ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

TIME SERIES CLASSIFICATION VIA TOPOLOGICAL DATA ANALYSIS

Ph.D. THESIS

Alperen KARAN

Department of Mathematical Engineering

Mathematical Engineering Programme

JUNE 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

TIME SERIES CLASSIFICATION VIA TOPOLOGICAL DATA ANALYSIS

Ph.D. THESIS

Alperen KARAN
(509152201)

Department of Mathematical Engineering

Mathematical Engineering Programme

Thesis Advisor: Prof. Dr. Atabey KAYGUN

JUNE 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

TOPOLOJİK VERİ ANALİZİ İLE ZAMAN SERİLERİNİN SINIFLANDIRILMASI

DOKTORA TEZİ

Alperen KARAN
(509152201)

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Atabey KAYGUN

HAZİRAN 2022

Alperen KARAN, a Ph.D. student of ITU Graduate School student ID 509152201 successfully defended the thesis entitled “TIME SERIES CLASSIFICATION VIA TOPOLOGICAL DATA ANALYSIS”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Atabey KAYGUN** Istanbul Technical University	.....
Jury Members :	**Prof. Dr. Mustafa NADAR** Istanbul Technical University	.....
	Prof. Dr. Süreyya AKYÜZ Bahçeşehir University	.....
	Assoc. Prof. Dr. Özgür MARTİN Mimar Sinan Fine Arts University	.....
	Asst. Prof. Dr. Gül İNAN Istanbul Technical University	.....

Date of Submission : **3 June 2022**
Date of Defense : **22 June 2022**

To Şeyda,
children of the world,
science,
and curiosity.

FOREWORD

First and foremost, I would like to express my extreme gratitude to my advisor Prof. Atabey KAYGUN for his continuous encouragement, patience and support. Thank you for pushing me beyond what I thought was possible. He has always been an invaluable source of inspiration for me.

I am grateful to Prof. Mustafa NADAR and Prof. Süreyya AKYÜZ for their insightful comments and positive attitudes during the long thesis supervision process. I would also like to express my sincere thanks to Assoc. Prof. Özgür MARTİN and Asst. Prof. Gül İNAN for kindly accepting to be members of my defense committee and for their helpful suggestions.

I am grateful to Göksu ORUÇ for suggesting me to pursue my PhD studies with Prof. KAYGUN on Topological Data Analysis. I would like to thank Yekta Said CAN for pointing me to the datasets which later became the main focus of my studies. I would also like to thank İsmail GÜZEL for our long and insightful discussions on TDA and persistent homology.

Special thanks to all faculty members and my research assistant friends at ITU Department of Mathematics. Thank you for providing me the nicest environment to work at.

I am indebted to my dearest friends Muhammet MEMİÇ, Vedat KURTAY, Ömer AKTEPE, Gökhan GÖKSU, Tuğba AYAN, Şeyma KARADERELİ, Şeyda ÇETİNTAŞ and Celil Ahmet CEYLAN for their nicest friendship and support. They have always been a source of joy in my life.

Bana inanıp bugüne kadar destekleyen annem Jülide ve babam Erhan KARAN'a teşekkürlerimi sunuyorum. Kız kardeşlerim Aybüke ve Alanur KARAN'a da sevgi ve destekleri için teşekkür ediyorum.

Aramızdan zamansızca ayrılan aile üyelerimizi hüzün, rahmet, özlem ve saygıyla anıyorum.

My heartfelt thanks are reserved for my dearest wife Şeyda (UÇAR) KARAN. She is the greatest source of excitement in my life. I am grateful to you for understanding me and always finding a way to make me happy. Eşimi bugünlere getiren tüm aile üyelerine de teşekkürlerimi sunuyorum.

June 2022

Alperen KARAN
(Data Scientist)

TABLE OF CONTENTS

Page

ABBREVIATIONS

TDA	**:** Topological Data Analysis
LOSOCV	**:** Leave One Subject Out Cross-Validation
RF	**:** Random Forest classifier
SVC	**:** Support Vector Machines classifier
LDA	**:** Linear Discriminant Analysis
AB	**:** AdaBoost Classifier
ACC	**:** Accelerometer
ECG	**:** Electrocardiogram
GSR	**:** Galvanic skin response
HR	**:** Heart rate
EMG	**:** Electromyography
RESP	**:** Respiration
EDA	**:** Electrodermal activity
TEMP	**:** Temperature
BVP	**:** Blood volume pulse

SYMBOLS

$\mathbb{N}$	**:** Natural numbers
$\mathbb{Z}$	**:** Integers
$\mathbb{R}$	**:** Real numbers
W_∞	**:** Bottleneck distance
W_p	**:** $p-$Wasserstein distance
fs	**:** Sampling frequency

LIST OF TABLES

Page

LIST OF FIGURES

Page

TIME SERIES CLASSIFICATION VIA TOPOLOGICAL DATA ANALYSIS

SUMMARY

This dissertation aims to demonstrate the power of Topological Data Analysis (TDA) and the subwindowing method for feature engineering in time series classification tasks. As an application, we used two publicly available datasets, WESAD and DriveDB. These datasets consisted of physiological signals collected under stressful and non stressful events. Furthermore, in order to assess the reliability of our methodology, we tested our feature engineering methods on a synthetic dataset that consists of artificial physiological signals mimicking a stress detection study. The results indicated that automatically created topological features can yield higher classification accuracies than signal-specific and hand-crafted features (such as heart rate derived from an ECG signal).

In the first chapter of this work, we briefly summarize TDA and persistent homology. Also, the methods for time series classification via persistent homology is discussed, and we make a literature review on the subject.

The second chapter is devoted to time series methods, and how we can classify them. We first define the method of sliding windows, and discuss why it can be useful in machine learning tasks. Then, we talk about time delay embeddings which transforms a univariate time series into a high dimensional dataset. We illustrate how the topology of the resulting dataset is affected by the delay parameter (also known as the embedding dimension). At the end of this chapter, we introduced the subwindowing methodology which solved the main problem of this work. We showed that this method allows us to reduce noise, improve computation time by a large amount, and use longer windows without incurring extra computational cost.

In the third chapter, the theoretical background for TDA and persistent homology is given. The chapter starts with discussing why we should see the data at different scales. Then we give preliminary definitions related with simplices and simplicial complexes. We state the Nerve theorem and talk about how a topological space and a simplicial complex can be homotopy equivalent under some assumptions. This theorem tells us that the Cech (and therefore Rips) complexes are topologically similar to the underlying object that the dataset was sampled from. Later in this chapter, we define simplicial homology and show how we can compute the homology of a simplicial complex. Note that we need a fixed distance (epsilon) parameter to build a simplicial complex on top of a dataset. On the other hand, persistent homology allows us to investigate the persistence of homology groups when epsilon varies. After presenting how persistent homology works, we define persistence diagrams and two widely used metrics between them. Also, we show that persistence diagrams are stable under small perturbations of the data. Lastly, we show some means of performing feature engineering of persistence diagrams.

The fourth chapter consists of the description of the datasets used in this dissertation and our methodologies. First, we introduce the three datasets (synthetic, WESAD and DriveDB) used in this study. For the synthetic dataset, there were two classes of physiological signals: stress and non stress. The classes for WESAD were baseline, amusement and stress. For DriveDB, the classes were relax, driving in the highway (low stress) and driving in the city (high stress). We then talk about the physiological signals included in the datasets, their sampling frequencies, and some preprocessing we did beforehand. Our experiments had some parameters such as window size, subwindow size, the embedding dimension in time delay embeddings. Later in this chapter, we discuss how these parameters were chosen, and how we did feature engineering for our experiments. Then, we present the machine learning algorithms and their hyperparameters used in our experiments.

Lastly in chapter four, we introduce the two cross-validation methodologies used in our experiments. For Leave-one-subject-out cross validation (LOSOCV), the model is trained on all subjects but one, and tested on the other. When each participant appears in the test set once and only once, the results are averaged. This cross validation technique tells us about the model's performance on a previously unseen subject. For intra-subject cross validation, we split each subjects data into two. We train on either half, and test on the other, then average the results. We get a final accuracy by averaging all accuracies obtained from each participant. This method shows whether the model can benefit from having the same subject's data on both the train and the test sets.

The results of the experiments are covered in the fifth chapter. We presented the results for the synthetic, WESAD and DriveDB datasets, respectively. The results for the synthetic dataset indicated that as the magnitude of the physiological change that mimics stress increases, stress detection accuracy also improves. For example, when the heart rate variability -an important stress indicator- is raised, the topological features could detect it almost perfectly. The results imply that stress detection errors in real-world datasets can be attributed to the noisy nature of the dataset itself, rather than the topological features. For example, such lack of effect can appear when some participants do not react to the stress condition.

When the results from the real datasets were investigated, we usually observed the highest affect recognition accuracies when features coming from all persistence diagrams (level sets and delay embeddings) are used. Nevertheless, using only one persistence diagram (resulting in much fewer features) we were able to achieve similar recognition performance. This tells us that high accuracies are attainable using a small number of automatically engineered topological features rather than hand-crafted signal-specific features. For the three-class tasks, we observed that stress conditions are well separated from other conditions. This result supports the hypothesis that topological features works pretty well in distinguishing chaotic time series from non chaotic ones. When we made a binary classification task (stress vs non stress), topological features again performed better than those used in the original studies for most of the physiological signals.

We have already stated that an important advantage of the subwindowing method is to be able to change the window size effectively. When we tested different window sizes, we observed that higher windows implied better stress detection performance. Furthermore, model performance with intra-subjects cross validation was significantly

higher than LOSOCV. This was an expected finding since the model can perform better on the test set when the data from the same subject appears in the training set.

Finally, in the sixth chapter, we outline our methodologies and their limitations. We also discussed what future works can aim for. For example, future studies can assess the performance of a model trained on one dataset and tested on another. Also, later research can use semi-supervised (rather than supervised) tasks for even improved accuracies. Lastly, one can use other vector representations of persistence diagrams for feature engineering.

TOPOLOJİK VERİ ANALİZİ İLE ZAMAN SERİLERİNİN SINIFLANDIRILMASI

ÖZET

Bu çalışma, zaman serilerinin sınıflandırılması için Topolojik Veri Analizi ve altpencereleme metodu ile özellik üretme yöntemlerinin gücünü göstermeyi amaçlar. Uygulama olarak, iki adet halka açık (WESAD ve DriveDB) veri kümesini kullandık. Bu veri kümeleri, stresli ve stressiz koşullar altında toplanmış fizyolojik sinyallerden oluşmaktaydı. Ayrıca, yöntemlerimizin güvenilirliğini anlamak için, özellik üretme yöntemlerimizi, stres belirleme çalışmalarını taklit eden ve sentetik fizyolojik sinyallerden oluşan bir veri seti üzerinde test ettik. Sonuçlar, otomatik olarak oluşturulmuş topolojik özelliklerin, sinyale özgü (ECG sinyalinden kalp atış hızı özelliği üretilmesi gibi) el yapımı özelliklerden daha yüksek doğrulukla sınıflandırma yapabileceğini gösterdi.

Bu çalışmanın birinci bölümü olan giriş bölümünde topolojik veri analizi ve kalıcı homoloji kısaca özetlenmiştir. Ayrıca, kalıcı homoloji ile zaman serileri sınıflandırmasının nasıl yapılabileceği tartışılmıştır ve konuyla ilgili yapılmış önceki çalışmalara değinilerek literatür taraması tamamlanmıştır.

İkinci bölümde, zaman serileri yöntemlerinden bahsedilerek, bunların sınıflandırmasının nasıl yapılabileceği tartışılmıştır. Kayan pencereler yöntemini tanıtarak, bunun makine öğrenmesi modellerinde niçin avantajlı olabileceğini dile getirdik. Ardından, zaman gecikme dönüşümünden bahsederek, bir zaman serisinin çok boyutlu bir veri kümesine nasıl dönüştürülebileceğini gösterip, oluşan veri kümesinin gecikme boyutunun farklı değerlerine göre topolojik olarak nasıl etkilendiğinden bahsedip bunu örneklendirdik. Bu bölümün sonunda ise, çalışmanın esas problemini çözen altpencereleme metodunu anlattık. Bu yöntemin, gürültünün giderilmesini, hesap süresinin önemli ölçüde azaltılmasını, ve büyük uzunluktaki kayan pencereleri hesap süresini artırmadan kullanılmasını sağladığını gösterdik.

Çalışmanın üçüncü bölümü topolojik veri analizi ve kalıcı homolojinin teorik altyapısını veriyor. Bölüm, elimizdeki veri kümesini niçin farklı ölçeklerde gözlemlememiz gerektiğini tartışarak başlıyor. Ardından simpleks ve basit kompleks ile ilgili temel tanımları veriyoruz. Nerve teoreminden bahsederek, bir topolojik uzay ile bir basit kompleksin bazı şartlar altında homotopi denk olduğunu gösteriyoruz. Bu teorem bize Cech (ve dolayısıyla Rips) komplekslerin, bir veri kümesinin örneklendiği objeye topolojik olarak benzediğini söylüyor. Daha sonra, basit homolojiden bahsederek, basit kompleksler üzerinde homoloji hesabının nasıl yapılabileceğini gösteriyoruz. Ne var ki, bir veri seti üzerine basit kompleks inşa etmek için sabit bir uzaklık parametresine (epsilon) ihtiyaç duyuyoruz. Kalıcı homoloji teorisi ise, farklı epsilon değerleri için oluşan farklı basit komplekslerdeki homoloji gruplarının hangi epsilon aralıklarında yaşadığını hesaplamamızı sağlıyor. Bu bölümde kalıcı homoloji hesabının nasıl yapıldığını gösterdikten sonra, kalıcılık

diyagramlarını ve bu diyagramlar arasında tanımlanabilecek metrikleri tanıtıyoruz. Ayrıca, kalıcılık diyagramlarının veri kümesindeki küçük sarsıntılara karşı kararlı olduğunu gösteriyoruz. Bu bölümde son olarak, kalıcılık diyagramlarından özellik üretme yöntemlerinden bazılarını gösterdik.

Dördüncü bölümde, yöntemlerimizi test ettiğimiz veri kümeleri ve kullandığımız yöntemler sunuluyor. Öncelikle, kullandığımız üç veri kümesini (sentetik, WESAD, ve DriveDB) tanıtıyoruz. Sentetik veri kümesinde stresli ve stressiz olmak üzere iki zaman serisi sınıfı kullanıldı. WESAD için zaman serisi sınıfları referans (rahat olma hali), eğlenme, ve stres şeklinde iken, DriveDB için bunlar rahatlama, otoyolda araba sürüş (düşük stres) ve şehirde araba sürüş (yüksek stres) şeklindeydi. Daha sonra her bir veri kümesinin içerdiği fizyolojik sinyalleri ve bunların örnekleme frekanslarını, hangi sınıflardan oluştuğunu, ve -varsa- ön işleme adımlarını anlatıyoruz. Deneylerimiz birtakım parametreler içeriyordu. Bunlara örnek olarak, pencere uzunluğu, altpencere uzunluğu, zaman gecikme dönüşümü boyutunu verebiliriz. Bu bölümde, bu parametrelerin nasıl seçildiğinden bahsedip, özellik üretmeyi bu deneyler özelinde nasıl yaptığımızı anlatıyoruz. Bundan sonra, deneylerde kullanılacak makine öğrenmesi modellerini, ve bunların hiperparametrelerini söylüyoruz.

Dördüncü bölümde son olarak, kullandığımız iki farklı çapraz geçerlilik yöntemini anlatıyoruz. Bir katılımcıyı dışarıda bırak çapraz geçerlilik yöntemi, makine öğrenmesi modeline biri hariç tüm katılımcıları eğitim setine koyup, modeli dışta kalan katılımcıda test ediyor. Her katılımcı bir defa test edildiğinde sonuçların ortalaması alınıyor. Bu yöntem bize, daha önce hiç görülmemiş bir katılımcı için modelin ne kadar kesin çalıştığını göstermektedir. Katılımcı içi çapraz geçerlilik yönteminde ise, her bir katılımcının verilerini ikiye bölüyor, sonrasında ise her iki yarımda ayrı ayrı eğitip, diğer yarımda test ederek ortalamasını alıyoruz. Her katılımcı için elde edilen skorların ortalaması alındığında, modelin genel performansını elde ediyoruz. Bu yöntem bize, modeli aynı katılımcının verisiyle eğitip test ettiğimizde model performansında bir artış olup olmadığını gösteriyor.

Çalışmamızın beşinci bölümü deney sonuçlarını içeriyor. Bu bölümde sırasıyla sentetik, WESAD, ve DriveDB veri kümelerinden elde edilen sonuçları inceledik. Sentetik veri kümesi sonuçlarına göre, taklit etmeye çalıştığımız stres durumunun etkisi artığında, stres belirleme kesinliği de arttı. Örneğin, stresin önemli bir indikatörü olan kalp hızı değişkenliği yüksek oranda arttığında, oluşturduğumuz topolojik özellikler bunu tam doğrulukla bulabildi. Bu durum, gerçek hayatta toplanmış veri kümelerindeki olası düşük kesinliğin kullandığımız özellik üretme yönteminden değil, veri kümesinin gürültülü doğasından kaynaklanabileceğini gösteriyor. Bu gürültü, bazı katılımcıların stres koşulu altında çok az fizyolojik stres tepkisi vermesi veya benzeri bir sebeple oluşabilir.

WESAD ve DriveDB veri kümelerinden elde edilen sonuçlar incelendiğinde, genellikle, tüm kalıcılık diyagramlarından elde edilen özelliklerin tamamı kullanıldığında en yüksek kesinlikle duygu sınıflandırması yapılabiliyor. Yine de, kesinlikten çok az miktarda ödün vererek, sadece bir kalıcılık diyagramı ile benzer sonuçları elde etmek mümkün olabildi. Bu bize gösteriyor ki, otomatik olarak oluşturulmuş ve sinyale özgü özelliklerden daha az sayıda olan topolojik özellikleri kullanarak yüksek bir kesinlik elde etmek mümkün olabiliyor. Üçlü sınıflandırma sonuçları için karışıklık matrisine baktığımızda stres sınıflarının diğer sınıflardan daha iyi ayırt edildiğini görüyoruz. Bu durum, topolojik özelliklerin kaotik ve kaotik olmayan zaman serilerini

birbirinden ayırt etmede işe yaradığı hipotezini destekliyor. Stres ve diğerleri şeklinde ikili sınıflandırma yapıldığında da sonuçlar yine çoğu fizyolojik sinyal için orijinal çalışmadaki bulgulardan daha iyi çıkıyor.

Altpencereleme metodunun önemli bir avantajının pencere boyutunu kolayca değiştirebilmek olduğunu söylemiştik. Bunu test etmek için uzun pencereler kullanıldığında modelin kesinliğinin arttığı gözlemlendi. Bunun yanı sıra, model performansının katılımcı içi çapraz geçerlilik yöntemi kullanıldığında, bir katılımcıyı dışarıda bırak çapraz geçerlilik yöntemine göre istatistiksel olarak anlamlı derecede yüksek olduğu gözlemlendi. Bu beklediğimiz bir sonuçtu, çünkü eğitim ve test kitlelerinde aynı katılımcı olduğu sürece model daha iyi performans gösterecektir.

Son bölüm olan altıncı bölümde kullandığımız yöntemler özetlenerek bunların kısıtlamaları anlatıldı. Gelecek çalışmalarda, bir veri kümesinde eğitilmiş modelin başka bir veri kümesinde test edilebileceği, gözetimli yerine yarı-gözetimli öğrenmenin daha yüksek kesinlikte sonuçlar üretebileceği, ve kalıcılık diyagramlarının farklı vektör temsilleri kullanılarak elde edilecek özellik üretme yöntemlerinin denenebileceğinden bahsedildi.

1. INTRODUCTION

This thesis demonstrates the effectiveness of persistent homology in time series classification tasks. In addition to creating sliding windows for the learning tasks, we also created sliding *subwindows* within each window. This made our topological features generated from the persistence diagrams obtained from the subwindows more robust to noise, and computationally much less expensive. The methods were tested on two publicly available datasets: WESAD [2] and DriveDB [3] that include physiological signals collected from human subjects under different affect and stress conditions. Furthermore, we also tested our methods on a synthetic dataset containing multiple time series mimicking physiological signals collected under stressful and non-stressful events. This dissertation is an extension of our previous study [4], and some passages, figures and tables in this dissertation have been quoted verbatim from our paper.

Typically, time series classification tasks involve time series of different and long durations. For example, in WESAD dataset, physiological measurements were collected from 15 participants under stressful and non stressful conditions ranging from 8 to 20 minutes. Feeding such datasets directly into a machine learning model is problematic because the number of training instances will be too small, and each training instance will have an infeasibly long duration. A common approach in such cases is to create sliding windows from each time series. Using this approach, one gets equally sized shorter time series which can be used as training instances. This method results in an augmented and more manageable dataset.

Topological Data Analysis (TDA) has become a common tool in data analysis tasks, especially when a dataset is assumed to lie on a fairly low dimensional object in a high dimensional space. A fundamental tool in TDA is *persistent homology* which generates a *persistence diagram* that contains critical topological information about the data, hence about the underlying object. Given a sequence of topological spaces, a persistence diagram keeps track of the birth-death times of topological features.

There are two main methods to compute a persistence diagram from a univariate time series. The first method is straightforward: we can directly compute the persistence diagram of the time series induced by the (sub- or super-) level sets (e.g. [5–7]). Alternatively, we can convert the time series to a higher dimensional dataset (called the time delay embedding) whose topological structure involves crucial clues about the time series itself. Then, we can compute the persistence diagram of the resulting dataset (e.g. [8–10]). This method has its theoretical guarantees from Taken's delay embedding theorem [11]. Unfortunately, persistent homology has a large computational complexity ($\mathcal{O}(n^3)$ for the standard algorithm). Therefore, topological feature extraction can be discouraging when the time series is long and hence, the time delay embedding is a large dataset. In such cases, subsampling was used for faster computation (e.g. [12, 13]).

Once a persistence diagram for each window is obtained, distance-based machine learning models (such as kNN) can be employed (e.g. [14, 15]). However, since finding distance between persistence diagrams is computationally expensive, such machine learning models can only be used if the number of training instances remains fairly low. Otherwise, one must resort to secondary methods to extract features from persistence diagrams first. In such cases, one must first engineer feature vectors and then use machine learning models on these feature vectors.

Feature engineering from a persistence diagram can be made purely on the statistical properties such as average birth or minimum death of the points in the persistence diagram (e.g. [9, 10]). However, such features are very sensitive to noise, so they can produce unreliable results when not used carefully. One can also use stable vector representations of a persistence diagram such as the Betti curve or the Persistence landscape. As these representations lie in a vector space, they can directly be fed into learning algorithms (e.g. [8, 16]), or one can engineer features (such as L^1-norm) on these curves ([17]). Other stable features such as persistent entropy and maximum persistence are also widely used in many applications ([7, 13]). We refer the reader to Pun, Xia and Lee [18] for a survey of different feature engineering techniques on persistence diagrams.

Studies using sliding window persistence have a variety of applications such as action classification [12], wheeze detection [13], chatter detection [19], (quasi)

periodicity quantification of videos [20], chaos detection [21] and financial time series classification [8]. There are also many studies that employ persistent homology of physiological signals for a range of applications, including activity recognition from EMG [16], detecting autism spectrum disorder from EEG [7], optimal delay embedding parameter selection for EEG [22], classification of ECG [9], and respiration rate estimation [23].

1.1 Problem Definition

Stress detection is an active area of study. Researchers apply different methods to obtain better classification accuracies to differentiate physiological signals collected under stressful vs non stressful events. As TDA methods have previously been shown effective in distinguishing chaotic time series from non chaotic ones, we expected to observe high performance when persistent homology is applied to the stress detection domain. Also, the methods needed to be computationally feasible. On the other hand, when windows got larger, persistent homology became more and more computationally expensive. This was a great drawback of TDA methods, and we solved it by using the subwindowing strategy.

2. TIME SERIES METHODS

Whenever a variable changes quantitatively in time, it can be expressed as a *time series*. This broad definition makes time series methods appear in almost every field. The number bacteria in a culture, the price of a stock, the wrist temperature of a person, the population of a city are all regarded as time series if they are measured at different timestamps.

Definition 2.1. A *(univariate) time series* is a function $T \to \mathbb{R}$ where T is the time domain.

In practice, we do not work with continuous functions defined on a finite interval. We usually have finite time-ordered samples of outputs $(x_t \mid t \in I)$ where I is a finite index set taken from T the time interval on which our function is defined. In this work, we will assume that the sample set I consists of elements for which successive measurements are all equal.

In this study, we study classification problems defined on collections of time series sampled from naturally occuring phenomena that come with discrete labels. When a number of time series are given labels, we may want to build a reliable system that classifies them. For instance,

- Music genre classification from audio samples,
- Physical activity (jogging, cycling etc.) recognition from accelerometer signals,
- Classification of home electric appliances from their electric consumption,
- Stress detection from wrist temperature

are all time series classification tasks.

Most machine learning classification algorithms work with finite collections of points embedded in an affine space of a small dimension. As such time series data sets,

possibly with varying lengths, with or without a collection of assigned labels do not seem to be suitable for such algorithms. However, there are methods that can convert a given time series into a finite collection of points embedded in an affine space of a fixed small dimension.

2.1 Sliding Windows

Let $x = (x_i : i = 0, \ldots, N)$ be a univariate time series in $\mathbb{R}$. The sequence of *sliding windows* on x with a window shift of k is a sequence of equally sized multivariate time series (Figure 2.1):

$$((x_{kn}, x_{kn+1}, x_{kn+2}, \ldots, x_{kn+d-1}) : n = 0, 1, \ldots, \lfloor (N-d+1)/k \rfloor). \tag{2.1}$$

Using the sliding windows method, one embedds a collection of time series, with possibly different lengths, into an affine space of a fixed small dimension as a cloud of points. Embedding a large time series into an affine space of a fixed dimension can be very useful for time series classification tasks in several ways. For example, it might not be practical to make classification on very long time series. A smartphone app that listens to and tells us the genre of a song would only be feasible if it could detect the genre in 5-10 seconds, not longer. Another scenario is that if each class contain only a few number of instances, one can augment the data using windowing. Note, however, that when the window shift is smaller than the window size, the sliding windows overlap. In such cases, to prevent data leakage between train and test sets, we should do the train-test split before windowing.

2.2 Delay Embeddings

The topology of a time series data embedded as a cloud of points may not very interesting. In such cases, using the method of *(time) delay embeddings* allows us to convert any time series as a dataset in a higher dimensional euclidean space similar to sliding windowing method we outlined above.

The d-dimensional delay embedding (also known as a *state space reconstruction*) of x with a shift of k is the subset of $\mathbb{R}^d$ is given by (Figure 2.2)

$$\{(x_{kn}, x_{kn+1}, x_{kn+2}, \ldots, x_{kn+d-1}) : n = 0, 1, \ldots, \lfloor (N-d+1)/k \rfloor\}. \tag{2.2}$$

Input: x: univariate time series.
ℓ: window length.
s: window shift.
Output: y: the sequence of sliding windows
Function *getWindows*
Let L be the length of x.
$a \leftarrow 0$
$b \leftarrow \ell - 1$
$i \leftarrow 0$
while $b < L$ **do**
$y_i \leftarrow (x_a, \ldots, x_b)$
$a \leftarrow a + s$
$b \leftarrow b + s$
$i \leftarrow i + 1$
end
return y
end

Figure 2.1 : Windowing algorithm.

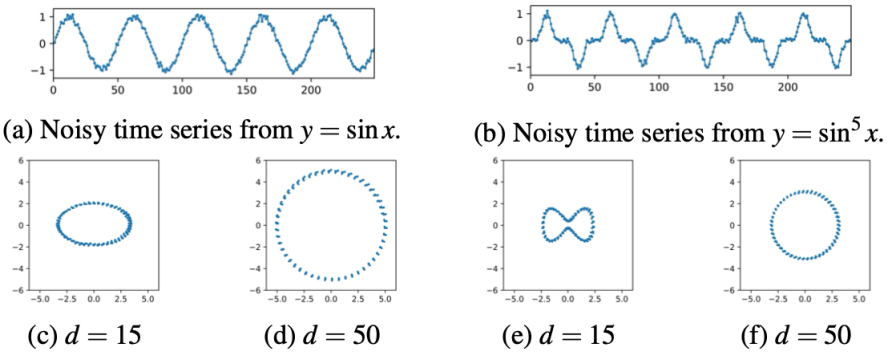
The number d is referred as the dimension of the delay embedding. Observe that the delay embedding is the sequence of sliding windows realized as a dataset where each sliding window becomes a point in the space.

Under certain conditions, Taken's embedding theorem [11] guarantees that delay embeddings captures contains essential information about the topology of the time series. For an arbitrary time series, one need not have an interesting topology. But, for example, if the time series is sampled from a periodic function, the time delay embedding will follow a closed path on $\mathbb{R}^d$. For instance, the time delay embeddings of time series sampled from $\cos x$ on a set of equally spaced points on $\mathbb{R}$, lies on a circle if the embedding dimension resonates with the frequency of $\cos x$ [24]. We refer the reader to Perea (2019) [25] for theoretical justifications and several examples on the subject.

Example. The two time series of length 250 in Figure 2.3 are sampled from five periods of $y = \sin x$ and $y = \sin^5 x$ with additional noise, respectively. The time delay embeddings with $d = 50$ for both time series are both circles with different radii. However, when the embedding dimension is small ($d = 15$), the former is an ellipse whereas the latter is the boundary of an eyeglasses-shaped object. This idea shows us that we can distinguish the two time series for different embedding dimensions using persistent homology.

Input: x: subwindow.
d: embedding dimension.
Output: The delay embedding of x into $\mathbb{R}^d$.
Function *delayEmbedding*
  Let L be the length of x.
  for $i = 0$ **to** $L - d$ **do**
    $y_i \leftarrow (x_i, \ldots, x_{i+d-1})$
  end
  return y
end

Figure 2.2 : Delay embedding of a subwindow.



(a) Noisy time series from $y = \sin x$.

(b) Noisy time series from $y = \sin^5 x$.

(c) $d = 15$ (d) $d = 50$ (e) $d = 15$ (f) $d = 50$

Figure 2.3 : Two noisy time series data along with their time delay embeddings for $d = 15$ and $d = 50$ under PCA visualization.

2.3 The Subwindowing Method

The windowing and the delay embedding methods we outlined above are standard and well-known methods in classification tasks on time series literature. The *subwindowing* method, which is essential to our analyses, is a new method we developed within the context of this thesis. It significantly reduces noise and the required computational power required to analyse any given data set.

Assume that we would like to compute features on a set of sliding windows with a fixed window size and fixed window shift. Instead of directly computing features on each window, we can create sliding *subwindows* on each window (Figure 2.4), then compute the features at the subwindows. Looking at the average and the standard deviation of the subwindow features, one can understand how the window behaves locally, and how this local behavior varies over time.

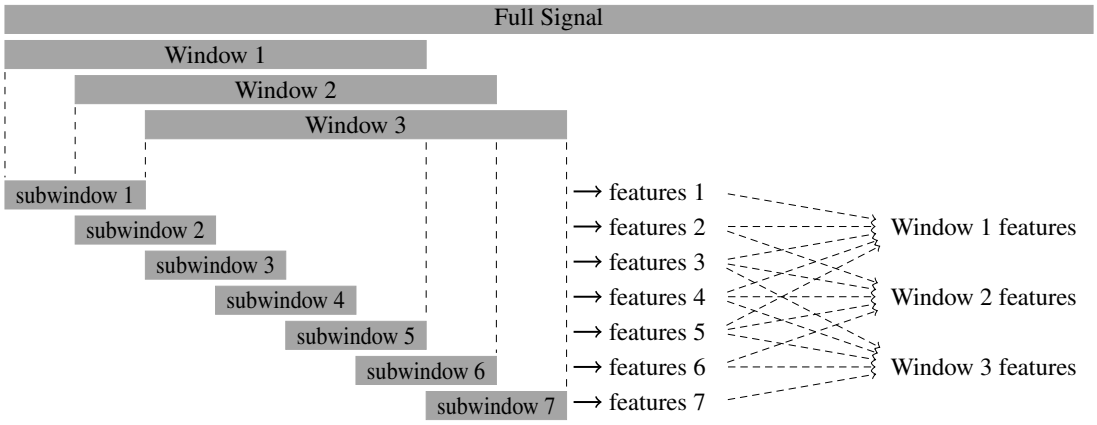


Figure 2.4 : The subwindowing method for feature construction.

This approach is especially useful if we want to compute topological features on each sliding window. In chapter 3, we will introduce several feature engineering methods on time series using Topological Data Analysis (TDA). Such methods are robust to stochastic noise, that is, small changes in the data cause small changes in the feature set. On the other hand, the plain TDA methods produce very unstable features if there are random perturbations in the time series, even for brief periods of time. Subwindowing performs well in situations we want to eliminate such random perturbations of short durations, and want to distinguish topologically stable windows from topologically unstable windows. With subwindowing, the noise is trapped into a few subwindows, and its effects on the feature vector of a window diminish largely by computing the mean and standard deviation of feature vectors of subwindows.

The second major advantage of employing the subwindowing method is a certain guaranteed reduction in computational complexity. Assume that the window features are obtained by finding the mean and standard deviation of the features from subwindows. Assume also that the subwindow shift is the same as the window shift as in Figure 2.4. Now, let Window N and Window $N+1$ be two consecutive windows with subwindows $sw_1, \ldots, sw_M$ and $sw_2, \ldots, sw_{M+1}$, respectively. Let f_i be the feature coming from sw_i. Then, the mean feature for Window N is

$$\mu_N := \frac{1}{M} \sum_{i=1}^{M} f_i \tag{2.3}$$

and the standard deviation of features for Window N is

$$\sigma_N := \sqrt{\frac{1}{M} \sum_{i=1}^{M} (f_i - \mu_N)^2} = \sqrt{\frac{\sum_{i=1}^{M} f_i^2}{M} - \mu_N^2}. \tag{2.4}$$

Observe that

$$\mu_{N+1} = \frac{1}{M}\sum_{i=2}^{M+1} f_i = \left(\frac{1}{M}\sum_{i=1}^{M} f_i\right) + \frac{f_{M+1} - f_1}{M} = \mu_N + \frac{f_{M+1} - f_1}{M}. \tag{2.5}$$

Also,

$$\sigma_{N+1}^2 = \frac{\sum_{i=2}^{M+1} f_i^2}{M} - \mu_{N+1}^2 = \sigma_N^2 + \frac{f_{M+1}^2 - f_1^2}{M} + \mu_N^2 - \mu_{N+1}^2. \tag{2.6}$$

So, μ_{N+1} can be written in terms of μ_N, f_1, f_{M+1} and M. Similarly, σ_{N+1} can be written as a function of $\sigma_N, \mu_N, \mu_{N+1}, f_1, f_{M+1}$ and M (Algorithm 2.5).

Input: x: for a particular feature where x_i is the feature value for i-th subwindow.
M: the number of subwindows in a window
Output: Rolling mean and standard deviation of subwindow features.
Function *windowFeatures*
  Let N be the length of x
  $\mu_0 \leftarrow \frac{1}{M}\sum_{i=1}^{M} x_i$
  $\sigma_0^2 \leftarrow -\mu_0^2 + \frac{1}{M}\sum_{i=1}^{M} x_i^2$
  for *i=1* **to** *N-M* **do**
    $\mu_i \leftarrow \mu_{i-1} + \frac{1}{M}(x_{i+M-1} - x_{i-1})$
    $\sigma_i^2 \leftarrow \sigma_{i-1}^2 + \mu_{i-1}^2 - \mu_i^2 + \frac{1}{M}(x_{i+M-1}^2 - x_{i-1}^2)$
  end
  return (μ, σ)
end

Figure 2.5 : Rolling mean and standard deviation of features.

This means once the mean and standard deviation for a window is computed, these features for the next sliding windows can be computed continuously regardless of the window size. If the window size is large, then computing features directly from the windows can be difficult. Subwindowing allows us to compute the features at the same time regardless of the window size.

3. TOPOLOGICAL DATA ANALYSIS AND PERSISTENT HOMOLOGY

Throughout this dissertation, a *dataset* will refer to a finite set of points in the euclidean space, possibly with duplicates. The aim of Topological Data Analysis (TDA) is to understand the shape of the underlying object that the dataset is sampled from. Sometimes we will refer to the topology of that object by saying "the topology of the dataset" although the actual topology of a dataset, obviously, is a totally disconnected set composed of disjoint collection of points.

Figure 3.1 : A point cloud data lying on a circle.

3.1 Seeing Data at Different Scales

In the first chapter, we have already discussed the importance of understanding the shape of the data. While this is quite easy for a 2-dimensional dataset (as in Figure 3.1), it becomes harder when the dataset is sampled from a high dimensional space. If the dataset is sampled uniformly and densely (with a reasonable amount of noise), we can assume that the $\varepsilon-$*offset* of X given by

$$B(X,\varepsilon) \coloneqq \bigcup_{x\in X} B(x,\varepsilon) \tag{3.1}$$

contains important topological information about the underlying object where $\varepsilon > 0$ and $B(x,\varepsilon)$ is the euclidean open ball with radius ε centered at x. In the Stability Theorem (Theorem 3.32) in Section 3.5, we will see that the topological descriptors of $B(X,\varepsilon)$ and the underlying object are similar whenever X is a “nice” sample.

Unfortunately, there is no easy way to determine which ε value best describes the topology of the dataset. For instance, in Figure 3.2 we draw $B(X, \varepsilon)$ for two different values of ε where X is the dataset given in Figure 3.1.

(a) (b)

Figure 3.2 : $B(X, \varepsilon)$ drawn for different values of ε where X is the data in Figure 3.1.

Observe that Figure 3.2a contains multiple connected components and a number of small holes. On the other hand, Figure 3.2b consists of a single connected component and a large hole. Clearly, the latter captures the topology of the dataset very well.

Homology, put simply, is an algebraic invariant that captures information about the "holes" inside a topological space in different dimensions. For dimension zero, homology classes describe the connected components. In dimensions 1 and 2, it shows cirle-like holes and spherical voids in the object, respectively. When a such a hole exists in $B(X, \varepsilon)$ for a wide range of ε values (e.g. the large hole in Figure 3.2(b)), it represents a topological feature of the underlying object. If, otherwise, a homology class exists only for a short range of ε, we say that it is a topological noise.

Throughout this chapter, we will first introduce simplicial complexes, which are combinatorial objects that will help us compute the homology of $B(X, \varepsilon)$ easily with linear algebra. Then we will define homology of a simplicial complex to actually describe the topology of a simplicial complex. Finally, we will define *persistent homology* to find the ε ranges of the homology classes in a sequence of simplicial complexes indexed by the parameter ε.

3.2 Simplices and Simplicial Complexes

Consider our dataset X as the vertices of a graph, and make any two vertices adjacent whenever they are closer than a fixed threshold ε. Although we could compute the homology classes (e.g. holes) of this graph, unfortunately, it wouldn't approximate to the homology of $B(X, \varepsilon)$. This is because, for example, whenever three vertices

are very close to each other, there still might be a hole in the formed triangle of disks, but not in $B(X,\varepsilon)$. To avoid this, we must fill in such small holes inside such triangles, and their higher dimensional analogues. This will give us the notion of a simplicial complex. Below we present simplicial complexes along with some preliminary definitions.

Definition 3.1. A set of vectors $X = \{x_0, \ldots x_n\}$ is said to be in *general position* if the set $\{x_1 - x_0, x_2 - x_0, \ldots, x_n - x_0\}$ is linearly independent. Such X is also called *affinely independent*.

Definition 3.2. An $n-$*dimensional simplex* (or simply an $n-$*simplex*) is the convex hull of a set of $n+1$ vectors (called the *vertices*) in general position. In other words, the $n-$simplex $[x_0, \ldots, x_n]$ generated by an affinely independent set $\{x_0, \ldots, x_n\}$ is

$$[x_0, \ldots, x_n] := \left\{ c_0 x_0 + \cdots + c_n x_n : \sum_{i=0}^{n} c_i = 1 \text{ and } c_i \geq 0 \ \ \forall i \right\}. \tag{3.2}$$

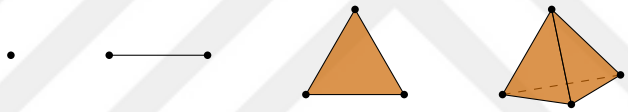


Figure 3.3 : Simplices of dimension 0 through 3.

Note that the ordering of vectors does not affect the resulting simplex. In other words, if $x_0, \ldots, x_n$ are the vertices of a simplex and $y_0, \ldots, y_n$ is any permutation of those vertices, then $[x_0, \ldots, x_n] = [y_0, \ldots, y_n]$.

Definition 3.3. Let Δ_X be a simplex generated by a set of affinely independent vectors X. The simplex Δ_Y generated by a subset $Y \subseteq X$ is called a *face* of Δ_X and we denote the relation by $\Delta_Y \subseteq \Delta_X$. If Y is a proper subset of X, then Δ_Y is called a *proper face* of Δ_X.

When $|Y| = 2$, Δ_Y is called an *edge* of Δ_X. When $|Y| = 3$, Δ_Y is called a *triangle* of Δ_X.

Definition 3.4. A *simplicial complex* is a set of simplices K satisfying

(i) Every face of a simplex of K is in K,

(ii) The intersection of any two simplices of K is a face of both of them.

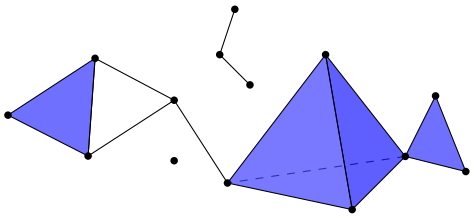


Figure 3.4 : A simplicial complex of dimension 3.

Definition 3.5. The *dimension* $\dim K$ of a simplicial complex K is the maximum dimension of the simplices in it.

Example. Consider the four vectors $x_0 = (0,0)$, $x_1 = (0,1)$, $x_2 = (1,0)$ and $x_3 = (2,0)$ in $\mathbb{R}^2$. The following set K (Figure 3.5) is a simplicial complex:

$$K = \{x_0, x_1, x_2, x_3, [x_0,x_1], [x_0,x_2], [x_1,x_2], [x_0,x_1,x_2], [x_2,x_3]\}. \tag{3.3}$$

If we were to remove any element in K except the last two ones, K would fail to satisfy being a simplicial complex.

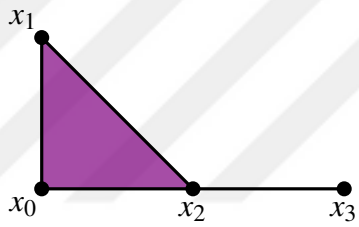


Figure 3.5 : A simplicial complex of dimension 2.

Definition 3.6. A *subcomplex* is a subset of a simplicial complex which itself is also a simplicial complex.

Definition 3.7. Let $n \in \mathbb{N}$ and K be a simplicial complex. The $n-$simplices in K will be denoted by K_n. The $n-$skeleton of K denoted by $K_{\leq n}$ is defined as

$$K_{\leq n} := \bigcup_{i=0}^{n} K_n. \tag{3.4}$$

In other words, $K_{\leq n}$ is the simplicial complex obtained by removing all simplices with dimension greater than n from K.

Observe that the $0-$skeleton of a simplicial complex is its vertex set X, and the $1-$skeleton is a graph. Also, there is a natural ordering between $n-$skeletons of a simplicial complex. That is

$$X = K_{\leq 0} \subseteq K_{\leq 1} \subseteq \cdots \subseteq K_{\leq (\dim K)} = K. \tag{3.5}$$

3.3 Different Types of Simplicial Complexes

Definition 3.8. A *clique* in a graph $G = (V, E)$ is a collection of vertices $C \subseteq V$ such that the subgraph of G on C is the complete graph on C. The *clique complex* $Cl(K)$ of a simplicial complex K is the largest simplicial complex having the same vertex and edge set as K, but for each clique $C = \{c_1, \ldots, c_\ell\}$ in the 1-skeleton, we take a simplex $[c_1, \ldots, c_\ell]$ in $Cl(K)$.

Our simplicial complex definition has a geometric and combinatorial part. A simplex is a geometric object, and a simplicial complex is a collection of the simplices. Naturally, we may want to express it in an purely geometrical or purely combinatorial way. A geometric realization of a simplex would allow us to embed the simplicial complex in a euclidean space, and a combinatorial realization will let us use arbitrary objects as the vertices. This raises the following definitions.

Definition 3.9. A *geometric simplicial complex* obtained from a simplicial complex K is the union (rather than set) of simplices in K.

Definition 3.10. An *abstract simplicial complex* K with vertex set X is a subset of $\mathcal{P}(X)$ satisfying

(i) For any $Y \in K$, any subset $Y' \subseteq Y$ is also in K.

(ii) For any pair of elements $Y, Y' \in K$, $Y \cap Y'$ is also in K.

Throughout this dissertation, we will sometimes refer to a geometric or an abstract simplicial complex as a simplicial complex, whenever it is clear from the context.

Now, our aim is to approximate the topology of the $\varepsilon-$offset $B(X, \varepsilon)$ with a simplicial complex. Below we define the nerve of a cover, and the Nerve Theorem, which is fundamental finding in Algebraic Topology.

Definition 3.11. Let X be a topological space and $\mathscr{U} = \{U_i : i \in I\}$ be a cover for X. The *nerve* $C(\mathscr{U})$ of $\mathscr{U}$ is the simplicial complex having $\mathscr{U}$ as its vertex set, and

$$[U_{i_0}, \ldots, U_{i_k}] \in C(\mathscr{U}) \text{ iff } \bigcap_{j=0}^{k} U_{i_j} \neq \emptyset. \tag{3.6}$$

Figure 3.6 shows the nerve of an open cover. Three sets, U_1, U_2 and U_3 constitute a cover for the given data. For its nerve, there should be three vertices corresponding to those vertices. Also, since the sets intersect pairwise, edges appear between them. However, since the three sets does not intersect, the there is no triangle in its nerve.

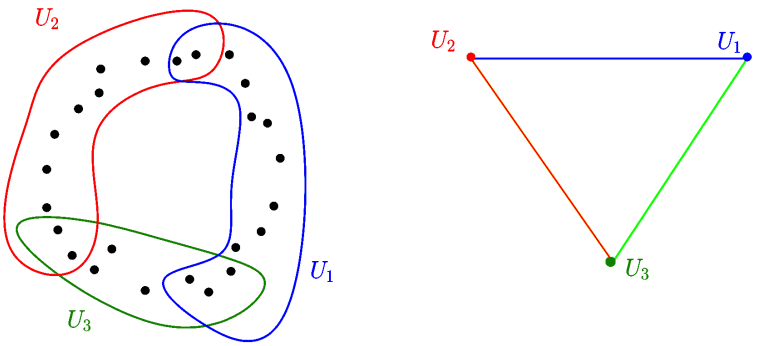


Figure 3.6 : The nerve of a cover of a set of sampled points in the plane [1].

Theorem 3.12 (Nerve Theorem)**.** *[1] Let X be a topological space with an open cover $\mathscr{U} = \{U_i : i \in I\}$. Assume that the intersection of elements of any subset of $\mathscr{U}$ is empty or contractible. Then, $\bigcup_{i\in I} U_i$ and the nerve $C(\mathscr{U})$ are homotopy equivalent.*

There are several formulations of the Nerve Theorem. In a weaker yet relevant form of the theorem, we replace the open cover with a finite cover consisting of closed convex subsets of a euclidean space. When the closed sets are chosen to be the closed $\varepsilon-$balls around each x in the finite set X, the union of the cover is simply the $\varepsilon-$offset $B(X,\varepsilon)$ and the nerve of the cover becomes the *Čech Complex* which will be defined below.

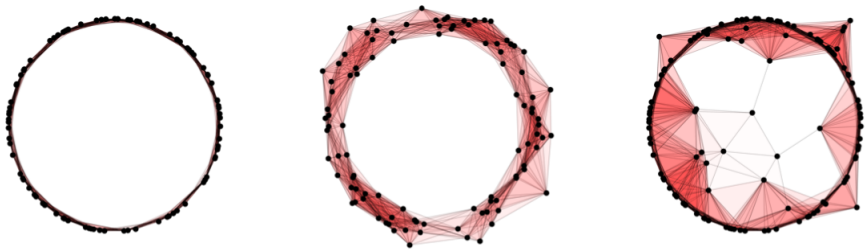
Definition 3.13. Let X be a finite dataset in a metric space and $\varepsilon > 0$. The *Čech Complex* $\text{Cech}(X,\varepsilon)$ is defined as the set of simplices $[x_0,\ldots,x_k]$ such that

$$\bigcap_{i=0}^{k} \bar{B}(x_i,\varepsilon) \neq \emptyset. \tag{3.7}$$

Unfortunately, finding the Čech complex is computationally very expensive. Pairwise intersection of closed balls give the edges, triple intersections give the triangles and so on. If ε is large, and there are a lot of edges, we need to check the existence of many $n-$simplices. To make the computations a lot faster, we might want to stop the algorithm after finding the edges. To this purpose, we introduce the *Rips complex*.

Definition 3.14. Let X be a finite dataset embedded in a metric space $(\mathbb{X},d)$ and $\varepsilon > 0$. The *Vietoris Rips complex* (or shortly the *Rips complex*) $\text{Rips}(X,\varepsilon)$ is defined as the set of simplices $[x_0,\ldots,x_k]$ such that $d(x_i,x_j) \leq \varepsilon$ for all $i,j \in \{0,\ldots,k\}$.

Remark. *The dimensions Čech and Rips complexes can be bigger than the space they are embedded in. For example, if 4 points in $\mathbb{R}^2$ are very close together, we will get a 3-simplex, a tetrahedron.*



(a) Without noise (b) Perturbated (c) Extra random points

Figure 3.7 : Rips complexes for different noise types.

Remark. *Rips complexes are robust to small perturbations in the data. In other words, if each data point is moved in the space by a small amount, the resulting complex will have similar "holes." (Figure 3.7 a&b). On the other hand, when a few extra random points are appended to the dataset, the resulting complex may have a very different shape and different number of holes (Figure 3.7 a&c).*

In $\mathbb{R}^n$, $\text{Rips}(X,2\varepsilon)$ is the clique complex of $\text{Cech}(X,\varepsilon)$, yielding a natural relation $\text{Cech}(X,\varepsilon) \subseteq \text{Rips}(X,2\varepsilon)$. Moreover, we have $\text{Rips}(X,\varepsilon) \subseteq \text{Cech}(X,\varepsilon)$. This is because if m points are within ε distance from each other, all of their ε-balls intersect. The two relations generate the following formula:

$$\text{Rips}(X,\varepsilon) \subseteq \text{Cech}(X,\varepsilon) \subseteq \text{Rips}(X,2\varepsilon). \tag{3.8}$$

This subset relation is not tight. A more precise form by de Silva and Ghrist [26] is given in Theorem 3.15.

Theorem 3.15. [26] *The following holds in $\mathbb{R}^n$:*

$$\text{Cech}\,(X,\varepsilon) \subseteq \text{Rips}(X,2\varepsilon) \subseteq \text{Cech}\left(X,\varepsilon\sqrt{2n/(n+1)}\right). \tag{3.9}$$

Corollary 3.16. *In $\mathbb{R}^n$ we have*

$$\text{Cech}\,(X,\varepsilon) \subseteq \text{Rips}(X,2\varepsilon) \subseteq \text{Cech}\left(X,\sqrt{2}\varepsilon\right). \tag{3.10}$$

There are also other ways to build a simplicial complex on top of a dataset. *Delaunay complex* and *Alpha complex* are two such examples. Furthermore, when the dataset is very large, one can create a *Witness complex* on top of a subsample from the dataset (called the *landmarks*) and determine which simplices to include by looking at the distance from non-landmark (called *witness*) points to landmarks. Since these complexes are out of the scope of this dissertation, we refer the curious reader to [27] for further details.

3.4 Simplicial Homology

In this section, we will introduce the theory of *simplicial homology*. Simplicial homology is defined on an arbitrary field, but in this work, we will only define it for $\mathbb{F}_2 = \{0,1\}$, the field with two elements. The reader may refer to [28] for a detailed theory of the subject, and to [29] for a fast introduction.

Definition 3.17. Let K be a simplicial complex. An $n-$*chain* C of K is defined as the formal sum

$$C \coloneqq \sum_{\sigma \in K_n} c(\sigma) \cdot \sigma \tag{3.11}$$

where $c(\sigma) \in \mathbb{F}_2$ and $c(\sigma) = 0$ for all but finitely many $\sigma \in K_n$. The vector space of $n-$chains of K is denoted by $C_n(K)$.

Definition 3.18. Let σ be an $n-$simplex and $\tau_0, \ldots, \tau_n$ be the $(n-1)-$dimensional faces of σ. The *boundary* $\partial_n(\sigma)$ of σ is defined as the formal sum

$$\partial_n(\sigma) \coloneqq \sum_{i=0}^{n} \tau_i. \tag{3.12}$$

The boundary of an $n-$chain $C = \sum c(\sigma) \cdot \sigma$ is

$$\partial_n(C) \coloneqq \sum c(\sigma) \partial_n(\sigma). \tag{3.13}$$

The boundary of a $0-$simplex is defined to be zero.

Definition 3.19. Let C be an $n-$chain. C is called an $n-$*cycle* if $\partial_n(C) = 0$. Also, C is called an $n-$*boundary* if there exists C^* with $\partial_{n+1}(C^*) = C$.

Example. Consider the simplicial complex K in Figure 3.8. Assume that we want to compute the boundary of the $2-$chain $C = T_1 + T_2 + T_3$. Since $\partial_2(T_1) = e_1 + e_2 + e_3$,

$\partial_2(T_2) = e_3 + e_4 + e_5$ and $\partial_2(T_3) = e_6 + e_7 + e_8$ and since we are working in $\mathbb{F}_2$, we have

$$\begin{aligned}\partial_2(C) &= (e_1 + e_2 + e_3) + (e_3 + e_4 + e_5) + (e_6 + e_7 + e_8) \\ &= e_1 + e_2 + \underbrace{e_3 + e_3}_{=0} + e_4 + e_5 + e_6 + e_7 + e_8. \end{aligned} \tag{3.14}$$

Furthermore,

$$\begin{aligned}\partial_1\big(\partial_2(C)\big) &= (v_1 + v_2) + (v_1 + v_3) + (v_2 + v_4) + (v_3 + v_4) \\ &\quad + (v_4 + v_5) + (v_4 + v_6) + (v_5 + v_6) \\ &= 0 \end{aligned} \tag{3.15}$$

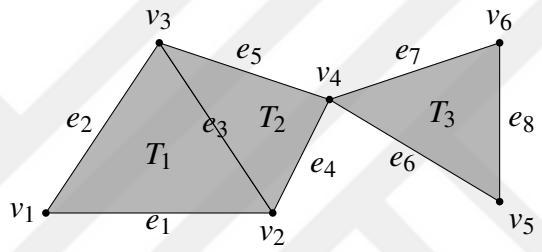


Figure 3.8 : A simplicial complex with 3 triangles.

Theorem 3.20. *For any* $n \in \mathbb{N}$,

$$\partial_n \circ \partial_{n+1} \equiv 0. \tag{3.16}$$

Theorem 3.20 states that the result of the above example is no coincidence. This fundamental property of the boundary map implies that the image of ∂_{n+1} is in the kernel of ∂_n. Then, we are allowed take the quotient of the kernel by the image, raising to the definition of the homology groups.

Definition 3.21. The *nth homology group* $H_n(K)$ of a simplicial complex K is the quotient vector space defined as

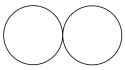
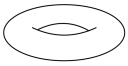
$$H_n(K) \coloneqq \operatorname{Ker}(\partial_n) \Big/ \operatorname{Im}(\partial_{n+1}). \tag{3.17}$$

Definition 3.22. The dimension of $H_n(K)$ is called the *nth Betti number* $\beta_n(K)$ of K and it is given by

$$\beta_n(K) \coloneqq \dim H_n(K) = \dim \operatorname{Ker}(\partial_n) - \dim \operatorname{Im}(\partial_{n+1}). \tag{3.18}$$

Intuitively, 0th, 1st and 2nd Betti numbers $(\beta_0, \beta_1, \beta_2)$ count the number of connected components, holes and voids of the topological object, respectively. In Table 3.1, we present a few topological objects and their Betti numbers.

Table 3.1 : Betti numbers of some topological objects.

	Object	β_0	β_1	β_2	β_{3+}
	Circle	1	1	0	0
	Double circle	1	2	0	0
	Four squares	4	2	0	0
	Surface of sphere	1	0	1	0
	Surface of torus	1	2	1	0

3.5 Persistent Homology

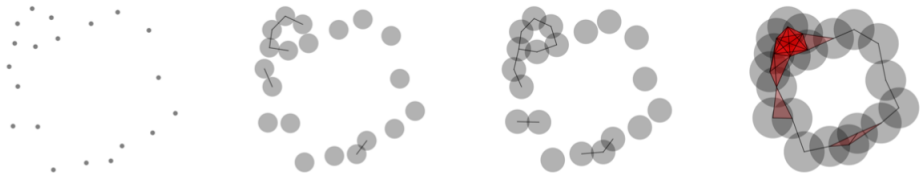
So far, we were able to build a simplicial complex on top of a dataset for a fixed ε, and compute its homology. However, as we have discussed earlier, the choice of a best ε is difficult to find, or perhaps it does not even exist. So, instead, we create the simplicial complexes over a range of ε values, and track how the homologies change as the scale parameter ε changes. Homologies that "persist" for a long interval of ε will be regarded as significant topological features, while those persist for a short range of ε values will be considered as topological noise.

Definition 3.23. A *filtration* $F = \{K_i : i \in \mathbb{I}\}$ is a growing sequence of simplicial complexes. That is, if $i \leq j \in I$, then $K_i \subseteq K_j$.

The ultimate aim of persistent homology is to understand how the topologies of K_i change throughout the filtration. We can create a filtration of Rips complexes $\text{Rips}(X, \varepsilon)$ by considering ε in the range $(0, \infty)$. If X is finite, then the Rips complex changes only at a finite number of ε values.

Example. Let us provide a simple example in $\mathbb{R}^2$ with the ordinary Euclidean metric. Assume that X is a noisy sample from the unit circle $S^1 = \{(x,y) : x^2 + y^2 = 1\}$ (Figure 3.9a). As ε grows, some components of $\text{Rips}(X, \varepsilon)$ merge together, some holes are

"born" and "die". For example, in the example depicted in Figure 3.9 at $\varepsilon = .34$, the Rips complex has one component and one hole, inducing the same topology as $\mathbb{X}$.



(a) $\varepsilon = .0$ (b) $\varepsilon = .14$ (c) $\varepsilon = .18$ (d) $\varepsilon = .34$

Figure 3.9 : The Rips complex $\text{Rips}(X, 2\varepsilon)$ (black and red) of a sample data set X (grey) drawn at different scales. At $\varepsilon = .18$, there are 10 components and one hole. At $\varepsilon = .34$ there is one component and one hole.

Proposition 3.24. *Let $F = \{K_i : i \in \mathbb{I}\}$ be a filtration of simplicial complexes and let $i \leq j \in \mathbb{I}$. For every $n \in \mathbb{N}$, we have a $\mathbb{F}_2$-linear map $\varphi^K_{i\to j} : H_*(K_i) \to H_*(K_j)$ induced by the inclusion $K_i \hookrightarrow K_j$. Furthermore, if $i \leq j \leq m \in \mathbb{I}$, then $\varphi^K_{i\to j} \circ \varphi^K_{j\to m} \equiv \varphi^K_{i\to m}$.*

Definition 3.25. The *persistence module* induced by a filtration $F = \{K_i : i \in \mathbb{I}\}$ of simplicial complexes is a sequence of $\mathbb{F}_2-$linear maps $H_*(K_i) \to H_*(K_j)$ such that $H_*(K_i) \to H_*(K_i)$ is the identity map for all i, and the composition of $H_*(K_{i_0}) \to H_*(K_{i_1})$ and $H_*(K_{i_1}) \to H_*(K_{i_2})$ is the map $H_*(K_{i_0}) \to H_*(K_{i_2})$ for $i_0 \leq i_1 \leq i_2$. If the index set $I = \mathbb{N}$, the persistence module becomes the sequence of maps

$$H_*(K_0) \to H_*(K_1) \to H_*(K_2) \to \dots \tag{3.19}$$

Definition 3.26. An *interval module* is the sequence of maps

$$0 \to \cdots \to 0 \to \mathbb{F}_2 \xrightarrow{\text{id}} \cdots \xrightarrow{\text{id}} \mathbb{F}_2 \to 0 \to \cdots \to 0 \tag{3.20}$$

where the number of leading/trailing zeros can be zero. If the first and last indices that $\mathbb{F}_2$ appears in the interval module are b and $d-1$, respectively, then we denote the interval module by $\mathbb{I}_{b,d}$. If there are no trailing zeroes in the interval module, we denote it by $\mathbb{I}_{b,\infty}$.

To track the homologies on the persistence module, one must decompose the persistence module into interval modules. If a persistence module induced by a filtration has a direct summand of $\mathbb{I}_{b,d}$, then a homology class is born at time b and dies at time d in the filtration. Below we present the *decomposition theorem* which states the conditions under which the persistence module can be decomposed into interval modules.

Theorem 3.27 (Decomposition Theorem [30])**.** *Let M be a persistence module over $\mathbb{I} \subseteq \mathbb{R}$. Then, M can be written as a direct sum of interval modules if $\mathbb{I}$ is finite or each M_i is finite-dimensional.*

Furthermore, whenever we can decompose M into interval modules, the decomposition is unique up to isomorphism and reordering of the interval modules.

3.6 Barcodes and Persistence Diagrams

Definition 3.28. A *persistence diagram* is a multiset (which is a set with possibly duplicated elements) of points in the extended plane $\overline{\mathbb{R}}^2$ which encodes the birth-death pairs of elements in the decomposition. The persistence diagram is assumed to contain the $x = y$ line: the points with zero lifetime.

A *barcode* is another visualization for decomposition of persistence module. In a barcode, a set of intervals are drawn on top of each other where each interval contains the birth-death information of the corresponding interval module.

Theorem 3.27 provides the assumptions under which we can expect the existence of a persistence diagram. Whenever there is a finite filtration or the dataset is finite (both of them are true in real-world scenarios), we can expect that a unique persistence diagram exists.

Now that we have defined persistence diagrams, we need a statistical measurement of the distance on the space of persistence diagrams. Below we present two widely used metrics.

Definition 3.29. Let D_1 and D_2 be two persistence diagrams along with their diagonals and let $\gamma : D_1 \to D_2$ be a bijection. The *Bottleneck distance* W_∞ between D_1 and D_2 [31] is defined as follows:

$$W_\infty(D_1, D_2) = \inf_{\gamma} \sup_{x \in D_1} \|x - \gamma(x)\|_\infty . \tag{3.21}$$

Definition 3.30. Given D_1, D_2 two persistence diagrams (with their diagonals), the *$p-$Wasserstein distance* [32] W_p is defined similar to the Bottleneck distance:

$$W_p(D_1, D_2) := \inf_{\gamma} \left(\sum_{x \in D_1} \|x - \gamma(x)\|_\infty^p \right)^{\frac{1}{p}} . \tag{3.22}$$

Definition 3.31. Let $(\mathbb{X}, d)$ be a metric space and $A, B \subseteq_{\text{closed}} \mathbb{X}$. The *Hausdorff distance* $d_{\mathcal{H}}$ between A and B is defined as

$$d_{\mathcal{H}}(A,B) := \inf_{\varepsilon}\{A \subseteq B(B,\varepsilon) \text{ and } B \subseteq B(A,\varepsilon)\}. \tag{3.23}$$

Theorem 3.32 (Stability [33])**.** *Let $(\mathbb{X}, d)$ be a metric space and $A, B \subseteq_{compact} \mathbb{X}$. Let D_A and D_B be the persistence diagrams obtained from A and B, respectively. Then we have*

$$W_\infty(D_A, D_B) \leq d_{\mathcal{H}}(A,B). \tag{3.24}$$

The Stability Theorem provides theoretical guarantees that if a dataset is sampled densely and uniformly from a topological object (so that the Hausdorff distance between the object and the dataset is small), then the bottleneck distance between their persistence diagrams is small.

3.7 Feature Engineering on Persistence Diagrams

There are several ways to construct features from a persistence diagram. For instance, one can compute the average of birth times of the points in the diagram which do not lie on the diagonal. However, such feature engineering methods are rarely used because of their high sensitivity to noise. This is because, a noise in the dataset can create or remove arbitrarily many extra points with very small yet positive lifetime, hence largely affecting the mean birth time.

In this chapter, we outline several feature engineering methods most of which are robust to noise and largely already exists in the literature. See for example [18] for a survey of such methods.

3.7.1 Via Bottleneck and Wasserstein distances

Both the Bottleneck and $p-$Wasserstein distances rely on finding a best matching between the points from both diagrams including the diagonals (Figure 3.10). This is a computationally expensive process (of the order of $\mathcal{O}(n^2)$), and makes distance based machine learning models (such as knn classification) inconvenient.

On the other hand, computing the Bottleneck or $p-$Wasserstein distance between an arbitrary persistence diagram D and the empty diagram $D_\emptyset$ (called the *amplitude* of

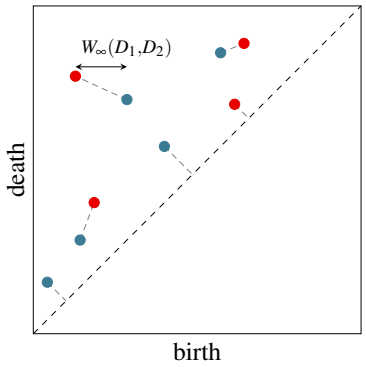


Figure 3.10 : Bottleneck distance between two persistence diagrams.

the diagram D) is considerably cheaper because the best matching between the two diagrams is obvious (Figure 3.11). We compute the distances explicitly:

$$W_\infty(D,D_\emptyset) = \frac{1}{\sqrt{2}} \sup_{x\in D} l_x \tag{3.25}$$

$$W_p(D,D_\emptyset) = \frac{1}{\sqrt{2}} \left(\sum_{x\in D} (l_x)^p \right)^{\frac{1}{p}} \tag{3.26}$$

where l_x is the lifetime of a point $x \in D$.

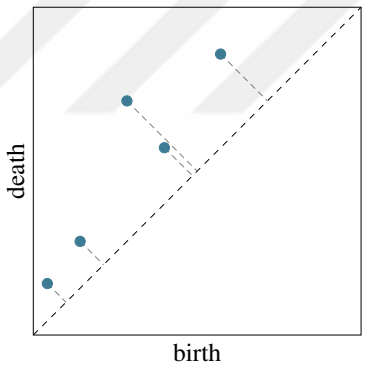


Figure 3.11 : The best matching between a persistence diagram and the empty diagram.

Our features for these metrics are the Wasserstein (for $p = 1$) and bottleneck amplitudes of the diagram.

3.7.2 Via persistent entropy

Perhaps the most widely used feature constructed from a persistence diagram is the *Persistent Entropy* [34]. It is a scale invariant feature, so if you multiply the lifetimes of the points in the diagram by a positive factor, the persistent entropy will not change.

Definition 3.33. The *persistent entropy* $PE(D)$ of a persistence diagram D is given by

$$PE(D) := \sum_{x \in D,\ l_x > 0} -\frac{l_x}{L_D} \ln\left(\frac{l_x}{L_D}\right) \tag{3.27}$$

where $L_D := \sum_{x \in D} l_x$ is the sum of lifetimes.

For our analyses, the persistent entropy of the empty diagram was assumed to be zero.

3.7.3 Via Betti curves and Persistence landscapes

The *Betti curve* is a function of time returning how many points in the persistence diagram are living at that time.

Definition 3.34. Let D be a persistence diagram and $\alpha = (b_\alpha, d_\alpha)$ be a point in the diagram. Consider the following function:

$$f_\alpha(x) := \begin{cases} 1, & b_\alpha \le x \le d_\alpha \\ 0, & \text{otherwise} \end{cases}. \tag{3.28}$$

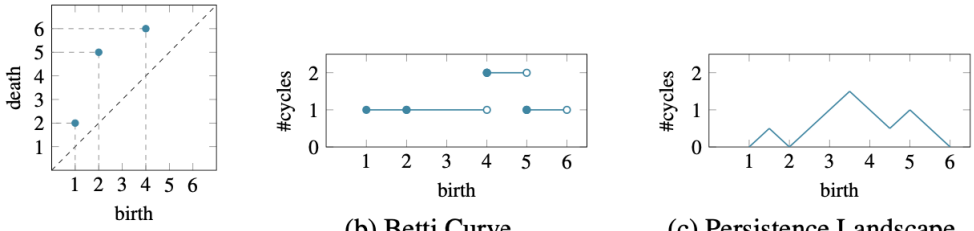
The *Betti Curve* (Figure **??**) obtained from D and f_α as the sum

$$Betti_D(x) := \sum_{\alpha \in D} f_\alpha(x). \tag{3.29}$$

Persistence landscape [35] is another statistical summary function and it is somewhat similar to the Betti curve. However, this time (contrary to f_α above), each point in the diagram creates a "$\wedge$" shaped function between its birth and death. The peak appears at the middle of the birth and death, and its height is proportional to the lifetime of the point. Furthermore, instead of adding them up, we create a sequence of persistence landscapes, ordered by the kth maximum value for any time (Figure **??**). The formal definition is given below.

Definition 3.35. Let D be a persistence diagram. Given $\alpha = (b_\alpha, d_\alpha) \in D$, let

$$g_\alpha(x) := \begin{cases} x - b_\alpha, & \text{if } b_\alpha \le x \le (b_\alpha + d_\alpha)/2 \\ d_\alpha - x, & \text{if } (b_\alpha + d_\alpha)/2 < x \le d_\alpha \\ 0, & \text{otherwise.} \end{cases} \tag{3.30}$$



(a) Persistence Diagram (b) Betti Curve (c) Persistence Landscape

Figure 3.12 : A persistence diagram (a), its Betti Curve (b) and Persistence Landscape (c).

Then, the *kth layer persistence landscape* of D is defined as

$$Landscape_D^k(x) := \underset{\alpha \in D}{\text{kmax}}\, g_\alpha(x) \tag{3.31}$$

where kmax is the kth maximum value among a finite set.

In this study, we use the first layer persistence landscapes, and we will drop the superscript $k = 1$. The four features engineered from the Betti curve and the (first layer) persistence landscape will be their L^1 and L^2 norms.

3.7.4 Alternative approaches

There are several other ways to compute features from a persistence diagram. *Persistence images* and *persistence silhouettes* were also occasionally used in previous studies.

4. THE DATA AND THE EXPERIMENTS

4.1 Dataset Description

In this study, we demonstrate the use of persistent homology for affect and stress recognition. For this purpose, we used one synthetic and two publicly available real datasets. In the synthetic dataset, we aimed to simulate physiological signals from stress and non-stress conditions. The real datasets, WESAD [2] and DriveDB [3] (Available at Physionet [36]) contain physiological signals from participants who were subjected to some stress and non-stress conditions in different environments. The datasets are shortly described below.

4.1.1 Synthetic dataset

The methodology that we used to generate our synthetic datasets is adapted from the methodologies used by Umeda (2017) [16]. In this dataset, we simulated physiological signals using Python's NeuroKit2 library [37] consisting of respiration (RESP) and electrocardiogram (ECG) signals. We generated samples with 120s of non-stress and 120s of stress condition for 20 simulated participants. The signals were generated at 50 Hz. We use the hypothesis that a sustained elevated respiration rate, a sustained elevated heart rate, or an increased heart rate variability are indicators for stress condition.

In the first experiment, we simulated RESP signals with different respiratory rates for the baseline (non-stress) and stress conditions. The respiratory rate for the baseline was set to 15 respirations per minute (rpm), and we used different integer values from 16 rpm to 20 rpm for the stress condition. In the second experiment, the baseline condition contained ECG signals with different heart rates. The baseline heart rate was set to be 70 bpm, and the stress heart rate (HR) was tested for different integer values from 71 bpm to 75 bpm. The standard deviation of the HR parameter was fixed to be 1 for both conditions. The last experiment was similar to the second one except that this time

we manipulated the standard deviation of the HR while keeping the HR constant. The baseline again had an HR of 70 bpm with a standard deviation set to 1, but this time while the stress condition also had an HR of 70 bpm, we used an increasing sequence of standard deviations of $2, 3, 4$, and 5 in our experiments. In order to explore how the results are affected by noise, we repeated our experiments with two different noise parameters 0.1 and 0.3 defined by the NeuroKit2 library.

4.1.2 The WESAD dataset

Physiological recordings from a wrist (Empatica E4) and a chest (RespiBAN) worn device were collected from 15 participants during three different conditions: baseline, stress and amusement (Figure 4.1a). All participants started the experiment with the baseline condition in which they did normal activities such as reading a magazine or sitting at a table approximately for 20 minutes. In the stress condition, participants went under the Trier Social Stress Test (TSST), which included doing a 5-minute presentation to an audience and counting backwards from 2023 with steps of 17. The stress condition lasted about 10 minutes. The amusement condition consisted of watching a sequence of funny video clips, which lasted about 7 minutes. Each of the three conditions was followed by a meditation phase aiming to bring subjects to a neutral state. The order of stress and amusement conditions was counterbalanced across participants.

Acceleration (ACC), RESP, ECG, electrodermal activity (EDA), electromyography (EMG), and temperature (TEMP) signals were collected from the chest-worn device at 700 Hz. The signals from the wrist-worn device were ACC, blood volume pulse (BVP), EDA, and TEMP, with sampling frequencies 32 Hz, 64 Hz, 4 Hz, and 4 Hz, respectively.

4.1.3 The DriveDB dataset

In this study, a total of 17 participants were recorded under three stress levels. The low and high stress conditions correspond to driving on the highway and in the city (Figure 4.1b). The experiment started with resting in the car, then driving on the highway and in the city several times, and finally ending the experiment with the rest condition again. The experiment lasted about 60-90 minutes, depending on participants' driving

speeds. The physiological signals that were recorded during the experiments ECG, EMG, galvanic skin response (GSR) from foot and hand, HR, and RESP. The sampling frequency for all signals was 15.5 Hz. Recordings of only 9 participants were analyzed since the markers that show the transition between different stress conditions were not available or legible for others.

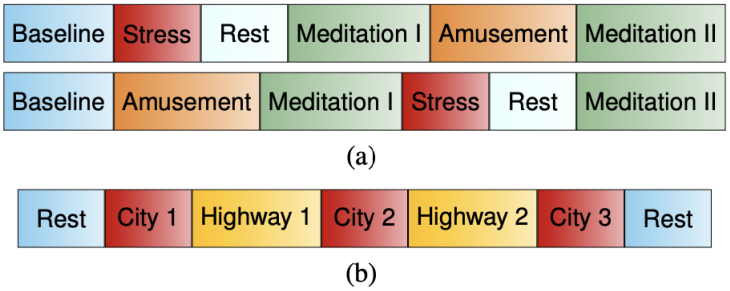


Figure 4.1 : The study protocols for WESAD (a) and DriveDB (b) datasets.

4.2 Experiment

All signals from all datasets were split by stress condition and by participant. To reduce computational time, all signals with a sampling rate higher than 100 Hz (*i.e.* chest signals from WESAD) were downsampled to 100 Hz. Moreover, signals from DriveDB dataset were upsampled from 15.5 Hz to 16 Hz to ensure consistency in our method. Resampling from 700 Hz to 100 Hz was done by selecting every 7th element in the discrete time series. Resampling from 15.5 Hz to 16 Hz was done by upsampling to 496 Hz using linear interpolation, then downsampling by selecting every 31st element. The ACC data in WESAD contained three time series for the accelerations in x, y, and z axes; we averaged them to get a single time series.

4.2.1 Sliding windows and subwindows

Our goal is to create topological features from the windows and use them for affect and stress recognition. Sliding windows with duration of 60 seconds and a shift of 2 seconds were created for all datasets. For our initial analyses, we specifically selected a window size of 60 seconds to make our findings comparable with the original WESAD study. We also varied the window size to test how accuracy is affected. Longer windows produce higher accuracies (Table 4.1). However, in real applications, one might want to keep the time durations short for stress detection.

Table 4.1 : Classification accuracies across different subwindow and window sizes for WESAD.

		Window size				
		10	20	30	60	120
Subwindow size	3	74.38	77.92	79.08	80.67	85.18
	4	75.17	76.63	78.36	81.35	85.01
	5	75.39	77.95	80.07	81.25	84.29

In each window, subwindows with 4s duration and 2s shift were created. This choice of subwindow size was the same regardless of the physiological signal in order to keep the algorithm as simple as possible. To obtain consistent and reliable persistence diagrams, the subwindow size should be large enough to exhibit the periodicity in the delay embedding (Figure 4.4). Our empirical tests showed that the subwindow size of 4 seconds is long enough to capture the local information about the physiological signal including periodicity, yet it is also short enough to make the computation of persistent homology coming from the delay embeddings feasible. In Table 4.1, we show the changes in accuracies across different window and subwindow sizes using an SVM classifier. We give the feature engineering and cross-validation methods that we used to obtain the accuracies in Table 4.1 later in this section. The choice of subwindow shift is more of a computational issue: if the shift drops from 2s to 1s, the number of subwindows, hence the time required for persistent homology computations are doubled.

The subwindowing method was useful in our study for several reasons. First, stressful events usually induce irregular physiological responses. For example, a typical response to stress is high heart rate variability. So, looking at how the subwindows behave across a window is informative for the current task. That is, subwindowing helps us understand the local topology of the window. Secondly, even in non-stress conditions, brief yet powerful noises are common due to participants' coughing, sudden movements, etc. (e.g. Figure 4.2). Third, the delay embedding of a window is a very large dataset (especially if the sampling frequency is high such as 100 Hz), making the computation of persistent homology impractical.

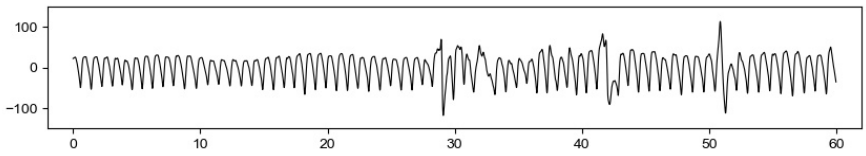


Figure 4.2 : A sample 60-second BVP signal from the baseline condition of WESAD dataset.

4.2.2 Delay embeddings and persistent homology of subwindows

As we noted earlier in Section 2, different embedding dimensions detect different topological information about the time series (Figure 2.3). We used 4 levels of delay embedding dimensions (see Figure 4.4): $.5fs$, fs, $1.5fs$, $2fs$ where fs is the sampling frequency of the particular signal. For example, for a signal with $fs = 100$Hz, the set of embedding dimensions were $\{50, 100, 150, 200\}$.

After converting each subwindow to 4 different point clouds, the persistence diagrams of the induced Rips filtrations were computed for a maximum homology dimension of 1 (Algorithm 4.3). Higher dimensional persistence diagrams were not computed since they require much more computational power, and we wanted to restrict our attention to connected components and one dimensional holes of the delay embeddings, but not to higher dimensional topological features.

Input: A subwindow *sw* with sampling frequency *fs*.
Output: The sequence of persistence diagrams as in Figure 4.4.
Function *getDiagrams*
  $D_1 \leftarrow$ upper level set persistence of *sw*.
  $D_2 \leftarrow$ lower level set persistence of *sw*.
  for i = 1 *to* 4 **do**
    dataset $\leftarrow$ delayEmbedding(*sw*, $i \cdot fs/2$)
    $D_{2i+1} \leftarrow H_0$-barcode of dataset.
    $D_{2i+2} \leftarrow H_1$-barcode of dataset.
  end
  return $(D_1, \ldots, D_{10})$
end

Figure 4.3 : Computation of persistent homology from subwindows.

In addition to the diagrams formed by the delay embeddings, two more persistence diagrams were computed: the 0 dimensional persistence diagrams created by the upper and lower level sets of the subwindows. Note that higher dimensional persistent homology cannot be computed from the subwindows because the subwindows are

univariate. Persistent homology of delay embeddings and level sets were computed using the Ripser [38] and Dionysus-2 libraries [39] of the Python programming language [40], respectively.

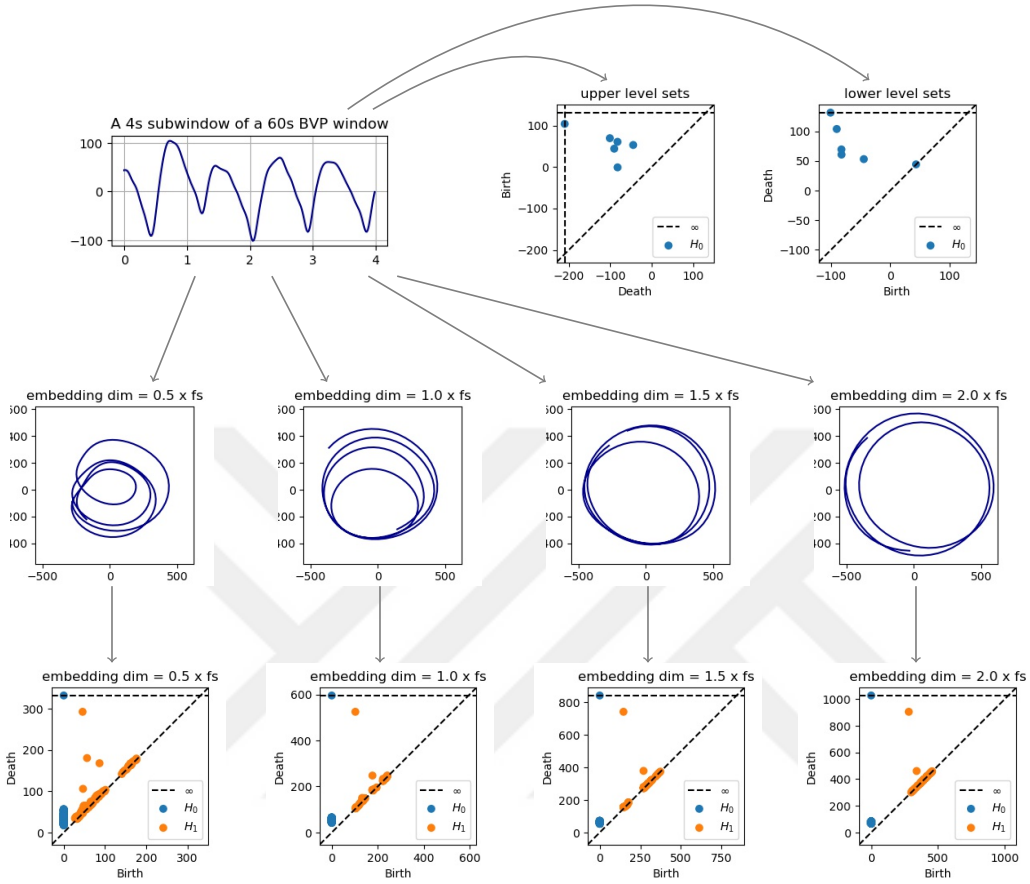


Figure 4.4 : The pipeline for the computation of persistence diagrams from a subwindow.

4.2.3 Feature engineering

Our methods have so far provided 6 persistence diagrams, 4 from delay embeddings and 2 for upper and lower level sets, for each subwindow. Then using the methods we outlined in Section 3.7, a total of 7 features were created from one homology class in a single persistence diagram. Since each subwindow yielded a total of 6 persistence diagrams and 4 diagrams coming from delay embeddings contained two homology classes, the total number of features created using persistent homology was 70.

In order to compute features for the 60s windows, the mean and standard deviation of the features obtained from (4s) subwindows were calculated. This allowed us to know how the signal behaves locally, and how this local behavior varies over a longer period. We also did the same for different window sizes (10, 20, 30, 60, 120, 180, 240,

and 300 seconds), and specifically looked at how the recognition accuracies change accordingly.

The learning algorithms were tested on several subsets of features. For each of the four delay embedding sizes (.5fs, 1fs, 1.5fs, 2fs), features from homology dimension zero (H0) and one (H1) were trained individually and together. Similarly, features coming from upper and lower level sets were trained one by one and together. Then features from all delay embeddings and level sets were combined and used as a full feature set. Before training the algorithms, constant features (*e.g.* full zeros) and features with a correlation higher than .9 were removed. This step was specifically essential since some machine learning algorithms such as Linear Discriminant Analysis are very sensitive to multicollinearity. The correlation heatmaps show that features were moderately correlated within sensors, and weakly correlated or not correlated between sensors (Figure 4.5). Then, features were normalized to the range $[0, 1]$ on both the training and the test set.

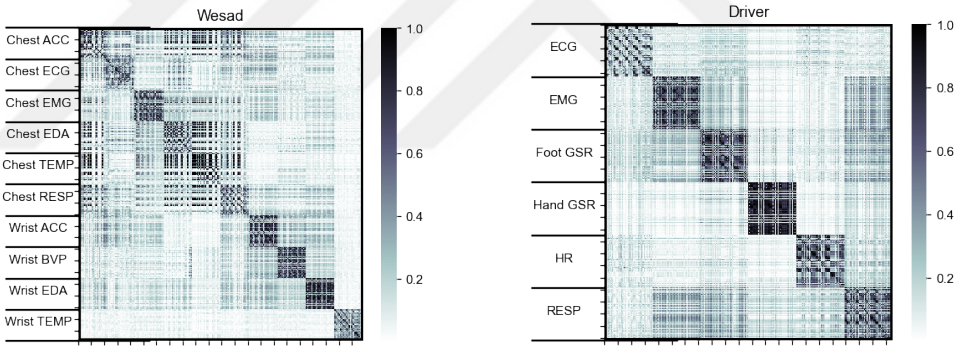


Figure 4.5 : Correlation heatmaps for WESAD and DriveDB.

4.2.4 Learning algorithms

The original WESAD study used five classifiers: Decision tree, Random forest (RF), AdaBoost Decision Tree (AB), Linear discriminant analysis (LDA), and k-nearest neighbor. Three of them (RF, AB, LDA) attained the highest accuracies for some signals. In addition to these three, we used a support vector classifier (SVC).

The tree-based models (RF and AB) were trained on 100 trees with a maximum depth of 5, and the SVC model was trained with a linear kernel and regularization parameter set to 0.1. We used the scikit-learn library [41] implementations of the

learning algorithms. For the reproducibility of our findings, we set the parameter `random_state` to 0 in our stochastic models.

4.2.5 Cross-validation

For all datasets, we used binary (stress vs. non-stress) classification models, but for WESAD and DriveDB datasets, we also used ternary classification models. Leave One Subject Out Cross-Validation (LOSOCV) method was used for all datasets. This method is similar to the k-fold cross validation if we let k be the number of participants, and each fold to be a participant's data. That is, the learning algorithm is trained on all subjects but one, tested on the remaining subject, and then the results are averaged (Figure 4.6). The biggest conceptual advantage of this method is that it helps us know how the model performs on a previously unseen participant. Furthermore, no data leakage between the train and test sets happens even when the sliding windows overlap highly.

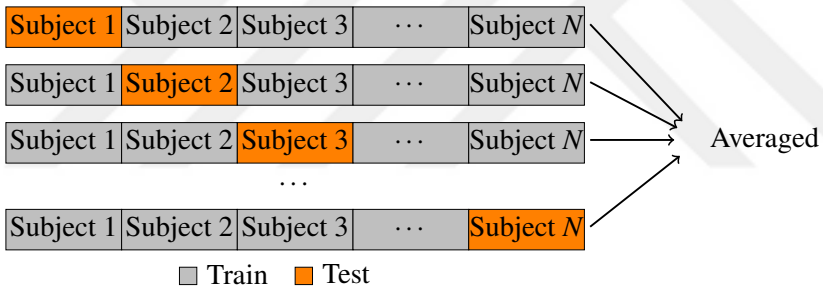


Figure 4.6 : Leave One Subject Out Cross Validation (LOSOCV).

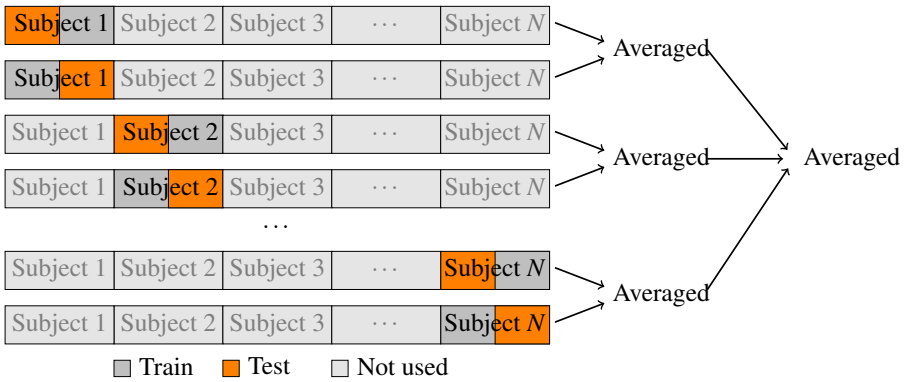


Figure 4.7 : Intra-subject cross-validation.

In order to assess how much of the performance is due to individual differences, we also used an *intra-subject* cross-validation. For this, we consider only data from a single participant, split each condition in half, then use the first halves to predict the

second, and *vice versa* (Figure 4.7). The mean accuracy gives the accuracy for that subject, and averaging over all subjects gives the overall accuracy.

5. RESULTS

5.1 Synthetic Dataset

For the synthetic dataset, we used an SVM classifier on all topological features for every signal. We used 60s windows and LOSOCV method to make our results comparable across datasets.

For the RESP signal, our baseline respiratory rate was 15 respirations per minute (rpm). To simulate stress conditions, we increased the respiratory rates from 16 rpm to 20 rpm with increments of 1 rpm. The success rate of our models in distinguishing baseline from the stress increased as we increased the stress levels measured by an increase in the respiratory rate. Our models were highly successful for 16 rpm. Furthermore, our models worked almost perfectly for 17 rpm and higher even in the presence of high noise (Figure 5.1).

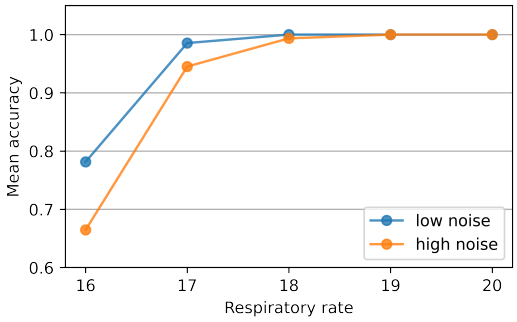


Figure 5.1 : Recognition accuracies as a function of respiratory rate with baseline 15 rpm.

For the ECG signals, the baseline heart rate was 70 bpm. To simulate the stress conditions, we gradually increased the heart rate from 71 bpm to 75 bpm with increments of 1 bpm. Our models distinguished the baseline from all stress levels nearly perfectly, even in the presence of high noise (Figure 5.2).

In our last analyses, we changed the underlying stress indicator. For this part of the study, the variability in the heart rate is assumed to be the main indicator of a stress condition. For this synthetic dataset, the mean heart rate for the baseline was 70

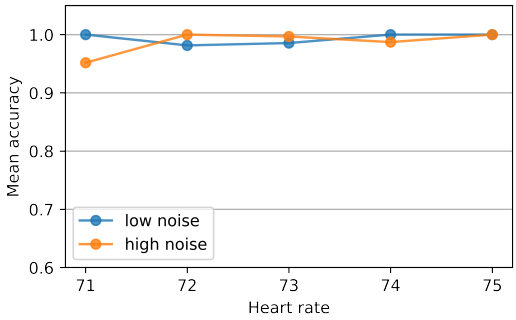


Figure 5.2 : Recognition accuracies as a function of heart rate with baseline 70 bpm.

bpm with a standard deviation of 1. To simulate the stress condition, we increased the standard deviation from 2 to 5 with increments of 1 as we kept the mean heart rate at 70 bpm. While our model was very successful in distinguishing the low stress condition (standard deviation 2), they performed nearly perfectly in high stress conditions (standard deviations 3 to 5) even in the presence of high noise (Figure 5.3).

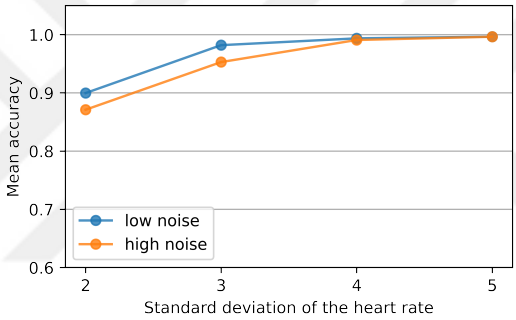


Figure 5.3 : Recognition accuracies as a function of standard deviation of the heart rate with baseline 1.

5.2 WESAD Dataset

Our findings from LOSOCV showed that our automatically created topological features were as effective as the signal-specific features in distinguishing different affect state conditions. We show our results in Table 5.1 and Table 5.2.

For the ternary classification problem (Table 5.1), a support vector classifier on all topological features yielded 81.35% accuracy and with an F_1-score of 73.44%. For almost all signals in WESAD, our automatically created features identified the affective states better than the original study. The most dramatic difference was in the ACC signals. For the chest ACC, our model performed 17% better, while for the wrist ACC, our model performed 11% better. This was followed by ECG, EMG, and wrist EDA where our model performed 10%, 6%, and 10% better than the original study. In almost

all cases, our model performed the best when all topological features were combined as a full feature vector using an SVM classifier.

Table 5.1 : Ternary classification problem accuracies for WESAD.

		Delay Embeddings		Level Sets			
	Clf	Acc	Dgm	Acc	Dgm	All Dgms	Original Findings
Chest							
ACC	SVC	72.34	$2fs$	69.77	Upper	**74.47**	56.56
ECG	SVC	**76.27**	$1fs$	66.11	Upper	72.72	66.29
EMG	SVC	59.00	$1fs$	54.64	Both	**59.67**	53.99
EDA	LDA	66.31	$.5fs, H0$	66.56	Upper	**70.03**	67.07
TEMP	LDA	54.75	$1fs, H1$	53.02	Upper	48.92	**55.68**
RESP	SVC	68.46	$2fs$	70.73	Both	**75.57**	72.37
Wrist							
ACC	RF	68.02	$.5fs$	67.28	Upper	**68.65**	57.20
BVP	SVC	71.64	$1fs$	60.67	Lower	**73.41**	70.17
EDA	RF	69.23	$2fs$	70.50	Both	**72.07**	62.32
TEMP	LDA	55.87	$.5fs$	54.79	Upper	54.93	**58.96**
All chest	SVC	**78.85**	$1fs, H0$	75.34	Upper	77.96	76.50
All wrist	RF	73.93	$2fs$	71.94	Upper	74.72	**75.21**
All	SVC	80.63	$1fs$	80.56	Lower	**81.35**	79.57

We have got a clear separation in the confusion matrix (Figure 5.4). The model performed better in distinguishing stress from non-stress. Most classification errors were between the baseline and amusement conditions.

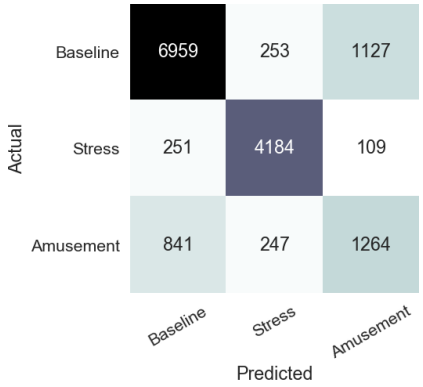


Figure 5.4 : Three-class problem confusion matrix for WESAD.

For the binary classification task (Table 5.2), the highest accuracy and the corresponding F_1-score were 94.46% and 93.26%. For nearly all physiological signals, we obtained higher accuracies with topological features. Again, the ACC signals captured the stress state very well: the accuracy for the chest ACC increased by 14%

and for the wrist ACC by 12% compared to the original study. The improvements for the ECG, EMG, and wrist EDA signals were 3%, 6%, and 5% for the binary task.

Table 5.2 : Binary classification problem accuracies for WESAD.

		Delay Embeddings		Level Sets			
	Clf	Acc	Dgm	Acc	Dgm	All Dgms	Original Findings
Chest							
ACC	RF	**87.69**	$.5fs, H0$	84.82	Both	84.18	73.87
ECG	LDA	**88.70**	$1fs, H0$	81.62	Both	85.23	85.44
EMG	LDA	**73.07**	$1fs$	68.03	Lower	69.75	67.10
EDA	LDA	76.57	$1fs, H1$	81.55	Lower	81.49	**81.70**
TEMP	SVC	**70.19**	$.5fs$	70.19	Both	68.07	69.49
RESP	LDA	81.72	$.5fs$	87.75	Both	**90.10**	88.09
Wrist							
ACC	RF	82.95	$.5fs$	82.81	Both	**83.52**	71.69
BVP	LDA	85.21	$1fs$	72.51	Lower	84.03	**85.83**
EDA	RF	81.19	$2fs$	84.01	Upper	**85.11**	79.71
TEMP	RF	**71.44**	$.5fs$	70.00	Both	70.02	69.24
All chest	SVC	89.32	$1fs, H0$	91.97	Lower	91.79	**92.83**
All wrist	SVC	**88.77**	$2fs$	87.04	Lower	88.55	87.12
All	SVC	92.91	$2fs, H1$	92.95	Both	**94.46**	92.28

We obtained the highest accuracies when features from all persistence diagrams used together in both ternary and binary classification tasks. However, not every persistence diagram contributed equally to the accuracy. For instance, if we used only the upper level set persistence of all physiological signals, we would end up with 61 features (a much smaller set of features than the one used in the original study) yet having reasonably high accuracies (79.61% and 92.47%) for the ternary and binary tasks.

We expected higher accuracies when the window size gets larger. The subwindowing method allowed us to run the same learning algorithms for different sliding window sizes without extra computational cost. For this purpose, we rerun the learning algorithms using several window lengths ranging from 10 to 300 seconds. Our intuition turned out to be true: longer window sizes implied higher accuracy for most signals (Figure 5.5). In particular, the accuracies for 300 second windows were as high as 89.86% and 96.42%, respectively for the ternary and binary tasks.

All of the analyses above were done using LOSOCV. A curious question at this point is to ask how much of these errors stem from individual differences. To answer this

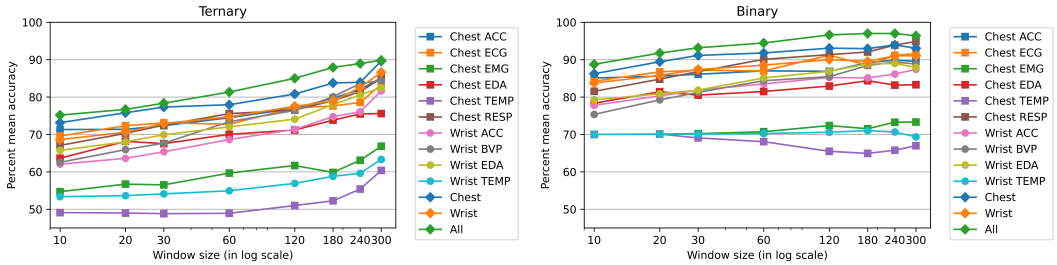


Figure 5.5 : WESAD accuracies for different window sizes.

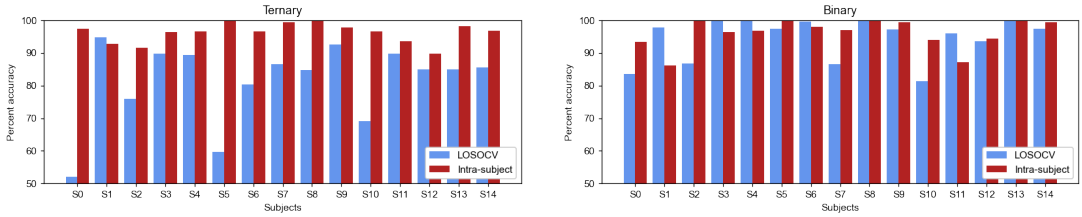


Figure 5.6 : LOSO and intra-subject cross-validation accuracies for WESAD.

question, we used an intra-subject cross-validation. We split the data for each subject in each condition into two subsets. Then we used one half to train and the other half to test the model, and *vice versa*. The sliding window size was again chosen to be 60 seconds. In Figure 5.6, we compare these methods.

The difference was most pronounced in the ternary task. For instance, our intra-subject model performed 45% better than our LOSO model for subject 0 (S0). To test this effect, we ran two repeated measures t-tests. For the ternary classification problem, our intra-subject model performed significantly better than our LOSO model. Our intra-subject model had an average accuracy of 96%, while our LOSO model had 81% with significance $p < .05$. Unfortunately, the accuracies for the binary classification task did not reach a statistical significance of $p < .05$ even though the mean accuracy was higher for the intra-subject rather than the LOSO model. This null finding is probably due to the ceiling effect. These findings indicate that when training and test sets contain data from the same subject, the learning algorithms perform significantly better.

5.3 DriveDB Dataset

The cross-validation scheme of the original DriveDB study is different than ours. They used 300-second non-overlapping windows with a leave-one-out cross-validation scheme where the model is trained on all windows but one, and tested on the

remaining. So, they mix windows from all subjects. Since we wanted to keep our method consistent across datasets, we again followed the same LOSO and intra-subject cross-validation methods we used for WESAD.

The accuracy for our ternary LOSO model was 85.81% with an F_1-score of 79.68% when we used 60s windows (Table 5.3). Our binary LOSO model performed significantly better: 98.07% accuracy with an F_1-score of 97.97% (Table 5.4). A noteworthy observation is that the highest accuracies were obtained from the RESP signal. This indicates that the stress levels of drivers can be accurately measured using only one signal. This greatly reduces the number of features for the learning algorithms.

Table 5.3 : Ternary classification problem accuracies for DriveDB.

		Delay Embeddings		Level Sets		
	Clf	Acc	Dgm	Acc	Dgm	All Dgms
ECG	RF	53.32	$2fs, H1$	**58.49**	Upper	56.68
EMG	RF	66.87	$.5fs$	**69.14**	Upper	66.49
Foot GSR	RF	79.30	$.5fs$	79.60	Both	**80.08**
Hand GSR	RF	**67.26**	$2fs, H0$	65.42	Upper	65.80
HR	SVC	59.30	$1fs, H1$	59.80	Both	**60.61**
RESP	SVC	81.28	$.5fs$	85.71	Both	**85.81**
All	SVC	82.50	$.5fs, H1$	**85.15**	Both	80.59

Table 5.4 : Binary classification problem accuracies for DriveDB.

		Delay Embeddings		Level Sets		
	Clf	Acc	Dgm	Acc	Dgm	All Dgms
ECG	RF	65.74	$2fs$	**71.11**	Upper	67.61
EMG	RF	79.54	$1fs, H0$	82.04	Both	79.91
Foot GSR	RF	90.83	$.5fs$	**92.26**	Upper	91.88
Hand GSR	RF	**82.47**	$1fs, H0$	78.75	Upper	81.44
HR	SVC	73.83	$1fs$	71.06	Both	**74.67**
RESP	RF	93.27	$1fs, H0$	**98.07**	Both	95.54
All	RF	94.96	$.5fs, H1$	**96.24**	Upper	95.39

Similar to the WESAD findings, we again found a separation between stress and non-stress conditions (Figure 5.7). The learning algorithms could easily distinguish *relax* from *city* and *highway* conditions.

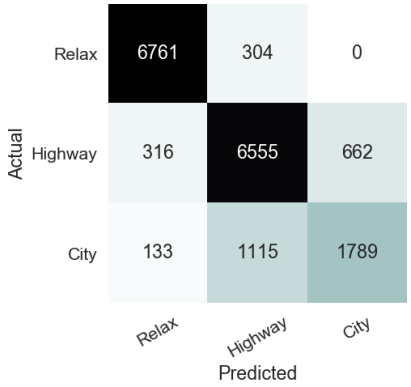


Figure 5.7 : Three-class problem confusion matrix for DriveDB.

Also in this dataset, a greater window size had a positive effect on classification accuracies (Figure 5.8). When the window size is 300 seconds, our models had 91.19% and 96.96% accuracies for the 3- and 2-class tasks, respectively.

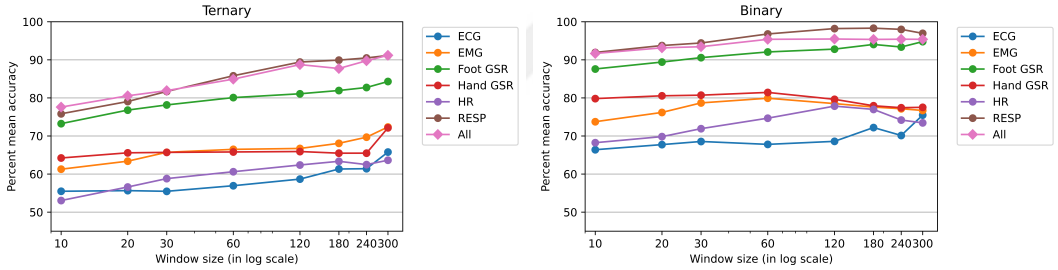


Figure 5.8 : DriveDB accuracies for different window sizes.

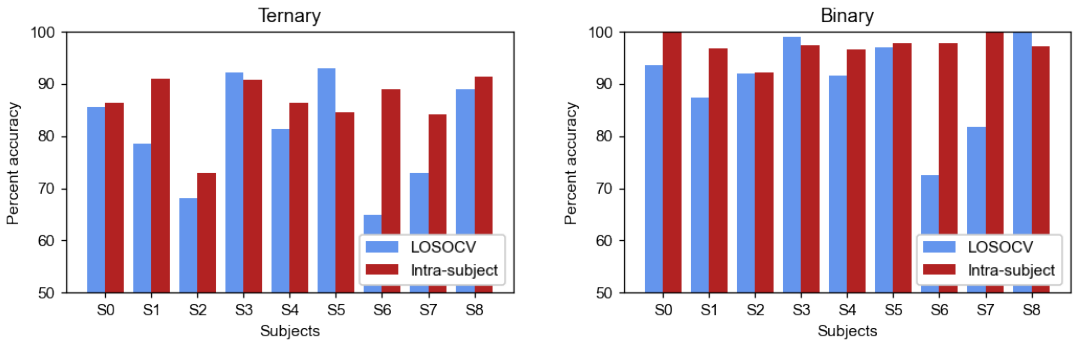


Figure 5.9 : LOSO and intra-subject cross-validation accuracies for DriveDB dataset.

Lastly, we tested if there are any differences in the accuracies of our intra-subject and LOSO models for 60s windows (Figure 5.9). To accomplish this task, we ran two repeated measures $t-$tests for the ternary and binary classification tasks separately. Although the mean accuracies were higher for intra-subject rather than

LOSO cross-validation, the $t-$tests did not reach statistical significance of $p < .05$. We believe that the null findings are a consequence of small sample size and ceiling effect.

6. CONCLUSION

The primary goal of this dissertation was to show the power of persistent homology in time series classification tasks. We tested our topological feature engineering pipeline on a synthetic and two real-world stress detection datasets (WESAD and DriveDB). As common in previous studies, we created sliding windows which constituted the labeled training instances. The subwindowing method we developed (which was previously used in [42] for a non-topological setting) allowed us to inspect how a (sliding) window behaves locally, and how this local behavior varies over time. Also, subwindowing helped us reduce noise and use long windows without incurring extra computational cost. The automatically created topological features did not require domain expertise for stress recognition, and yielded higher accuracies than the accuracies reported in the original studies.

Our feature engineering pipeline was briefly as follows. For each window of duration 60 seconds, we created 4-second subwindows within it. Each subwindow describes the local behavior of the window it belongs to. Then, persistence diagrams were computed from subwindows using upper and lower level sets and time delay embeddings of different embedding dimensions. A number of features were calculated from each persistence diagram such as the bottleneck amplitude or persistent entropy. These features were robust to noise (at least under some loose assumptions) and they were merged together to get a single topological feature vector for each subwindow. To get a single feature vector for a 60-second window, the mean and standard deviation of the subwindow feature vectors was calculated, and merged into a single vector. The subwindowing method helped us realize each 60-second window as the average and standard deviation of the local topological snapshots in it. These feature vectors were then fed into classical machine learning algorithms. Different window sizes were also tested.

In the existing literature, feature engineering methods usually rely on signal-specific features, such as heart rate derived from an ECG signal. The biggest contribution

of this work was to create an automatic topological feature engineering pipeline that is robust to noise and that yields higher accuracies with relatively fewer features. Furthermore, subwindowing allowed us to make continuous stress detection/measurement with little extra computational cost. For any two successive 60-second windows that differ by a subwindow, the feature engineering pipeline only required to compute features from that subwindow. Consequently, feature engineering becomes multiple times faster. In fact, once a feature vector for a window was computed, computing features for the next one did not depend on the window size. For both real datasets, we tested for different window sizes (from 10s to 300s) and observed that longer windows implied better stress detection. Given this result, the advantage of subwindowing method is immediate.

The computational complexity of the standard persistent homology algorithm is $O(n^3)$ (see [29] for slightly lower upper bounds). Thus, whenever we double the size of the dataset, it will take approximately eight times more time to compute the persistence diagram. Since subwindowing creates tiny subwindows from a window, this disadvantage of feature engineering via persistent homology vanishes remarkably.

We used two cross-validation methods, and compared the results. Leave-one-subject-out cross validation scheme is done by training the model on all participants but one, and testing on the other. This method allows us to assess how much a model can generalize to a previously unseen participant. When every participant appears on the test set once and only once, the results are averaged. We also used an intra-subject cross validation. Each participant's data is split into two. We train the model on one half and test on the other, and vice versa. This way, we know how much a participant's own data contains information about their affect and stress state. Once a cross validation score is obtained for every participant, the results are averaged on all participants. The results indicated that using oneself's own labeled data as training notably improved stress recognition performance.

6.1 Limitations and Future Work

We aimed to build an automatic topological feature engineering pipeline that works nicely on time series classification tasks. However, not every time series exhibit a different topological structure for different classes. For instance, vertical and horizontal shifts, or flipping around the y axis create topologically invariant time series. Therefore, effects such as "increased skin temperature" cannot be captured by topological features. Also, even some classes differ topologically, the generated features can be dominated by others if the class difference looks like a topological noise. For example, the topological features of an ECG signal is dominated by "QRS interval" where the largest peak occurs. A difference appearing at "ST interval" can be difficult to classify using topological features. Hence, one has to append signal-specific features to the topological features to obtain state-of-the-art stress detection accuracies.

The two real-world datasets used in this work contained physiological signals measured under stress and non stress conditions. The data were collected using different devices whose sensors were placed on different places of the body. Also, the datasets had different sampling frequencies, and had different levels of preprocessing in their raw form. Thus, it was conceptually problematic to apply features obtained using either dataset on the other. A future study can address this issue by applying the topological features coming from the WESAD study on another dataset again collected via Empatica E4 (the wrist-worn device used in WESAD) to see whether we can transfer the stress-detecting features from one dataset on the other.

The two cross validation methods in our study indicated that using one's own data as training is more effective in distinguishing stress than using other people's labeled data. In practice, a person does not spend time labeling their affect/stress state. So, a future study can create a semi-supervised setting where we have the participant's unlabeled data, and a fitted machine learning model trained on other participants' labeled stress data. Such as study will likely perform better than the results of LOSOCV.

Lastly, we created only a few features from a persistence diagram and used a small number of learning algorithms. A later study can include other vector representations

of a persistence diagram, and use it with more complex machine learning algorithms such as xgboost for improved accuracy.

REFERENCES

[1] **Chazal, F. and Michel, B.** (2021). An introduction to topological data analysis: fundamental and practical aspects for data scientists, *Frontiers in Artificial Intelligence*, *4*.

[2] **Schmidt, P.**, **Reiss, A.**, **Duerichen, R.**, **Marberger, C. and Van Laerhoven, K.** (2018). Introducing wesad, a multimodal dataset for wearable stress and affect detection, *Proceedings of the 20th ACM international conference on multimodal interaction*, pp.400–408.

[3] **Healey, J.A. and Picard, R.W.** (2005). Detecting stress during real-world driving tasks using physiological sensors, *IEEE Transactions on intelligent transportation systems*, *6*(2), 156–166.

[4] **Karan, A. and Kaygun, A.** (2021). Time series classification via topological data analysis, *Expert Systems with Applications*, *183*, 115326.

[5] **Wang, Y.**, **Behroozmand, R.**, **Johnson, L.P. and Fridriksson, J.** (2020). Topology highlights neural deficits of post-stroke aphasia patients, *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, IEEE, pp.754–757.

[6] **Gonzalez-Diaz, R.**, **Paluzo-Hidalgo, E. and Quesada, J.F.** (2019). Towards emotion recognition: a persistent entropy application, *International Workshop on Computational Topology in Image Context*, Springer, pp.96–109.

[7] **Majumder, S.**, **Apicella, F.**, **Muratori, F. and Das, K.** (2020). Detecting Autism Spectrum Disorder Using Topological Data Analysis, *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp.1210–1214.

[8] **Majumdar, S. and Laha, A.K.** (2020). Clustering and classification of time series using topological data analysis with applications to finance, *Expert Systems with Applications*, *162*, 113868.

[9] **Ignacio, P.S.**, **Dunstan, C.**, **Escobar, E.**, **Trujillo, L. and Uminsky, D.** (2019). Classification of single-lead electrocardiograms: TDA informed machine learning, *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, IEEE, pp.1241–1246.

[10] **Chung, Y.M.**, **Hu, C.S.**, **Lo, Y.L. and Wu, H.T.** (2021). A persistent homology approach to heart rate variability analysis with an application to sleep-wake classification, *Frontiers in physiology*, *12*, 202.

[11] **Takens, F.**, (1981). Detecting strange attractors in turbulence, *Dynamical systems and turbulence, Warwick 1980*, Springer, pp.366–381.

[12] **Dirafzoon, A.**, **Lokare, N. and Lobaton, E.** (2016). Action classification from motion capture data using topological data analysis, *2016 IEEE global conference on signal and information processing (globalSIP)*, IEEE, pp.1260–1264.

[13] **Emrani, S.**, **Gentimis, T. and Krim, H.** (2014). Persistent homology of delay embeddings and its application to wheeze detection, *IEEE Signal Processing Letters*, *21*(4), 459–463.

[14] **Marchese, A. and Maroulas, V.** (2018). Signal classification with a point process distance on the space of persistence diagrams, *Advances in Data Analysis and Classification*, *12*(3), 657–682.

[15] **Seversky, L.M.**, **Davis, S. and Berger, M.** (2016). On time-series topological data analysis: New data and opportunities, *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp.59–67.

[16] **Umeda, Y.** (2017). Time series classification via topological data analysis, *Information and Media Technologies*, *12*, 228–239.

[17] **Wang, Y.**, **Ombao, H. and Chung, M.K.** (2019). Statistical persistent homology of brain signals, *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp.1125–1129.

[18] **Pun, C.S.**, **Xia, K. and Lee, S.X.** (2018). Persistent-Homology-based Machine Learning and its Applications–A Survey, *arXiv preprint arXiv:1811.00252*.

[19] **Khasawneh, F.A. and Munch, E.** (2016). Chatter detection in turning using persistent homology, *Mechanical Systems and Signal Processing*, *70*, 527–541.

[20] **Tralie, C.J. and Perea, J.A.** (2018). (Quasi) Periodicity quantification in video data, using topology, *SIAM Journal on Imaging Sciences*, *11*(2), 1049–1077.

[21] **Tempelman, J.R. and Khasawneh, F.A.** (2020). A look into chaos detection through topological data analysis, *Physica D: Nonlinear Phenomena*, *406*, 132446.

[22] **Altındiş, F.**, **Yılmaz, B.**, **Borisenok, S. and İçöz, K.** (2021). Parameter investigation of topological data analysis for EEG signals, *Biomedical Signal Processing and Control*, *63*, 102196.

[23] **Erden, F. and Cetin, A.E.** (2017). Period estimation of an almost periodic signal using persistent homology with application to respiratory rate measurement, *IEEE Signal Processing Letters*, *24*(7), 958–962.

[24] **Perea, J.A. and Harer, J.** (2015). Sliding windows and persistence: An application of topological methods to signal analysis, *Foundations of Computational Mathematics*, *15*(3), 799–838.

[25] **Perea, J.A.** (2019). Topological time series analysis, *Notices of the American Mathematical Society*, *66*(5), 686–694.

[26] **De Silva, V. and Ghrist, R.** (2007). Coverage in sensor networks via persistent homology, *Algebraic & Geometric Topology*, *7*(1), 339–358.

[27] **Zomorodian, A.J.** *et al.* (2012). *Advances in Applied and Computational Topology: American Mathematical Society Short Course on Computational Topology, January 4-5, 2011, New Orleans, Louisiana*, volume 70, American Mathematical Soc.

[28] **Hatcher, A.** (2000). *Algebraic topology*, Cambridge Univ. Press, Cambridge, `https://cds.cern.ch/record/478079`.

[29] **Otter, N.**, **Porter, M.A.**, **Tillmann, U.**, **Grindrod, P. and Harrington, H.A.** (2017). A roadmap for the computation of persistent homology, *EPJ Data Science*, *6*, 1–38.

[30] **Chazal, F.**, **De Silva, V.**, **Glisse, M. and Oudot, S.** (2016). *The structure and stability of persistence modules*, Springer.

[31] **Cohen-Steiner, D.**, **Edelsbrunner, H. and Harer, J.** (2007). Stability of persistence diagrams, *Discrete & computational geometry*, *37*(1), 103–120.

[32] **Cohen-Steiner, D.**, **Edelsbrunner, H.**, **Harer, J. and Mileyko, Y.** (2010). Lipschitz functions have L p-stable persistence, *Foundations of computational mathematics*, *10*(2), 127–139.

[33] **Chazal, F.**, **Cohen-Steiner, D.**, **Guibas, L.J.**, **Mémoli, F. and Oudot, S.Y.** (2009). Gromov-Hausdorff stable signatures for shapes using persistence, *Computer Graphics Forum*, volume 28, Wiley Online Library, pp.1393–1403.

[34] **Atienza, N.**, **Escudero, L.M.**, **Jimenez, M.J. and Soriano-Trigueros, M.** (2019). Persistent entropy: a scale-invariant topological statistic for analyzing cell arrangements, *arXiv preprint arXiv:1902.06467*.

[35] **Bubenik, P.** (2015). Statistical topological data analysis using persistence landscapes., *J. Mach. Learn. Res.*, *16*(1), 77–102.

[36] **Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., ...Stanley, H.E.** (2000). PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals, *circulation*, *101*(23), e215–e220.

[37] **Makowski, D., Pham, T., Lau, Z.J., Brammer J.C., Lespinasse, F., Pham, H., ...Chen, S.H.A.** (2021). NeuroKit2: A Python toolbox for neurophysiological signal processing, *Behavior Research Methods*, `https://doi.org/10.3758/s13428-020-01516-y`.

[38] **Tralie, C.**, **Saul, N. and Bar-On, R.** (2018). Ripser. py: A lean persistent homology library for python, *Journal of Open Source Software*, *3*(29), 925.

[39] **Mozorov, D.**, (2014), Welcome to Dionysus 2 documentation!, `https://mrzv.org/software/dionysus2/`.

[40] **Van Rossum, G. and Drake Jr, F.L.** (1995). *Python reference manual*, Centrum voor Wiskunde en Informatica Amsterdam.

[41] **Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E.** (2011). Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, *12*, 2825–2830.

[42] **Hönig, F.**, **Batliner, A. and Nöth, E.** (2007). Fast recursive data-driven multi-resolution feature extraction for physiological signal classification, *3rd Russian-Bavarian Conference on Bio-medical Engineering*, pp.47–52.

CURRICULUM VITAE

Name Surname: Alperen Karan

Work Experience

- *Data Scientist*, Getir, 09.2021 - Present.
- *Research Assistant*, Istanbul Technical University, 12.2013 - 09.2021.

Education

- **M.A.** Bogazici University, Psychology, 2019.
- **M.S.** Bogazici University, Mathematics, 2015.
- **B.S.** Bogazici University, Mathematics, 2013.

Publications

- **Karan, A.**, & Kaygun, A. (2021). Time Series Classification via Topological Data Analysis. *Expert Systems with Applications, 115326.*
- **Karan, A.**, & Mungan, E. (2018). In Further Search of Tonal Grounds in Short Term Memory of Melodies. In R. Parncutt & A. Schiavio (Ed.), *Proceedings of the Fifteenth International Conference on Music Perception and Cognition* (p. 237-243), Karl-Franzens Universitaet Graz.
- Gillam, W. D., & **Karan, A.** (2017). The Hausdorff topology as a moduli space. *Topology and its Applications, 232*, 102-111.