

Time Series Classification via Topological Data Analysis

Alperen Karan^{a,*}, Atabey Kaygun^a

^a*Department of Mathematical Engineering, Istanbul Technical University, 34467, Istanbul, Turkey*

Abstract

In this paper, we develop topological data analysis methods for classification tasks on univariate time series. As an application, we perform binary and ternary classification tasks on two public datasets that consist of physiological signals collected under stress and non-stress conditions. We accomplish our goal by using persistent homology to engineer stable topological features after we use a time delay embedding of the signals and perform a subwindowing instead of using windows of fixed length. The combination of methods we use can be applied to any univariate time series and allows us to reduce noise and use long window sizes without incurring an extra computational cost. We then use machine learning models on the features we algorithmically engineered to obtain higher accuracies with fewer features.

Keywords: Persistent homology, Time delay embedding, Machine learning, Stress recognition.

1. Introduction

In this study, we use persistent homology to perform classification tasks on two publicly available multivariate time series datasets (Schmidt et al., 2018; Healey & Picard, 2005) that include physiological data collected during stressful and non-stressful tasks. To make our analyses more thorough, we also tested our methods on a synthetic time series dataset. Instead of directly computing signal-specific features from *sliding windows* and *subwindows* on modalities such as electrocardiogram and wrist temperature (Figure 5), we extracted features using *persistence diagrams* and their statistical properties. Subwindowing method allowed us to reduce noise without incurring an extra computational cost. We then developed machine learning models and then assessed the performance of our models by varying window sizes and using different flavors of persistence diagrams.

*Corresponding author

Email addresses: karana@itu.edu.tr. (Alperen Karan), kaygun@itu.edu.tr (Atabey Kaygun)

Topological Data Analysis (TDA) techniques usually work with points embedded in an affine space of large enough dimension. However, TDA techniques can still be applied to time series data sets, whether they are univariate or multivariate. One can convert a univariate time series into a finite collection of points in a d -dimensional affine space using *delay embedding* methods, of which one can compute persistent homology. Since Taken’s Theorem implies that the delay embeddings produce topologically invariant subsets on a non-chaotical dynamical system (Takens, 1981), one can reasonably expect that persistent homology produces features that would distinguish different time series.

There is a handful of research for time series classification using topological data analysis. The reader may refer to Perea (2019) for theoretical background and real-world applications of sliding window persistence. A typical strategy for such classification tasks is to create equally sized sliding windows from the time series, then to find the corresponding delay embeddings and compute the persistent homology. If the windows are long, then the resulting dataset becomes large, and thus, persistent homology computation becomes prohibitive. In such cases, subsampling can be used to reduce the dataset size (e.g., Dirafzoon et al., 2016; Emrani et al., 2014), or the time series can be divided into non-overlapping subseries which are then converted to delay embeddings and to multiple persistence diagrams (Majumdar & Laha, 2020). Persistent homology computation can also be made on the level set filtration rather than delay embeddings (Majumder et al., 2020) or from both (Chung et al., 2021).

Once a persistence diagram for each window is obtained, distance-based machine learning models (such as kNN) can be employed (e.g., Seversky et al., 2016; Marchese & Maroulas, 2018). However, since finding distance between persistence diagrams is computationally expensive, such machine learning models can only be used if the number of windows 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.

There are studies that extract features from statistical properties of the diagrams, such as the mean and standard deviations of lifetimes (Ignacio et al., 2019; Chung et al., 2021), but such features are sensitive to noise and should be used very cautiously. One can also map a persistence diagram to a function such as a Betti curve or a Persistence Landscape. As these functions lie in a vector space, they can directly be fed into learning algorithms (e.g., Umeda, 2017; Majumdar & Laha, 2020), or one can engineer features (such as L^1 norm) on these curves (Wang et al., 2019). More stable features such as persistent entropy and maximum persistence are also widely used in many applications (Majumder et al., 2020; Emrani et al., 2014). We refer the reader to Pun et al. (2018) for a survey of different feature engineering techniques using persistent homology.

Studies using sliding window persistence have a variety of applications such as action classification (Dirafzoon et al., 2016), wheeze detection (Emrani et al., 2014), chatter detection (Khasawneh & Munch, 2016), (quasi)periodicity quantification of videos (Tralie & Perea, 2018), chaos detection (Tempelman & Kha-

sawneh, 2020) and financial time series classification (Majumdar & Laha, 2020). There are also many studies that employ persistent homology of physiological signals for a range of applications, including activity classification from EMG (Umeda, 2017), detecting autism spectrum disorder from EEG (Majumder et al., 2020), optimal delay embedding parameter selection for EEG (Altındaş et al., 2021), classification of ECG (Ignacio et al., 2019), and respiration rate estimation (Erden & Cetin, 2017).

In our study, we start with using sliding windows of a fixed size since this is going to allow us to compare our findings with the original studies (Schmidt et al., 2018; Healey & Picard, 2005). Moreover, within each window, we run another sliding window (which we will call a *subwindow*) with a much shorter length, yet long enough to contain at least one cycle of periodic-like signals. Thus the subwindows capture the local information about the window they were taken from, and one can measure statistically how these local features vary over the window. We specifically created different delay embeddings with different embedding sizes, and then constructed topologically stable features from the resulting persistence diagrams. Finally, we trained and tested machine learning models on these newly engineered features.

1.1. Plan of the article

The rest of this paper is set up as follows. In Section 2, we start by reviewing persistent homology, metrics and kernels on the space of persistence diagrams. In Section 3, we show the feature engineering methods on persistence diagrams using the metrics and kernels defined. In Section 4, we investigate sliding windows and time delay embeddings of univariate time series, and we demonstrate how feature engineering on sliding subwindows is independent of any window size. In Section 5, we discuss the datasets we use in this study. We explain our methodology in detail in Section 6. We then present our results and their analysis in Section 7 and Section 8. Finally, in Section 9, we provide the pseudocodes for the algorithms used in this study.

2. Topological Data Analysis and Persistent Homology

Most of the material we present in this section is used to set up the background and the notation, and can be found in most books and articles on TDA. We refer the reader to Edelsbrunner & Harer (2010) for a detailed introduction on the subject.

We work within the metric space \mathbb{R}^n with a fixed distance function d and a fixed dimension $n \geq 1$. The reader might assume d is the euclidean metric even though the arguments we provide work equally well with any other metric. We use a finite sample of points X from an unknown region Ω in \mathbb{R}^n . Our ultimate aim is to gain some insight into topological/homological invariants of the region Ω using only X .

2.1. Simplices

Given a finite set of points X in \mathbb{R}^n , the simplex $S(X)$ spanned by X is the convex hull spanned by X . The points in X are called *the vertices* of the simplex $S(X)$. Any subset $Y \subseteq X$ spans a subcomplex $S(Y)$ called *faces* of the complex $S(X)$. The dimension of the face, and therefore the simplicial complex itself, is determined by the number of points.

2.2. Simplicial complexes

An m -dimensional *simplicial complex* $C_* = (C_0, \dots, C_m)$ in \mathbb{R}^n is a finite graded collection of simplices where simplices with a fixed dimension- i is denoted by C_i . We also ask that given any two simplices S and S' in C_* , the intersection $S \cap S'$ is a subsimplex of both S and S' . The dimension of a simplicial complex C_* is the maximal dimension among all of the simplicial contained in C_* .

2.3. Homology of a complex

Given a simplicial complex $C_* = C_1, \dots, C_m$ in \mathbb{R}^n , we form a \mathbb{R} -vector space $\text{Span}(C_i)$ spanned by i -dimensional simplices of C_* together with operators $d_i: \text{Span}(C_i) \rightarrow \text{Span}(C_{i-1})$ defined as

$$d_i[\mathbf{x}_1, \dots, \mathbf{x}_i] = \sum_{j=1}^i (-1)^j [\mathbf{x}_1, \dots, \widehat{\mathbf{x}}_j, \dots, \mathbf{x}_i] \quad (1)$$

where $\widehat{\mathbf{x}}_j$ indicates that specific vertex is missing. Now, $d_{i-1}d_i = 0$ for every $i > 1$, and therefore, $\ker(d_i) \supseteq \text{im}(d_{i+1})$. Then we define the i -th homology $H_i(C_*)$ as the quotient subspace $\ker(d_i)/\text{im}(d_{i+1})$ for every $i \geq 0$. For each $i \geq 0$, the number $\dim_{\mathbb{R}} H_i(C_*)$ is called the i -th *Betti number of the complex* C_* .

2.4. Rips complexes

One of the most commonly used simplicial complexes obtained from a finite sample of points is the *Rips complex*. Given X and $\varepsilon > 0$, a finite set of points $U \subset X$ forms a simplex if every pair of points in $\mathbf{x}, \mathbf{y} \in U$ are at-most ε -apart. The resulting simplicial complex is denoted as $\text{Rips}(X, \varepsilon)$. For $\varepsilon = 0$, the Rips complex consists only of the sampled vertices X , which is our the data set, and for a large ε , the Rips complex becomes a single high dimensional simplex.

2.5. Filtrations

A filtered complex is a collection of complexes F_ε such that $F_{\varepsilon_1} \subseteq F_{\varepsilon_2}$ whenever $\varepsilon_1 \leq \varepsilon_2$. The Rips complex we defined in Subsection 2.4 is an example. One can compute the homology of a Rips complex at a particular ε_0 . However, a choice of ε_0 at which the Rips complex has the homology type of Ω may not necessarily exist unless $X \subseteq \Omega$ is ε -dense for some ε^1 . So, we need to look at the

¹A subset $X \subseteq \Omega$ is called ε -dense if for every $\mathbf{x} \in \Omega$ there is a point $\mathbf{y} \in X$ such that $d(\mathbf{x}, \mathbf{y}) < \varepsilon$. See (Björner, 1995) and (Otter et al., 2017, page 10).

“persistent” features of the homology of the data at different scales of ε in order to see which homology classes live for a small range of ε , and which homology classes persist longer.

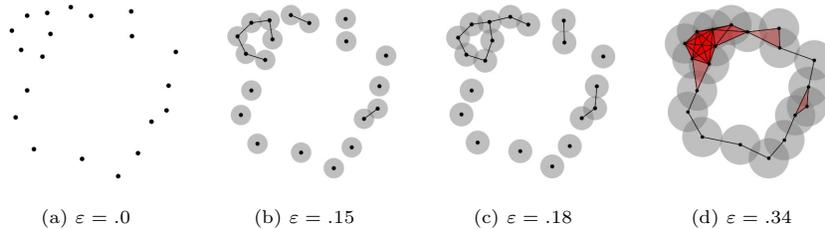


Figure 1: The Rips complex $\text{Rips}(X, 2\varepsilon)$ (black and red) of a sample dataset X (grey) drawn at different scales. At $\varepsilon = .18$, there are 10 components and one hole. At $\varepsilon = .34$, there is one component and one hole.

2.6. Persistent homology

Consider a finite filtration $(K_\varepsilon)_{\varepsilon \in I}$ of simplicial complexes. By definition, $\varepsilon_i \leq \varepsilon_j$ implies $K_{\varepsilon_i} \subseteq K_{\varepsilon_j}$. This relation yields a linear map between the homology groups $H_n(K_{\varepsilon_i}) \rightarrow H_n(K_{\varepsilon_j})$ for arbitrary n (Chazal et al., 2012). Moreover, there is an interval $[b, d]$ where b is the minimum index where a $H_n(K_{\varepsilon_b})$ is non-zero, and d is the minimum index after which $H_n(K_{\varepsilon_\ell})$ are all 0. The numbers b and d are called *the birth time* and *the death time* of the corresponding homological features.

The multiset of pairs (b_i, d_i) of birth and death times is called the *persistence diagram* (Edelsbrunner et al., 2000). A persistence diagram is assumed to contain infinitely many copies of the *diagonal* which includes the points with simultaneous birth and death. This allows us to create bijections between two persistence diagrams.

2.7. Wasserstein and Bottleneck distances

Let D_1, D_2 be persistence diagrams along with their diagonals, and let $\gamma : D_1 \rightarrow D_2$ be a bijection. The *bottleneck distance* W_∞ (Cohen-Steiner et al., 2007) between D_1 and D_2 is defined as follows (see Figure 2):

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

The p -Wasserstein distance (Cohen-Steiner et al., 2010) is defined similarly:

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

The Bottleneck distance is stable under small perturbations of the data, and the p -Wasserstein distance is stable under certain assumptions.

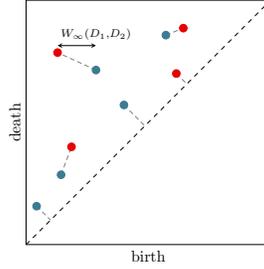


Figure 2: The bottleneck distance between two persistence diagrams.

2.8. Persistence landscapes and Betti curves

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

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

The *Betti Curve* (Figure 3b) obtained from D and f_α as the sum

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

Persistence landscape, introduced by Bubenik (Bubenik, 2015), is another statistical summary function that can be obtained from a persistence diagram D (Figure 3c). Given $\alpha \in D$, let

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

Then, the k^{th} layer persistence landscape of D is defined as

$$Landscape_D^k(x) := \text{kmax}_{\alpha \in D} g_\alpha(x)$$

where kmax is the k^{th} maximum value among a finite set. In this study, we use the first layer persistence landscapes, and we will drop the superscript $k = 1$.

3. Feature engineering on persistence diagrams

There are several ways to construct features from a persistence diagram, such as calculating the mean birth times of points in the diagram. However, these features are not stable in the sense that a slight perturbation in the dataset can create or remove several points with short lifetimes.

In this study, we engineered features based on topological data analysis methods using distances on persistence diagrams, persistent entropy, and Betti curves. We outline our method in this section. We also provide the pseudocodes for our feature engineering methods in Algorithm 4.

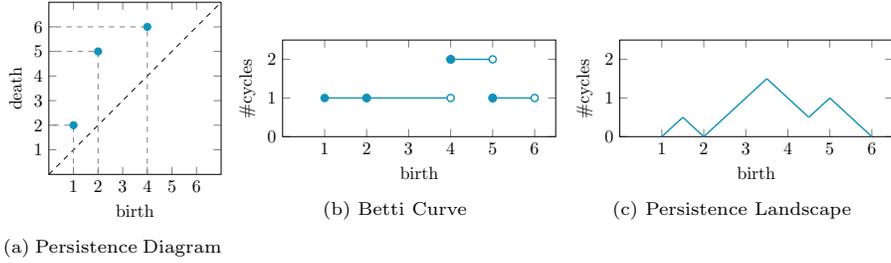


Figure 3: A persistence diagram (a), its Betti Curve (b) and Persistence Landscape (c).

3.1. Via Bottleneck and Wasserstein distances

Let D_\emptyset be the diagram containing only the diagonal. For any diagram D , we can compute the Wasserstein distance for $p = 1$: $W_1(D, D_\emptyset)$ and the bottleneck distance $W_\infty(D, D_\emptyset)$ as two features of a diagram. Let l_x be the lifetime of a point $x \in D$. We can write them down explicitly since the perfect matching between any diagram D with D_\emptyset is obvious:

$$W_1(D, D_\emptyset) = \frac{1}{\sqrt{2}} \sum_{x \in D} (l_x) \quad W_\infty(D, D_\emptyset) = \frac{1}{\sqrt{2}} \sup_{x \in D} (l_x)$$

3.2. Via persistent entropy

Persistent Entropy is another summary statistic that can be derived from a persistence diagram (Atienza et al., 2019). A key feature of persistent entropy is that it is scale-invariant. The persistent entropy $PE(D)$ of a persistence diagram D is given by

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

where $L_D := \sum_{x \in D} l_x$ is the sum of lifetimes. Whenever the diagram contained only the diagonal, the persistent entropy was assumed to be zero.

3.3. Via norms

Finally, we consider the Betti curve and first layer persistence landscape of a diagram D as real-valued functions, and then compute their L^1 and L^2 norms. These are going to constitute our four new features constructed from these kernels.

4. Sliding Windows and Delay Embeddings

4.1. Converting univariate series into multivariate series

Let $x = (x_i : i = 0, \dots, N)$ 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 (Algorithm 1):

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

Creating equally sized shorter windows from a large time series can be very useful for machine learning tasks. 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, train-test splits should be carefully made.

4.2. Delay embeddings

The method of *delay embeddings* allows us to embed a collection of time series of varying lengths into a euclidean space of a fixed dimension. Under some assumptions, the topology of the delay embedding contains essential information about the nature of the time series. We refer the reader to Perea (2019) for theoretical justifications and several examples on the subject. 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 (Algorithm 2)

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

The number d is referred as the dimension or the size of the delay embedding.

One should note that if y is a periodic signal, the time delay embedding on y will follow a closed path on \mathbb{R}^d . For example, the time delay embeddings of $y = \{\cos x : x \in \mathbb{T}\}$, where \mathbb{T} is a set of equally spaced points on \mathbb{R} , lies on a circle if the embedding dimension resonates with the frequency of $\cos x$ (Perea & Harer, 2015).

4.2.1. An example

The two time series of length 250 in Figure 4 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.

4.3. The subwindowing method

The subwindowing approach we developed in this article is essential to our analyses since it reduces noise and the required computational power. 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 5), 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.

This approach is especially useful if we want to compute topological features on windows using persistent homology on delay embeddings. If a window contains a noisy part (as most physiological signals do), the corresponding delay embedding and the resulting persistence diagrams will be noisy as well. With

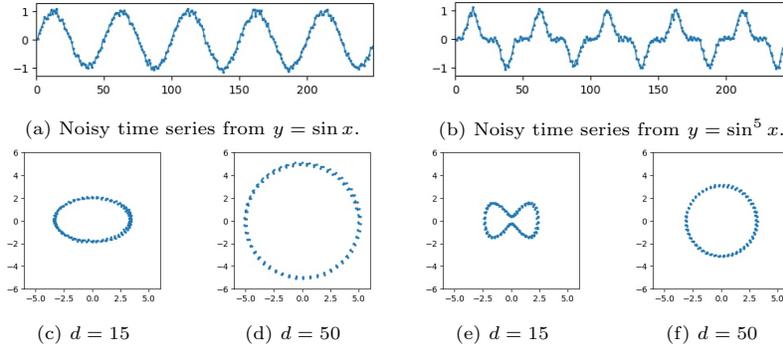


Figure 4: Two noisy time series data along with their time delay embeddings for $d = 15$ and $d = 50$ under PCA visualization.

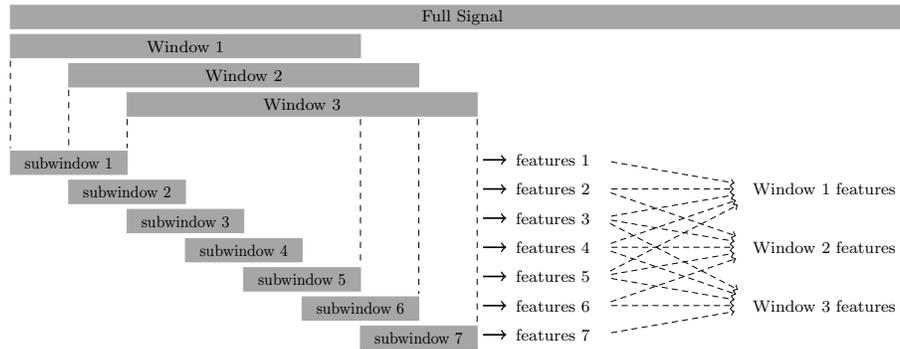


Figure 5: The subwindowing method for feature construction

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.

Moreover, the subwindowing method has advantages for computation. 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 5. Now, let Window N and Window $N + 1$ be two consecutive windows with subwindows sw_1, \dots, sw_M and sw_2, \dots, sw_{M+1} , respectively. Let f_i be the feature coming from sw_i . Then, the mean feature for Window N is

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

and the standard deviation of features for Window N is

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

Observe that

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

Also,

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

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 5).

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 sampling frequency of the signal is high, and 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.

5. 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 (Schmidt et al., 2018) and DriveDB² (Healey & Picard, 2005) contain physiological signals from participants who were subjected to some stress and non-stress conditions in different environments. The datasets are shortly described below.

5.1. Synthetic dataset

The methodology that we used to generate our synthetic datasets is adapted from the methodologies used by Umeda (2017). In this dataset, we simulated physiological signals using Python’s NeuroKit2 library (Makowski et al., 2021) 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.

²Available at <https://physionet.org/content/drivedb/1.0.0/> (Goldberger et al., 2000)

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.

5.2. 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 6a). 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.

5.3. 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 6b). 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.

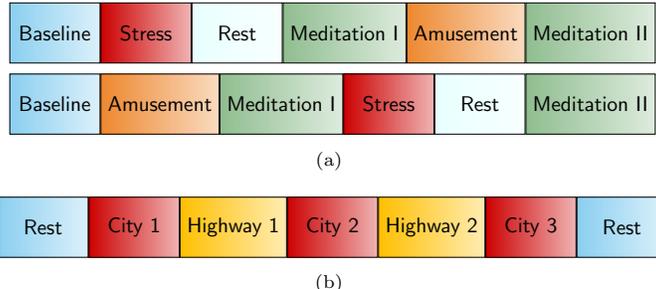


Figure 6: The study protocols for WESAD (a) and DriveDB (b) datasets.

6. Methodology

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.

6.1. 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 1). However, in real applications, one might want to keep the time durations short for stress detection.

		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

Table 1: Classification accuracies across different subwindow and window sizes for WESAD.

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 8). 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 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 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 7). 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.

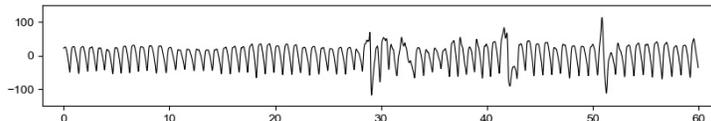


Figure 7: A sample 60-second BVP signal from the baseline condition of WESAD dataset.

6.2. Delay embeddings and persistent homology of subwindows

As we noted earlier in Section 4, different embedding dimensions detect different topological information about the time series (Figure 4). We used 4 levels of delay embedding dimensions (see Figure 8): $.5fs$, fs , $1.5fs$, $2fs$ where fs is the sampling frequency of the particular signal. For example, for a signal with $fs = 100\text{Hz}$, 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 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.

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 (Tralie et al., 2018) and Dionysus-2 libraries (Morozov, 2018) of the Python programming language (Van Rossum & Drake Jr, 1995), respectively.

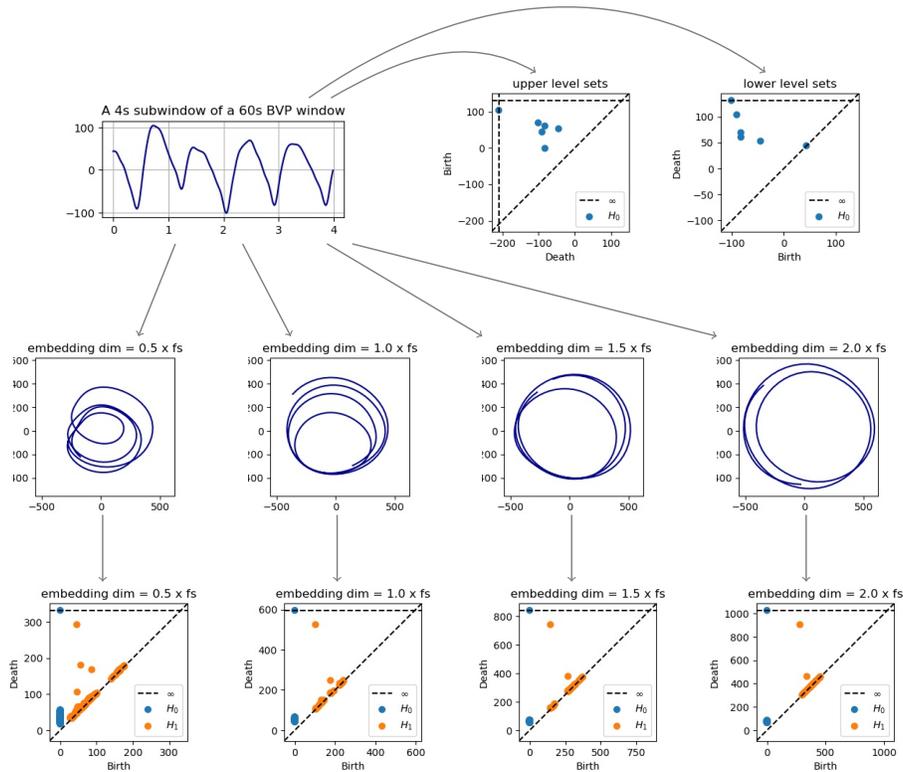


Figure 8: The pipeline for the computation of persistence diagrams from a subwindow.

6.3. 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, 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 9). Then, features were normalized to the range $[0, 1]$ on both the training and the test set.

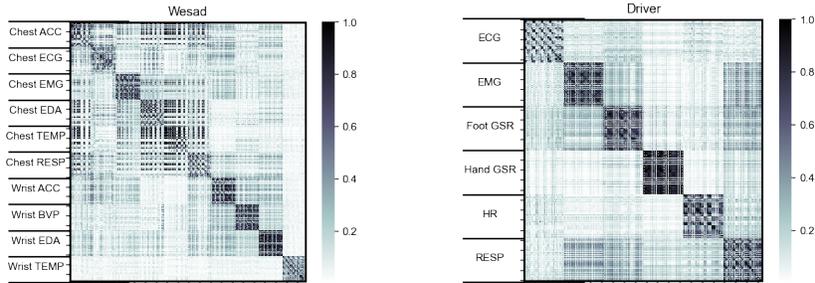


Figure 9: Correlation heatmaps for WESAD and DriveDB.

6.4. 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 (Pedregosa et al., 2011) implementations of the learning algorithms. For the reproducibility of our findings, we set the parameter `random_state` to 0 in our stochastic models.

6.5. 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 (LOSO CV) 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 10). 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.

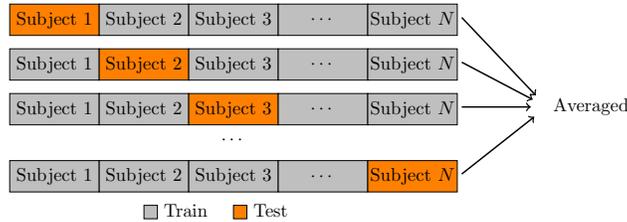


Figure 10: Leave One Subject Out Cross Validation (LOSO CV).

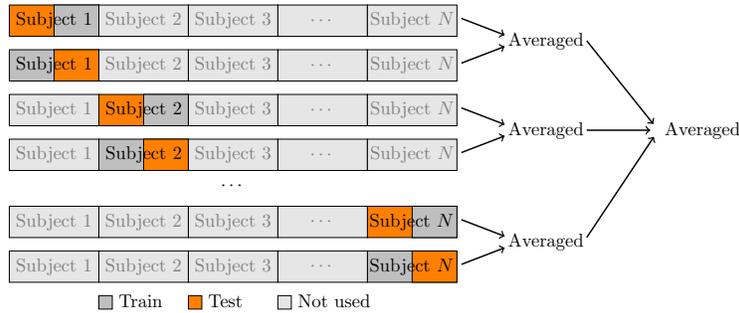


Figure 11: Intra-subject cross-validation.

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 11). The mean accuracy gives the accuracy for that subject, and averaging over all subjects gives the overall accuracy.

7. Results

7.1. 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 12).

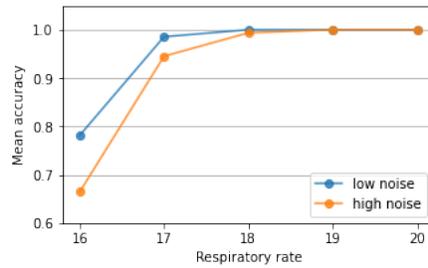


Figure 12: Recognition accuracies as a function of respiratory rate with baseline 15 rpm.

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 13).

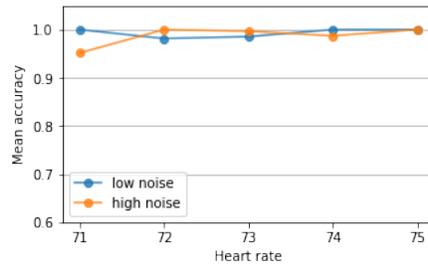


Figure 13: Recognition accuracies as a function of heart rate with baseline 70 bpm.

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 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 14).

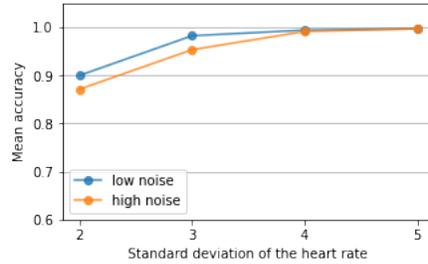


Figure 14: Recognition accuracies as a function of standard deviation of the heart rate with baseline 1.

7.2. 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 2 and Table 3.

For the ternary classification problem (Table 2), 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.

	Clf	Delay Embeddings		Level Sets		All Dgms	Original Findings
		Acc	Dgm	Acc	Dgm		
<i>Chest</i>							
ACC	SVC	72.34	2fs	69.77	Upper	74.47	56.56
EKG	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
<i>Wrist</i>							
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

Table 2: Ternary classification problem accuracies for WESAD.

We have got a clear separation in the confusion matrix (Figure 15). The model performed better in distinguishing stress from non-stress. Most classification errors were between the baseline and amusement conditions.

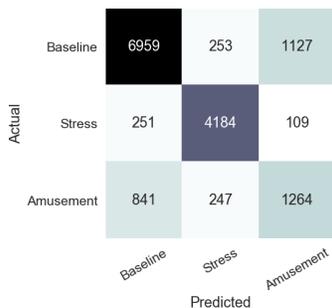


Figure 15: Three-class problem confusion matrix for WESAD.

For the binary classification task (Table 3), 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.

		Delay Embeddings		Level Sets		All Dgms	Original Findings
	Clf	Acc	Dgm	Acc	Dgm		
<i>Chest</i>							
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
<i>Wrist</i>							
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

Table 3: Binary classification problem accuracies for WESAD.

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 sub-windowing 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 16). In particular, the accuracies for 300 second windows were as high as 89.86% and 96.42%, respectively for the ternary and binary tasks.

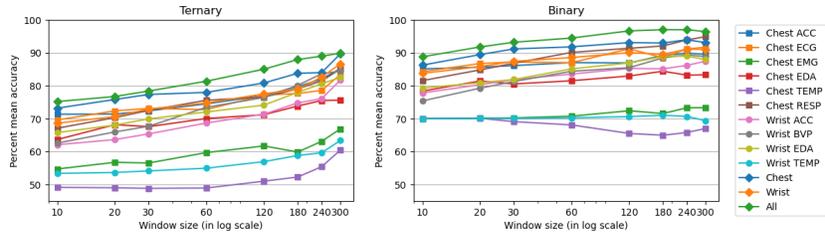


Figure 16: WESAD accuracies for different window sizes.

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 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 17, we compare these methods.

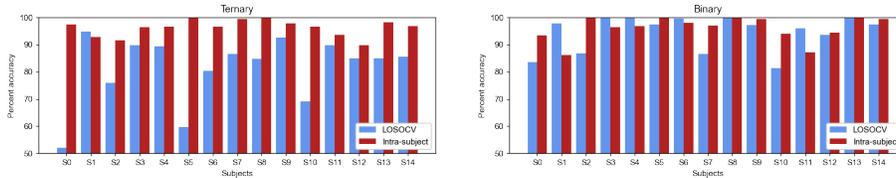


Figure 17: LOSO and intra-subject cross-validation accuracies for WESAD.

The difference was most pronounced in the ternary task. For instance, our intra-subject model performed 45% better than our LOSO model. 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.

7.3. 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 4). Our binary LOSO model performed significantly better: 98.07% accuracy with an F_1 -score of 97.97% (Table 5). 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.

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

Table 4: Ternary classification problem accuracies for DriveDB.

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

Table 5: Binary classification problem accuracies for DriveDB.

Similar to the WESAD findings, we again found a separation between stress and non-stress conditions (Figure 18). The learning algorithms could easily distinguish *relax* from *city* and *highway* conditions.

8. Conclusion

The main aim of this study was to demonstrate the power of Topological Data Analysis (TDA) techniques in classification of time series. Specifically, we used persistence diagrams and their statistical properties to distinguish physiological signals collected under stress and non-stress conditions. As common in previous studies, this was done by creating sliding windows of a fixed duration, computing features, and training and testing machine learning models on these engineered features. The subwindowing approach we developed allowed us to inspect how the signal behaves locally, and how this local behavior varies over longer periods. Then, using TDA methods, we were able to create persistence diagrams from subwindows, create features on persistence diagrams and apply machine learning algorithms.

A proper feature engineering of a signal usually requires field knowledge. For instance, heart rate variability can be derived as a feature from an ECG signal. Our findings showed that most of the automatically generated topological features are at least as effective as signal-specific features in affect recognition. Under our models, the topological features we generated from the acceleration signals produced significantly better results than the baseline WESAD findings. Combined with the respiration signals, the accuracies of our models improved even further. Given that nearly all smartwatches and smartbands already have built-in accelerometer sensors, one can easily see that our method can readily, widely and cheaply be used in other studies and applications.

To test the effectiveness of our methods thoroughly, we also used a synthetic dataset in which elevated heart rate, elevated respiration rate, and variability in the heart rate were taken to be indicators of stress. On the synthetic dataset, our methods worked nearly perfectly in distinguishing stress from the baseline even in the presence of high noise. This tells us that the lower accuracies we obtained from the real datasets might be due to the limitations of our hypothesis on stress indicators we used for our synthetic dataset. It is possible that instead of a *sustained* elevation, random sporadic changes in the heart and respiratory rate might be an accurate indicator of stress. Also, it is possible that the stress conditions in the experiments (e.g. driving in the city) may not cause observable stress in the subjects. Therefore, it is unrealistic to expect to see high accuracies from machine learning models on the real datasets we used in this study.

The features in this study came from four different delay embeddings and level sets, which allowed us to make comparisons. The highest accuracies were obtained when we used features from all of the diagrams. Although this forces us to use a large number of features, it was possible to use features from only one delay embedding or level sets to get much fewer features, and obtain slightly less accuracy.

We have seen that longer windows usually implied higher recognition accuracies. However, this especially becomes computationally problematic when features are directly computed from the whole window. So, we implemented the subwindowing strategy, which was previously used (Hönig et al., 2007) on a different dataset with different feature engineering methods. Using this method,

one computes features on the subwindows, then finds the window features by taking averages and standard deviations. We observed that once the mean and standard deviation of a window is known, and when an observation is replaced with a new one within the window, there is a relatively cheaper computing strategy for the new mean and standard deviation regardless of the window size. Using this observation, we could use longer windows to compute new features from sliding windows and their subwindows to improve accuracy without incurring heavy computational costs.

Our findings also indicate that when the learning algorithm is trained and tested on disjoint data subsets coming from the same subject, the accuracies are higher than the LOSOCV approach. Hence, a device monitoring stress from a person may start with a relatively lower recognition accuracy, but it is going to improve its discriminative power when worn by the same person for long enough to update the parameters of the learning algorithm. In the original DriveDB study, the authors used 300 second non-overlapping windows with leave one out cross-validation (using both intra- and inter-subjects data) and reached approximately 97% accuracy. When we trained our model on 300 second windows with LOSOCV, we reached higher than 91% accuracy with fewer features. Given that we only included 9 subjects (compared to 24 in the original study) due to missing data and we used LOSOCV, our findings are promising.

8.1. Future work

This work used a single subwindow size for every stress condition in the signal. For future works, one can manipulate the subwindow size of a periodic signal and make sure that the delay embedding contains one and only one closed loop. Also, in addition to the ε parameter in the Rips filtration, one can compute the multiparameter persistent homology where the dataset grows with time in the other dimension. The feature engineering methods can include other vector representations of persistence diagrams, and they can be combined with classical signal-specific features. Lastly, since no classification method appears to be a clear winner, one might consider using ensemble learners to increase the accuracy levels on these datasets.

9. Appendix

This section contains the pseudocodes for the functions used in this study. Most functions can be parallelized. For instance, once the subwindows are created, one can compute the corresponding persistent homology in parallel for

all subwindows.

Algorithm 1: The subwindowing algorithm.

Input: x : univariate time series.
 fs : sampling frequency.
 ℓ : subwindow length (in seconds).
 κ : subwindow shift (in seconds).
Output: y : the sequence of sliding subwindows
Function *getSubwindows*
 Let L be the length of x .
 $s \leftarrow \kappa \cdot fs$
 $a \leftarrow 0$
 $b \leftarrow \ell \cdot fs - 1$
 $i \leftarrow 0$
 while $b < L$ **do**
 $y_i \leftarrow (x_a, \dots, x_b)$
 $a \leftarrow a + s$
 $b \leftarrow b + s$
 $i \leftarrow i + 1$
 end
 return y
end

Algorithm 2: Delay embedding of a subwindow.

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**
 $y_i \leftarrow (x_i, \dots, x_{i+d-1})$
 end
 return y
end

Algorithm 3: Computation of persistent homology from subwindows.

Input: A subwindow sw with sampling frequency fs .

Output: The sequence of persistence diagrams as in Figure 8.

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 ( $D_1, \dots, D_{10}$ )
```

end

Algorithm 4: Feature engineering.

Input: A persistence diagram D

Output: A 7-dimensional vector f

Function *getFeatures*

```
for  $h \in D$  do  
   $\ell_h \leftarrow death(h) - birth(h)$   
end  
 $f_1 \leftarrow \frac{1}{\sqrt{2}} \|\ell\|_1$   
 $f_2 \leftarrow \frac{1}{\sqrt{2}} \|\ell\|_\infty$   
 $S \leftarrow \sum_{h \in D} \ell_h$   
 $p \leftarrow \frac{1}{S} \ell$   
 $f_3 \leftarrow - \sum_{h \in D} p_h \ln(p_h)$   
Let  $\beta$  be the Betti curve of  $D$   
 $f_4 \leftarrow \|\beta\|_1$   
 $f_5 \leftarrow \|\beta\|_2$   
Let  $\lambda$  be the landscape vector of  $D$   
 $f_6 \leftarrow \|\lambda\|_1$   
 $f_7 \leftarrow \|\lambda\|_2$   
return ( $f_1, \dots, f_7$ )
```

end

Algorithm 5: Rolling mean and standard deviation of features

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

Conflict of interest

The authors declare no conflict of interest.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

- Altındaş, F., Yılmaz, B., Borisenok, S., & İçöz, K. (2021). Parameter investigation of topological data analysis for eeg signals. *Biomedical Signal Processing and Control*, *63*, 102196.
- Atienza, N., Escudero, L. M., Jimenez, M. J., & Soriano-Trigueros, M. (2019). Persistent entropy: a scale-invariant topological statistic for analyzing cell arrangements. *arXiv preprint arXiv:1902.06467*, .
- Björner, A. (1995). Topological methods. In *Handbook of combinatorics, Vol. 1, 2* (pp. 1819–1872). Elsevier Sci. B. V., Amsterdam.
- Bubenik, P. (2015). Statistical topological data analysis using persistence landscapes. *J. Mach. Learn. Res.*, *16*, 77–102.
- Chazal, F., De Silva, V., Glisse, M., & Oudot, S. (2012). The structure and stability of persistence modules. *arXiv preprint arXiv:1207.3674*, *21*.

- Chung, Y.-M., Hu, C.-S., Lo, Y.-L., & 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.
- Cohen-Steiner, D., Edelsbrunner, H., & Harer, J. (2007). Stability of persistence diagrams. *Discrete & computational geometry*, *37*, 103–120.
- Cohen-Steiner, D., Edelsbrunner, H., Harer, J., & Mileyko, Y. (2010). Lipschitz functions have 1 p-stable persistence. *Foundations of computational mathematics*, *10*, 127–139.
- Dirafzoon, A., Lokare, N., & Lobaton, E. (2016). Action classification from motion capture data using topological data analysis. In *2016 IEEE global conference on signal and information processing (globalSIP)* (pp. 1260–1264). IEEE.
- Edelsbrunner, H., & Harer, J. (2010). *Computational topology: an introduction*. American Mathematical Soc.
- Edelsbrunner, H., Letscher, D., & Zomorodian, A. (2000). Topological persistence and simplification. In *Proceedings 41st annual symposium on foundations of computer science* (pp. 454–463). IEEE.
- Emrani, S., Gentimis, T., & Krim, H. (2014). Persistent homology of delay embeddings and its application to wheeze detection. *IEEE Signal Processing Letters*, *21*, 459–463.
- Erden, F., & 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*, 958–962.
- Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., & Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, *101*, e215–e220.
- Healey, J. A., & Picard, R. W. (2005). Detecting stress during real-world driving tasks using physiological sensors. *IEEE Transactions on intelligent transportation systems*, *6*, 156–166.
- Hönig, F., Batliner, A., & Nöth, E. (2007). Fast recursive data-driven multi-resolution feature extraction for physiological signal classification. In *3rd Russian-Bavarian Conference on Bio-medical Engineering* (pp. 47–52).
- Ignacio, P. S., Dunstan, C., Escobar, E., Trujillo, L., & Uminsky, D. (2019). Classification of single-lead electrocardiograms: Tda informed machine learning. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)* (pp. 1241–1246). IEEE.

- Khasawneh, F. A., & Munch, E. (2016). Chatter detection in turning using persistent homology. *Mechanical Systems and Signal Processing*, *70*, 527–541.
- Majumdar, S., & Laha, A. K. (2020). Clustering and classification of time series using topological data analysis with applications to finance. *Expert Systems with Applications*, *162*, 113868.
- Majumder, S., Apicella, F., Muratori, F., & Das, K. (2020). Detecting autism spectrum disorder using topological data analysis. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1210–1214). IEEE.
- Makowski, D., Pham, T., Lau, Z. J., Brammer, J. C., Lespinasse, F., Pham, H., Schölzel, C., & Chen, S. H. A. (2021). Neurokit2: A python toolbox for neurophysiological signal processing. *Behavior Research Methods*, . URL: <https://doi.org/10.3758/s13428-020-01516-y>. doi:10.3758/s13428-020-01516-y.
- Marchese, A., & Maroulas, V. (2018). Signal classification with a point process distance on the space of persistence diagrams. *Advances in Data Analysis and Classification*, *12*, 657–682.
- Morozov, D. (2018). Dionysus 2 – library for computing persistent homology.
- Otter, N., Porter, M. A., Tillmann, U., Grindrod, P., & Harrington, H. A. (2017). A roadmap for the computation of persistent homology. *EPJ Data Science*, *6*.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.
- Perea, J. A. (2019). Topological time series analysis. *Notices of the American Mathematical Society*, *66*, 686–694.
- Perea, J. A., & Harer, J. (2015). Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, *15*, 799–838.
- Pun, C. S., Xia, K., & Lee, S. X. (2018). Persistent-homology-based machine learning and its applications—a survey. *arXiv preprint arXiv:1811.00252*, .
- Schmidt, P., Reiss, A., Duerichen, R., Marberger, C., & Van Laerhoven, K. (2018). Introducing wesad, a multimodal dataset for wearable stress and affect detection. In *Proceedings of the 20th ACM international conference on multimodal interaction* (pp. 400–408).

- Seversky, L. M., Davis, S., & Berger, M. (2016). On time-series topological data analysis: New data and opportunities. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 59–67).
- Takens, F. (1981). Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980* (pp. 366–381). Springer.
- Tempelman, J. R., & Khasawneh, F. A. (2020). A look into chaos detection through topological data analysis. *Physica D: Nonlinear Phenomena*, 406, 132446.
- Tralie, C., Saul, N., & Bar-On, R. (2018). Ripser. py: A lean persistent homology library for python. *Journal of Open Source Software*, 3, 925.
- Tralie, C. J., & Perea, J. A. (2018). (quasi) periodicity quantification in video data, using topology. *SIAM Journal on Imaging Sciences*, 11, 1049–1077.
- Umeda, Y. (2017). Time series classification via topological data analysis. *Information and Media Technologies*, 12, 228–239.
- Van Rossum, G., & Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.
- Wang, Y., Ombao, H., & Chung, M. K. (2019). Statistical persistent homology of brain signals. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1125–1129). IEEE.