, the edges are never the boundary of a chain, and so can be excluded from ∂. Already this is a 3/4 space savings, as the boundary matrix size shrinks from $(n + (n-1))^2$ to $n * (n-1) = n^2 - n$.

Examining the columns of ∂, it is clear that each only has two 1's, corresponding to the bounding vertices. The column addition operation in the persistence algorithm will always cancel out the lowest 1 in the column being added to. Depending on the value of the columns being added, this will leave the column with 0, 1, or 2 1's. We do not need the column representation to carry out these operations. We can simply track when each vertex enters each edge using lists and hash maps. The persistence algorithm is still the same, just accelerated over these data

structures using Numba.

Adding two columns together is a straightforward update to the lists containing the boundaries of each edge. Several levels of indirection are required to convert between the time in the filtration and the indices in the lists tracking the boundaries of the edges. This additional complexity allows for a significant speedup, as shown in Figure A.1. Additionally, it is possible to only return the k longest lived persistence points, a valuable feature in some contexts.

The speed of this implementation is directly dependent on the use of hash maps in Python and loop acceleration from Numba. It is not the algorithmically fastest approach, but it is empirically the fastest. As such, we share the source code instead of the pseudocode.

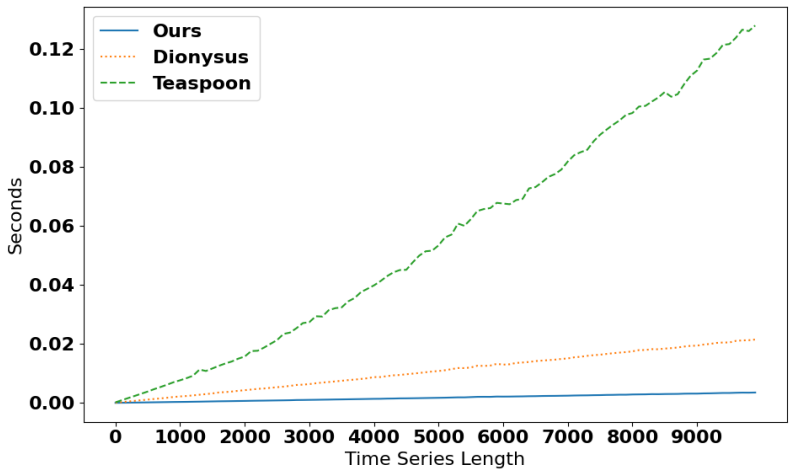

Figure A.1: Run time comparison of three methods for computing 0-dimensional SLSP of a time series. Random time series from length $10 - 10000$ are tested, increasing the length in steps of 100. The time is the result of averaging run time over 50 runs.

```python
# Python 3.6.9
import numpy as np # version 1.19.4
from numba import njit, uint32 # version 0.53.1

# Numbda decorator for JIT compilation
@njit(fastmath=True, cache=True)
def TimeSeriesSLSP(ts, n_points):
    """
    Accepts a time series a returns the n_points
    longest lived 0-dimensional persistence pairs.
    If the entire SLSP PD is needed, n_points
    should be set to the length of the time series.
```

```
    For a time series, there is only ever on infinite
    persistence point.
    It corrsponds to the minimum of the time series,
    and is not returned.
    This method does not return 0-life persistence points,
    as they do not have meaning.
    Any 0 pairs in the output should be ignored.
    Additionally, it assumed all value of the time series are >= 0.
    If the time series has negative values,
    it should first be shifted to be positive.
    The shift can be reversed as a post-processing step.

    This approch does not use a boundary matrix.
    Drawing out the boundary matrix for a time series
    makes it clear that the "low" indices can be
    computed directly from the time series.
    """

    # v is an abbreviation for vertex.
    v_ord = np.argsort(ts)
    rank_ord = {}

    # There are library functions to find rank order, but this is
    faster due to Numba.
    for i in range(len(v_ord)):
        rank_ord[v_ord[i]] = i

```

```
    n_edges = len(ts)-1

    # e is an abbreviation for edge, the 1-simplices between sample
    points.
    e_max_val = np.zeros(n_edges)

    e_birth_vert = np.zeros(n_edges).astype(uint32)
    e_death_vert = np.zeros(n_edges).astype(uint32)

    # Find the rank of when each edge was born and died.
    for i in range(n_edges):
        e_max_val[i] = np.max(ts[i:i+2])
        if ts[i] > ts[i+1]:
            e_birth_vert[i] = rank_ord[i+1]
            e_death_vert[i] = rank_ord[i]
        else:
            e_birth_vert[i] = rank_ord[i]
            e_death_vert[i] = rank_ord[i+1]

    # The edges indices sorted by their maximum vertex value.
    e_max_ord = np.argsort(e_max_val)

    # The index of high vertex for each edge.
    e_high_idx = e_birth_vert[e_max_ord]
    # The index of low vertex for each edge.
    e_low_idx = e_death_vert[e_max_ord]

```

```
    lows = {}
    for i in range(n_edges+1):
        lows[i]=-1

    # Apply persistent homology reduction.
    for i in range(n_edges):
        while True:
            low = e_low_idx[i]
            low_idx = lows[low]
            if  low_idx == -1:
                lows[low] = i
                break
            else:
                cur_high = e_high_idx[i]
                added_high = e_high_idx[low_idx]
                if cur_high > added_high:
                    e_low_idx[i] = cur_high
                    e_high_idx[i] = added_high
                else:
                    e_low_idx[i] = added_high

    # Compute the death times.
    death_times = e_max_val[e_max_ord]
    # Compute the life times.
    life_times = death_times - ts[v_ord[e_low_idx]]

    # The persistence diagram.
```

```
92      # Ignores infinite and 0 life points.
93      pd = np.zeros((n_points,2))
94
95      if len(life_times) == 0:
96          # return early if no persistence points.
97          return pd
98
99      # Count number of persistence points.
100     num_non_zero = 0 # This be returned to assist with downstream
    calculations
101     for i in range(len(life_times)):
102         if life_times[i] != 0:
103             num_non_zero += 1
104
105     if num_non_zero == 0:
106         # return early if no persistence points.
107         return pd
108
109     # Return the persitence diagram in life/death coordainates.
110     # death-life = birth if needed.
111
112     pd_sort = np.argsort(life_times)[::-1][:n_bars]
113     pd[:len(life_times),0] = life_times[pd_sort[:n_bars]]
114     pd[:len(life_times),1] = death_times[pd_sort[:n_bars]]
115
116     return pd
```

Appendix B: Results

B.1 Full TopRocket Results

The full average results of TopRocket over the same 109 datasets and 30 splits as MultiRocket. The development datasets are in bold.

Name	**Average Accuracy**
ACSF1	0.8309999982515971
Adiac	0.8209718604882558
ArrowHead	0.9032380898793538
BME	1.0
Beef	0.7633333365122478
BeetleFly	0.8949999988079071
BirdChicken	0.903333326180776
CBF	0.9941481550534567
Car	0.9266666650772095
Chinatown	0.9667638500531515
ChlorineConcentration	0.7814149300257365
CinCECGTorso	0.9684541046619415

Coffee	1.0
Computers	0.8547999997933705
CricketX	0.826153842608134
CricketY	0.8526495695114136
CricketZ	0.8431623816490174
Crop	0.7746706306934357
DiatomSizeReduction	0.9506535847981771
DistalPhalanxOutlineAgeGroup	0.8163069566090901
DistalPhalanxOutlineCorrect	0.8454106112321218
DistalPhalanxTW	0.7040767431259155
ECG200	0.8923333366711934
ECG5000	0.9468592564264934
ECGFiveDays	0.9953155259291331
EOGHorizontalSignal	0.864917121330897
EOGVerticalSignal	0.8162062605222066
Earthquakes	0.7489208579063416
ElectricDevices	0.9020317355791728
EthanolLevel	0.7188000023365021
FaceAll	0.9840039392312367
FaceFour	0.8939393897851308
FacesUCR	0.9693658550580343

FiftyWords	0.8425641119480133
Fish	0.9832380970319112
FordA	0.9588383654753367
FordB	0.9331275661786397
FreezerRegularTrain	0.998409362634023
FreezerSmallTrain	0.9917894721031189
GunPoint	0.9973333338896434
GunPointAgeSpan	0.9954641401767731
GunPointMaleVersusFemale	1.0
GunPointOldVersusYoung	1.0
Ham	0.848888897895813
Haptics	0.5469696978727977
Herring	0.6104166666666667
HouseTwenty	0.9775910377502441
InlineSkate	0.49787878692150117
InsectEPGRegularTrain	1.0
InsectEPGSmallTrain	0.9973226288954417
InsectWingbeatSound	0.670824917157491
ItalyPowerDemand	0.9647230366865794
LargeKitchenAppliances	0.9378666659196218
Lightning2	0.7437158465385437

Lightning7	0.7840182662010193
Mallat	0.964079600572586
Meat	0.9822222252686819
MedicalImages	0.8094298203786214
MiddlePhalanxOutlineAgeGroup	0.7229437271753947
MiddlePhalanxOutlineCorrect	0.8450171808401744
MiddlePhalanxTW	0.5878787795702617
MixedShapesRegularTrain	0.9807422757148743
MixedShapesSmallTrain	0.9574295540650686
MoteStrain	0.901144826412201
OSULeaf	0.9688705265522003
OliveOil	0.9077777663866679
PhalangesOutlinesCorrect	0.8590132097403208
Phoneme	0.36213079988956454
PigAirwayPressure	0.7964743574460348
PigArtPressure	0.9443910260995229
PigCVP	0.867147437731425
Plane	1.0
PowerCons	0.9751851816972097
ProximalPhalanxOutlineAgeGroup	0.8549593528111775
ProximalPhalanxOutlineCorrect	0.9071019411087036

ProximalPhalanxTW	0.8074796775976817
RefrigerationDevices	0.7520888884862263
Rock	0.8539999902248383
ScreenType	0.669866661230723
SemgHandGenderCh2	0.9492777665456136
SemgHandMovementCh2	0.7877037008603414
SemgHandSubjectCh2	0.9285185257593791
ShapeletSim	0.9994444429874421
ShapesAll	0.9462777694066365
SmallKitchenAppliances	0.8278222203254699
SmoothSubspace	0.9791111171245575
SonyAIBORobotSurface1	0.9485302150249482
SonyAIBORobotSurface2	0.9541098316510518
StarLightCurves	0.9812489867210388
Strawberry	0.9802702705065409
SwedishLeaf	0.9739733318487803
Symbols	0.9728308200836182
SyntheticControl	0.9945555607477824
ToeSegmentation1	0.9292397697766622
ToeSegmentation2	0.9351281921068827
Trace	1.0

TwoLeadECG	0.9977172871430715
TwoPatterns	1.0
UMD	0.9775463044643402
UWaveGestureLibraryAll	0.98406849304835
UWaveGestureLibraryX	0.8780662556489308
UWaveGestureLibraryY	0.8137632588545481
UWaveGestureLibraryZ	0.821673180659612
Wafer	0.9998053153355916
Wine	0.9129629611968995
WordSynonyms	0.7875653107961019
Worms	0.7523809512456258
WormsTwoClass	0.7995670914649964
Yoga	0.9403111080328623

Bibliography

[1] Zheng Zhang, Ping Tang, and Rubing Duan. Dynamic time warping under pointwise shape context. *Information sciences*, 315:88–101, 2015.

[2] Mengsen Zhang, William Kalies, Scott Kelso, and Emmanuelle Tognoli. Topological portraits of multiscale coordination dynamics. *Journal of Neuroscience Methods*, 339:108672, 06 2020.

[3] Paolo Barsocchi, Pietro Cassará, Daniela Giorgi, Davide Moroni, and Maria Antonietta Pascali. Computational topology to monitor human occupancy. In *Multidisciplinary Digital Publishing Institute Proceedings*, volume 2, page 99, 2018.

[4] Bastian Rieck Heike Leitte. Enhancing comparative model analysis using persistent homology. 2014.

[5] Joseph Collins, Michaela Iorga, Dmitry Cousin, and David Chapman. Passive encrypted iot device fingerprinting with persistent homology. In *NeurIPS 2020 Workshop on Topological Data Analysis and Beyond*, 2020.

[6] Joseph R Collins. Network anomaly detection via persistent homology. *UMBC Student Collection*, 2019.

[7] Enron. Enron email dataset. http://www.cs.cmu.edu/~enron/enron_mail_20150507.tar.gz, 2015.

[8] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, November 2019.

[9] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The UCR time series classification archive.

[10] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[11] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[12] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[13] Fuming Fang and Takahiro Shinozaki. Electrooculography-based continuous eye-writing recognition system for efficient assistive communication systems. *PloS one*, 13(2):e0192684, 2018.

[14] Robert Thomas Olszewski. *Generalized feature extraction for structural pattern recognition in time-series data*. Carnegie Mellon University, 2001.

[15] Ashit Gandhi. Content-based image retrieval: Plant species identification. *Oregon State University*, 2002.

[16] Yanping Chen, Adena Why, Gustavo Batista, Agenor Mafra-Neto, and Eamonn Keogh. Flying insect classification with inexpensive sensors. *Journal of insect behavior*, 27(5):657–677, 2014.

[17] Chang Wei Tan, Geoffrey I Webb, and François Petitjean. Indexing and classifying gigabytes of time series under time warping. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 282–290. SIAM, 2017.

[18] O Al-Jowder, EK Kemsley, and RH Wilson. Mid-infrared spectroscopy and authenticity problems in selected meats: a feasibility study. *Food Chemistry*, 59(2):195–201, 1997.

[19] Romain Briandet, E Katherine Kemsley, and Reginald H Wilson. Discrimination of arabica and robusta in instant coffee by fourier transform infrared spectroscopy and chemometrics. *Journal of agricultural and food chemistry*, 44(1):170–174, 1996.

[20] Jingkun Gao, Suman Giri, Emre Can Kara, and Mario Bergés. Plaid: a public dataset of high-resoultion electrical appliance measurements for load identification research: demo abstract. In *proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 198–199, 2014.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[22] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. MultiRocket: Multiple pooling operators and transformations for fast and effective time series classification. *arXiv preprint arXiv:2102.00457*, 2021.

[23] Ryan M Rifkin and Ross A Lippert. Notes on regularized least squares. 2007.

[24] Angus Dempster, François Petitjean, and Geoffrey I Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[25] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 248–257, August 2021.

[26] Jason Lines, Sarah Taylor, and Anthony Bagnall. HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1041–1046, Barcelona, Spain, December 2016. IEEE.

[27] Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. HIVE-COTE 2.0: A new meta ensemble for time series classification. *arXiv:2104.07551 [cs]*, April 2021.

[28] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for Time Series Classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, November 2020.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[30] Mathieu Carrière, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, and Yuhei Umeda. Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In *International Conference on Artificial Intelligence and Statistics*, pages 2786–2796. PMLR, 2020.

[31] Christoph D Hofer, Roland Kwitt, and Marc Niethammer. Learning representations of persistence barcodes. *J. Mach. Learn. Res.*, 20(126):1–45, 2019.

[32] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I Webb. TS-CHIEF: A scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, 34(3):742–775, 2020.

[33] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey I Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 33(3):607–635, 2019.

[34] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010.

[35] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[36] Ulrich Bauer. Ripser: efficient computation of vietoris–rips persistence barcodes. *Journal of Applied and Computational Topology*, 5(3):391–423, 2021.

[37] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Perez, Matteo Caorsi, Anibal M Medina-Mardones, Alberto Dassatti, and Kathryn Hess. Giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration. page 6.

[38] Yu-Min Chung and Austin Lawson. Persistence curves: A canonical framework for summarizing persistence diagrams. *arXiv preprint arXiv:1904.07768*, 2019.

[39] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence Images: A Stable Vector Representation of Persistent Homology. page 35.

[40] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4741–4748, 2015.

[41] Peter Bubenik et al. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(1):77–102, 2015.

[42] Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. Stochastic convergence of persistence landscapes and silhouettes. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 474–483, 2014.

[43] Floris Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381. Springer, 1981.

[44] Jose A Perea, Anastasia Deckard, Steve B Haase, and John Harer. SW1PerS: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data. *BMC bioinformatics*, 16(1):1–12, 2015.

[45] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical review A*, 45(6):3403, 1992.

[46] Audun Myers, Elizabeth Munch, and Firas A Khasawneh. Persistent homology of complex networks for dynamic state detection. *Physical Review E*, 100(2):022314, 2019.

[47] Audun D Myers, Melih Yesilli, Sarah Tymochko, Firas Khasawneh, and Elizabeth Munch. Teaspoon: A comprehensive python package for topological signal processing. In *NeurIPS 2020 Workshop on Topological Data Analysis and Beyond*, 2020.

[48] Audun D Myers, Firas A Khasawneh, and Brittany T Fasy. Separating persistent homology of noise from time series data using topological signal processing. *arXiv preprint arXiv:2012.04039*, 2020.

[49] Dmitriy Morozov. Dionysus 2 – library for computing persistent homology. https://github.com/mrzv/dionysus, 2018.

[50] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery.

[51] Yu-Min Chung, William Cruse, and Austin Lawson. A persistent homology approach to time series classification, 2020.

[52] Nieves Atienza, Rocío González-Díaz, and Manuel Soriano-Trigueros. On the stability of persistent entropy and new summary functions for topological data analysis. *Pattern Recognition*, 107:107509, 2020.

[53] Yu-Min Chung and Sarah Day. Topological fidelity and image thresholding: A persistent homology approach. *Journal of Mathematical Imaging and Vision*, 60(7):1167–1179, 2018.