CHANGE POINT DETECTION VIA HIERARCHICAL DIRICHLET PROCESS HIDDEN MARKOV MODELS WITH TOPOLOGICAL EMISSIONS

BY

Brent Oneil Joseph Young

BS, University of North Carolina – Chapel Hill, 1999

MS, University of North Carolina – Wilmington, 2005

Ph.D., Rutgers University – New Brunswick, 2011

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF SCIENCE

IN THE DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

AT FORDHAM UNIVERSITY

NEW YORK

DECEMBER, 2024

Contents

1 Introduction

Change Point Detection (CPD) is the task of discerning meaningful changes in signals, time series, or other streams of sequential data. CPD has a long history of study in statistics, signal processing, and machine learning stretching back to the 1950s. The original goal was to determine changes in the mean of independent, identically distributed (*iid*) Gaussian random variables for quality control in industrial processes. Since that time, researchers have used change point detection techniques in a variety of fields: speech processing, financial analysis, network traffic analysis, etc. The review paper by Truong, Oudre, and Vayatis [1] gives a nice selection of papers showing the diversity of applications for CPD.

Change Point Detection methods can be broadly grouped into two basic categories. *Online* CPD methods attempt to determine changes to the signal in real–time as they occur. *Offline* CPD methods retrospectively detect change points once all signals are received. There are a number of classic algorithms for both types. See [1] for a nice overview of the classic algorithms for offline CPD. The paper by Adams and MacKay [2] shows how Bayesian analysis can be used to predict the most likely next data point and so detect a change in the sequential data in real–time.

The focus in this thesis will be on the use of *Hidden Markov Models* (HMMs) for change point detection. As we review below, HMMs assume the process generating the sequential data has some (finite) number of particular states. The process begins in one of these states and at each time point has some probability of changing to one of the other states. The *Markovian property* assumes the probability of the next state depends only on the current state of the process. Once in a particular state, the process emits data independently of the history of the process. This is accomplished by choosing a random variable to model the emissions, and the parameters of the emission random variables are learned in the process of training the model. The connection to change point analysis is clear; change points in the time series should correspond to changes in state of the underlying Markov process. As described, using a Hidden Markov Model for change point detection is clearly an offline

technique. However, once the modeled is trained on historical data, it can be easily deployed for online detection as well. As more data is available, the underlying HMM can be retrained to improve its overall accuracy.

One major issue with traditional Hidden Markov Models is that the number of states must be specified in advance. If the number of states is known, this poses no issue. Otherwise, an obvious technique is to train the model with varying number of states and choose the one with best performance. A more sophisticated approach would be to ascertain the optimal number of states directly from the data. This can be achieved by grafting a *Hierarchical Dirichlet Process* (HDP) onto the Hidden Markov Model (HDP–HMM) which effectively allows the number of possible states to be infinite. For the purposes of change point detection, it also allows the model to respond to new types of variation in the signal as they occur.

HDP–HMM has been paired with a number of emission models to create a wide variety of signal processing tools. In this thesis, we focus on incorporating *topological data analysis* (TDA) with Markov Models to determine if this pairing is reasonably effective at detecting change points in various kinds of time series data. The goal of TDA is to use tools from computational homology to learn something about the overall "shape" of data. In particular, *persistent homology* attempts to find holes of various dimensions in the data and measure their lifetime as data points are connected to more and more of their neighbors. Chapters VII, VIII, and IX of [3] provide a detailed accounting of the techniques involved. The output of persistent homology is a *persistence diagram* which can be vectorized as a *persistence image.* We assume these persistence images are generated from a Gaussian random variable, and the HDP–HMM attempts to learn how many distinct states are present in the data along with the mean and covariance matrices from the persistence images assigned to those states.

To begin, we review the major tools used in the analysis: Hierarchical Dirichlet Process Hidden Markov Models and Persistent Homology. We then turn to details of the development of HDP–HMM with Topological Emissions in the Julia programming language. Next, the performance of the model on a number of synthetic time series is examined. In particular,

we look for change points in two–dimensional time series data with distinct changes in the underlying topology, detecting changes of variance in time series data, and changes in parameter for the famous Lorenz dynamical system (which induce dramatic changes in the shape of the data). Finally, the model will be tested against a number of real–world datasets to test its efficacy in various scenarios. Specifically, we examine human motion detection and change point detection in time series of images.

2 Hierarchical Dirichlet Process Hidden Markov Models

In this chapter, we introduce the essential ideas of Hidden Markov Models (HMMs). Once the basic concepts and algorithms are described, we consider various generalizations of HMMs using Hierarchical Dirichlet Processes (HDPs) which allow us to leave the number of states unspecified (and so learnable from the training data).

2.1 Essentials of Hidden Markov Models

A Markov Chain is a stochastic process where the current state of the process z_t can take on one of a finite set of possible values. The time index t is assumed to be a non–negative integer (i.e. $t \in \{0, 1, 2, 3, \cdots\}$), and for convenience the finite set of states are labeled by positive integers: $z_t \in \{1, 2, \cdots L\}$. At any time t, the current history of the Markov Chain is given by the sequence of states $(z_0, z_1, z_2, \ldots z_t)$. The Markovian property stipulates that the next state only depends on the history through the current state, and these transition probabilities are independent of t.

$$\mathbb{P}(z_{t+1}|z_0, z_1, z_2, \ldots z_t) = \mathbb{P}(z_{t+1}|z_t)$$

This allows us to specify the transitions using a probability matrix, $\boldsymbol{\pi}$.

$$\mathbb{P}(z_{t+1} = j|z_t = i) = \boldsymbol{\pi}_{ij}$$

Given the vector $\vec{\pi}_0$ describing the probability distribution of the initial state, the probability distribution of states at any future time step t can be computed by matrix multiplication. [4, Chapter 4]

$$\vec{\pi}_t = \boldsymbol{\pi}^t \vec{\pi}_0$$

For machine learning, we often have time series data $(x_0, x_1, x_2, \cdots, x_T)$, and our goal

is to predict the next data point x_{T+1} (here x_t could be any sort of data). One idea for attacking this problem is to assume there is an underlying Markov Process with a finite number of states with labels from the set $\{1, 2, 3, \cdots, L\}$.

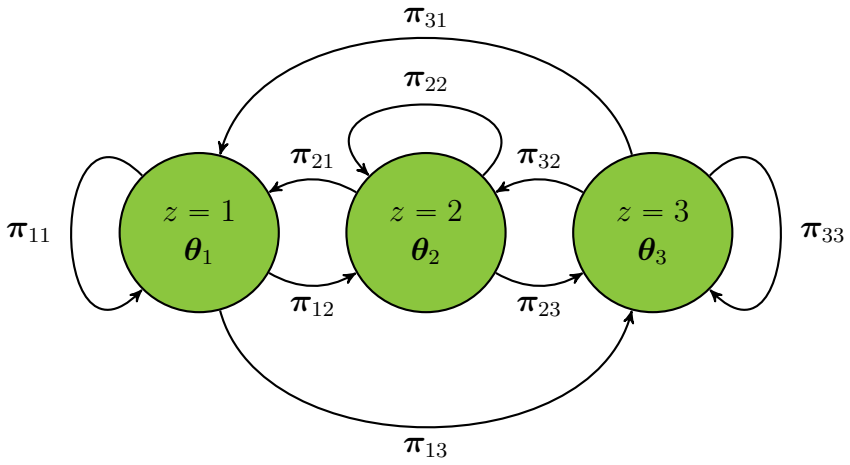


Figure 1: A Three State HMM

Each state z is now characterized by a number of *emission parameters*, $\boldsymbol{\theta}_s$, which control how the data point x_t is generated. The emission of data is assumed to depend only on the current state (and not any of the previous emissions).

$$\mathbb{P}(z_{t+1} = j | z_0, z_1, \cdots, z_t = i) = \mathbb{P}(z_{t+1} = j | z_t = i) = \boldsymbol{\pi}_{ij}$$

$$\mathbb{P}(x_{t+1} | (z_0, x_0), (z_1, x_1), \cdots, (z_t, x_t), z_{t+1} = j) = \mathbb{P}(x_{t+1} | z_{t+1} = j) = \mathbb{P}(x_{t+1} | \boldsymbol{\theta}_j)$$

Such a process is called a Hidden Markov Model, and the main tasks are inferring the transition matrix $\boldsymbol{\pi}$ and the emission parameters $\{\boldsymbol{\theta}_i\}_{i=1}^S$ from the training data. The primary algorithm for training an HMM is the *Forward–Backward Algorithm* (also known as the *Baum–Welch Algorithm*) which is a special case of the Expectation Maximization Algorithm. For details, see Appendix A of [5].

A major issue when training a Hidden Markov Model is deciding how many states the model should have. In some applications, the number of states may be known in advance and training is straightforward. For our intended use, the number of states is typically unknown

since we know neither the number of change points in the data nor the number of states producing the data we are attempting to model. As with K–Means, one approach is to train the model with a wide range of total states and choose the one which maximizes the likelihood of the training data. An alternate approach is to learn the optimal number of states using a *Hierarchical Dirichlet Process* (HDP).

2.2 Hierarchical Dirichlet Processes

The basic Dirichlet Process was proposed in 1973 by Ferguson [6] and is effectively a probability measure on the space of probability measures. In other words, sampling from a Dirichlet Process gives a probability measure on some sample space. These processes are characterized by two parameters: a *scaling parameter* α_0 and a *base probability measure* μ_0. If we sample a random measure μ via $\mu \sim \mathrm{DP}(\alpha_0, \mu_0)$, then almost surely μ will be a discrete probability measure of the form

$$\mu = \sum_{i=1}^{\infty} \beta_i \delta_{\phi_i}.$$

Here, the sequence $(\phi_i)_{i=1}^{\infty}$ is drawn *iid* from the base measure μ_0, and δ_{ϕ_i} represents the Dirac Delta measure centered at value ϕ_i. The sequence of coefficients $(\beta_i)_{i=1}^{\infty}$ can be generated by the *stick–breaking construction*.

We begin the construction by drawing a sequence of auxiliary probabilities $(p_i)_{i=1}^{\infty}$ chosen *iid* by

$$p_i \sim \mathrm{Beta}(1, \alpha_0).$$

Given these probabilities, we define

$$\beta_i = p_i \prod_{j=1}^{i-1} (1 - p_i).$$

We can think of $\beta_1 = p_1$ as the length of the sub–interval $[0, p_1]$ of the unit interval $[0, 1]$. Once we break off this portion, the remaining interval has length $1 - p_1$, and $\beta_2 = p_2(1 - p_1)$ is

then a portion of this remainder. Breaking off this sub–interval leaves a remainder of length $(1-p_2)(1-p_1)$, and $\beta_3 = p_3(1-p_2)(1-p_1)$ is a portion of this remainder. Continuing this process gives us the sequence $(\beta_i)_{i=1}^{\infty}$ which is guaranteed to sum to one. Since this process is crucial to later developments, we refer to the stick breaking procedure by

$$\boldsymbol{\beta} \sim \text{GEM}(\alpha_0)$$

(where GEM stands for Griffiths, Engen, and McCloskey [7]).

In many instances, data are naturally subdivided into groups, and we need to learn a probability distribution for each group. If we simply choose

$$\mu_j \sim \text{DP}(\alpha_0, \mu_0)$$

iid for each group, we run into the issue that different groups will typically have a completely different sets of parameters $(\phi_{ji})_{i=1}^{\infty}$. If the base measure μ_0 is a continuous probability distribution (as is often the case), then the sets of parameters will be distinct with probability one. To remedy this, Teh et al. proposed a Hierarchical Dirichlet Process which samples the base measure μ_{base} from another Dirichlet Process.[7]

$$\begin{aligned} \mu_{\text{base}} &\sim \text{DP}(\gamma, \nu) \\ \mu_j &\sim \text{DP}(\alpha_0, \mu_{\text{base}}) \end{aligned}$$

In other words, we first sample the base distribution μ_{base} from a Dirichlet Process with scale parameter γ and base probability measure ν. Then, we sample a sequence of measures $(\mu_j)_{j=1}^{N}$ *iid* from a Dirichlet Process with parameters α_0 and μ_{base}. Since μ_{base} is a discrete probability measure (almost surely), the sequence of measures selected from the subsequent Dirichlet Process will share the same parameters $\{\phi_{ji}\}_{i=1}^{\infty} = \{\phi_i\}_{i=1}^{\infty}$ with probability one (though in a different order and with a different sequence of weights $(\beta_{ji})_{i=1}^{\infty}$).

Training the model consists of inferring the parameters γ, ν, and α_0 from the training data. This can be accomplished by one of a variety of Markov Chain Monte Carlo (MCMC) sampling schemes which sequentially update these parameters on several passes through the training data. Details are given in Section 5 of [7].

2.3 HDP–HMMs

Pairing Hidden Markov Models with Hierarchical Dirichlet Processes gives us a way to effectively make the number of states in the HMM infinite. That is, there is no need to set the number of states in an HDP–HMM *a priori*, and we can infer the actual number of states in the model by counting how many states are occupied once training is complete (and perhaps pruning low occupancy states). As we will only have a finite amount of training data to work with, only a finite number of states will ever be occupied. The first such model was proposed by Teh et al. in [7].

Essentially, we use the base measure ν to generate a sequence of *iid* emission parameters $(\boldsymbol{\theta}_i)_{i=1}^{\infty}$. Since the HDP generates a discrete probability measure with probability one, any subsequent measure generated will share this same sequence. As we are building a Markov Chain, we only need the transition probabilities, $\boldsymbol{\pi}$, which we think of conceptually as an infinite matrix of probabilities. Each row of the transition matrix is generated independently by the stick–breaking procedure described above.

$$\boldsymbol{\pi}_{i\cdot} \sim \mathrm{GEM}(\alpha_0)$$

As before, training the model involves inferring ν (the probability measure giving the emission parameters) and the scale parameter α_0.

One issue impacting the performance of the standard HDP–HMM is that it does not treat self–transitions any differently than other transitions. This can lead to trained models which oscillate wildly between states having nearly identical emission parameters. Considering our

ultimate application is change point detection in time series data, we would expect the data to persist in a state for a significant number of time steps if the change points are to be at all meaningful. To ameliorate this deficiency, Fox et.al. proposed a *sticky* version of the original HDP–HMM (sHDP–HMM).[8] The idea is to add a parameter κ to the model which increases the likelihood of every self–transition.

$$\begin{aligned} \boldsymbol{\beta} &\sim \text{GEM}(\gamma) \\ \boldsymbol{\pi}_{i\cdot} &\sim \text{DP}\left(\alpha_0 + \kappa, \frac{\alpha_0\boldsymbol{\beta} + \kappa\delta_i}{\alpha_0 + \kappa}\right) \end{aligned}$$

In words, we first generate $\boldsymbol{\beta}$ by the stick–breaking construction. We then generate the i–th row of $\boldsymbol{\pi}$ using the Dirichlet Process with scale parameter $\alpha_0 + \kappa$ and the probability distribution $\mu_0 = \dfrac{\alpha_0\boldsymbol{\beta} + \kappa\delta_i}{\alpha_0 + \kappa}$ over the natural numbers. This leads to a model where

$$\mathbb{E}(\boldsymbol{\pi}_{ij}) = \begin{cases} \frac{\alpha_0\boldsymbol{\beta}_j}{\alpha_0+\kappa}, & j \neq i \\ \frac{\alpha_0\boldsymbol{\beta}_i+\kappa}{\alpha_0+\kappa}, & j = i \end{cases}.$$

A slight generalization of the sHDP–HMM was proposed by Zhou et al. in [9]: the disentangled sticky HDP–HMM (dsHDP–HMM). Instead of a single parameter κ, we generalize to a vector $\boldsymbol{\kappa}$ which allows the model to learn a different self–transition parameter for each state. The basic setup is to choose all the components $\boldsymbol{\kappa}_i$ *iid* according to

$$\boldsymbol{\kappa}_i \sim \text{Beta}(\rho_1, \rho_2)$$

where ρ_1 and ρ_2 are new parameters for the model. As above, we sample the vector $\boldsymbol{\beta}$ via

$$\boldsymbol{\beta} \sim \text{GEM}(\gamma).$$

We then sample an auxiliary transition matrix $\tilde{\boldsymbol{\pi}}$ whose rows are chosen *iid* by

$$\tilde{\boldsymbol{\pi}}_{i\cdot} \sim \mathrm{DP}(\alpha_0, \boldsymbol{\beta}).$$

Finally, we construct the actual transition matrix $\boldsymbol{\pi}$ using

$$\boldsymbol{\pi}_{i\cdot} = (1 - \boldsymbol{\kappa}_i)\tilde{\boldsymbol{\pi}}_{i\cdot} + \boldsymbol{\kappa}_i \delta_i.$$

This distribution is equivalent to first sampling a biased coin toss $w \sim$Bernoulli$(\boldsymbol{\kappa}_i)$. If $w = 1$ we remain in state i; otherwise, we select a new state according to the probabilities given by the i–th row of $\tilde{\boldsymbol{\pi}}$.

2.4 Weak Limits of HDP–HMMs

As noted by Fox et al. in [8], the usual MCMC sampling schemes to infer parameters in the various types of HDP–HMMs can be slow to converge. One way to alleviate this issue is to use the *degree L weak limit approximation* to the Dirichlet Process. This involves picking an upper bound L on the number of states (chosen much larger than the anticipated number of states) and truncating the stick–breaking process after $L-1$ steps (the probability of the L–th state is determined by the previous ones). It can be shown that this truncated version of the stick–breaking process simply gives a Dirichlet random variable.

$$\mathrm{GEM}_L(\alpha_0) \equiv \mathrm{Dir}(\alpha_0/L, \alpha_0/L, \cdots, \alpha_0/L)$$

In the context of HDP–HMMs, this truncation allows us to use a variant of the Baum–Welch Algorithm to train the model (offering a significant increase in efficiency). We detail the implementation of this scheme in Chapter 4.

3 Persistent Homology using Persistence Images

Chapter 2 largely dealt with the structure of Hidden Markov Models (and their generalization via Hierarchical Dirichlet Processes). In this chapter, we examine the specific analysis performed on the time series data: *persistent homology*. This branch of algebraic topology tries to associate algebraic data (i.e. Abelian groups) to topological spaces. The result is a *persistence diagram* which encodes the birth and death of various topological features in the data. This diagram can be vectorized into a *persistence image* which can then be analyzed in a number of different ways. In the sections below, we give a brief introduction to this powerful area of data analysis.

3.1 A Primer on Algebraic Topology

A *topological space* is essentially a point set X together with a specification of open sets for X.[1] Intuitively, the collection of open sets gives a notion of which points in X are "close" to one other. This generalizes the geometric notion of shape to an axiomatic setting. For current purposes, we assume our topological spaces are *connected* (essentially, any space X under consideration should be a single, contiguous object).

One of the major questions in topology is whether one topological space can be continuously deformed into another (such spaces are said to be *homeomorphic*). Currently, there is no general answer to this question. However, topologists have developed a number of tools which are capable of distinguishing between topological spaces in certain instances. Since these tools can detect differences in shape (at least in part), they have found wide use in data analysis. In Algebraic Topology, algebraic data is associated to a topological space in such a way that the algebraic structures capture some aspect of the underlying topology. If two spaces yield different data from the process, then those spaces cannot be homeomorphic.

Broadly speaking, the two major tools of Algebraic Topology are *homotopy* and *homology*. [10] Of the two, homotopy is certainly the easier to understand conceptually. Homotopy

[1] The collection of opens sets is called a *topology* on X and must satisfy various axioms.

associates a sequence of algebraic groups[2] to a topological space, and if two spaces have different sequences of groups, they must be topologically distinct. The simplest of the homotopy groups is referred to as the *fundamental group* of a space X, $\pi_1(X)$. This group is constructed by first picking any point $x_0 \in X$ (the choice is irrelevant for a connected space) and considering every possible loop in X based at x_0. Two loops are considered equivalent if they can be continuously deformed into each other without leaving the space X. The fundamental group consists of the equivalence classes of these loops.[3]

Figure 2 below shows how the fundamental group can be visualized for a disk versus an annulus. In the disk, note that loop ℓ_1 can be continuously deformed into loop ℓ_2 *without*

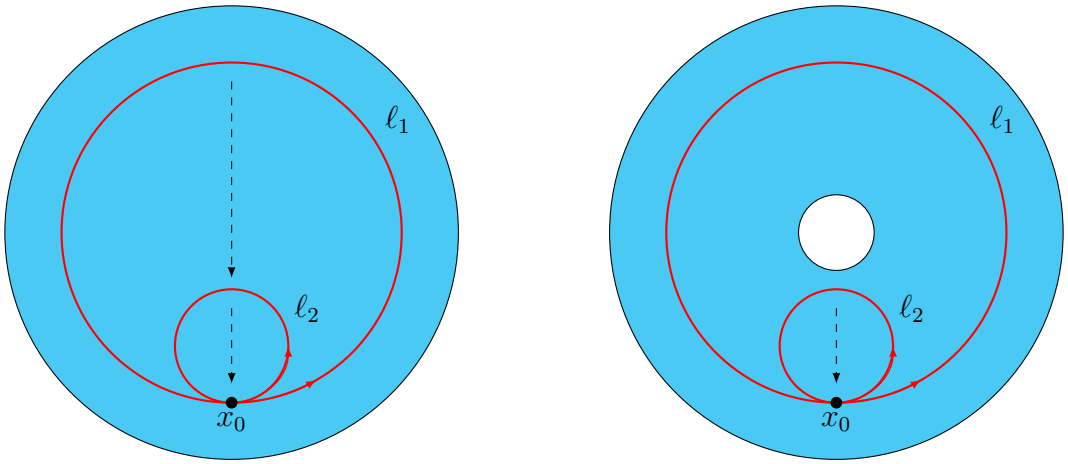


Figure 2: Fundamental Group of a Disk vs. an Annulus

leaving the topological space. In fact, any loop based at any point can be continuously shrunk back to the base point (it helps to imagine the loops as tiny rubber bands tacked down at the base point). As it turns out, $\pi_1(\text{disk}) = \{e\}$ (the trivial group consisting of one element) since any loop in the disk can be continuously deformed into any other loop.

The annulus is different. Certainly ℓ_2 can be deformed back to the point x_0, but ℓ_1 gets caught around the hole in the center of the annulus. As a result, the fundamental group of the

[2] A group is a set of objects G together with a binary operation $* : G \times G \to G$ satisfying certain properties. In particular, there must be an identity element e in G, and each element $g \in G$ must have an inverse $g * g^{-1} = g^{-1} * g = e$.

[3] The group operation is concatenation of loops, and the inverse of a loop is simply the loop traversed in the opposite direction.

annulus is completely different from the disk. In fact, it can be shown that $\pi_1(\text{annulus}) = \mathbb{Z}$ (the set of integers). The idea is that any loop in the annulus is characterized by how many times it loops around the central hole, and whether it loops around in a clockwise or counter–clockwise manner.

As another example, it can be shown that $\pi_1(\text{sphere}) = \{e\}$ since any loop on the surface of a sphere can be continuously shrunk back to a point (or said differently, the surface of a sphere has no holes). On the other hand, $\pi_1(\text{torus}) = \mathbb{Z} \times \mathbb{Z}$ which consists of pairs of integers. Essentially, a torus has a large central hole and another smaller hole laterally. The pair of integers counts how many times a loop winds around each hole and in what direction.

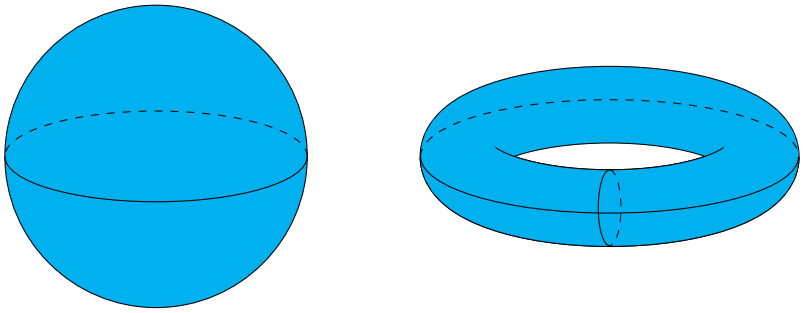


Figure 3: Sphere vs. Torus

The fundamental group can be generalized into higher dimensional analogues giving a sequence of higher dimensional homotopy groups for any topological space. Note that only a finite number of these groups will be non–trivial for any finite–dimensional topological space. The drawback to homotopy is that the groups it generates can be quite complicated in general, making homotopy rather difficult to calculate algorithmically. This impasse led to the development of *homology groups*.

The fundamental building blocks of homology are n–simplices which are generalizations of triangles.[4] In Figure 4, note that each face of a simplex inherits an orientation based on the (arbitrary) numbering of the vertices. Homology begins by triangulating a topological space to produce a collection of n–simplices for $n \in \{0, 1, 2, \cdots, d\}$ where d is the dimension of the space. If the collection of n–simplices is denoted $\left\{\Delta_i^{(n)}\right\}_{i=1}^{N_n}$, then the primary objects

[4] Homology can also be defined using cubical simplices.

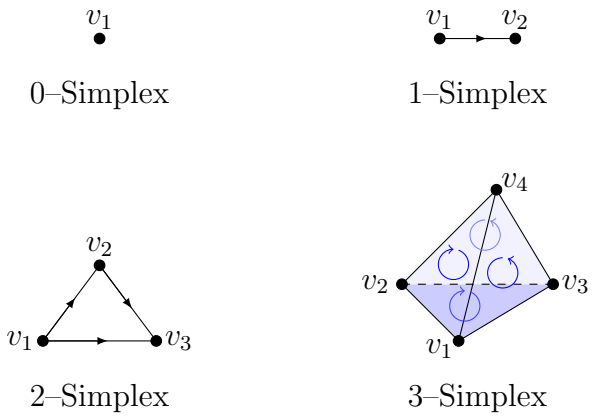


Figure 4: A Gallery of n–Simplices

under consideration in homology are *chains*: formal sums of simplices of the same dimension with coefficients typically taken from the integers (or the integers modulo p).

$$c = \sum_{i=1}^{N_n} a_i \Delta_i^{(n)}$$

The central operation we perform on chains is the linear *boundary operator* which produces a chain representing the boundary of a given simplex.

$$\partial[v_1, v_2, \cdots, v_k] = \sum_{i=1}^{k} (-1)^{i+1} [v_1, v_2, \cdots, \hat{v}_i, \cdots, v_k]$$

The notation $\hat{v}_i$ indicates that vertex i has been removed from the simplex. Note the alternating sum based on which index has been eliminated.[5] As a particular example, the boundary of the tetrahedron in Figure 4 is given by

$$\partial[v_1, v_2, v_3, v_4] = [v_2, v_3, v_4] - [v_1, v_3, v_4] + [v_1, v_2, v_4] - [v_1, v_2, v_3].$$

If a chain has the property that $\partial c = 0$, then it is called a *cycle*. In particular, *the boundary of any chain is always a cycle.* In other words, applying the boundary operator twice always

[5]The boundary of a 0–simplex is defined to be 0: $\partial[v] = 0$.

results in zero. A simple example is given by the boundary of a 2–simplex.

$$\begin{aligned}\partial\left(\partial[v_1, v_2, v_3]\right) &= \partial[v_2, v_3] - \partial[v_1, v_3] + \partial[v_1, v_2] \\ &= [v_3] - [v_2] - [v_3] + [v_1] + [v_2] - [v_1] = 0\end{aligned}$$

The n–th homology group, $\mathcal{H}_n(X)$, of a topological space X consists of the n–dimensional cycles in X modulo cycles that are boundaries of $(n+1)$–dimensional chains. In effect, $\mathcal{H}_n(X)$ counts the n–dimensional "holes" in a topological space by looking at all possible cycles (i.e. chains with no boundary) and removing those that come from the boundary of a higher–dimensional structure in X. The first three homology groups are the easiest to understand geometrically.

- $\mathcal{H}_0(X)$ counts the number of connected components in X.
- $\mathcal{H}_1(X)$ counts the number of holes in X (analogous to the hole in an annulus).
- $\mathcal{H}_2(X)$ counts the number of voids in X (analogous to the void in the interior of a sphere).

While homology is more abstract, it turns out to be far easier to compute than homotopy. In the next section, we look at a particularly nice variant of homology known as *persistent homology* which is well adapted to the characterization of point–clouds.

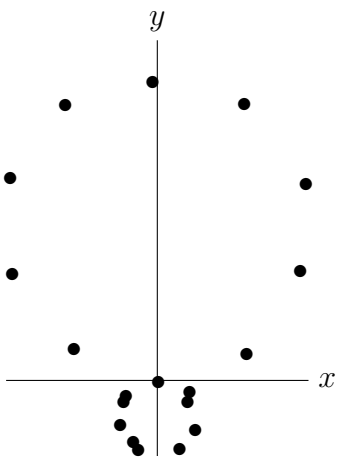
3.2 Persistent Homology

Imagine we have a finite set of distinct points in the plane given by

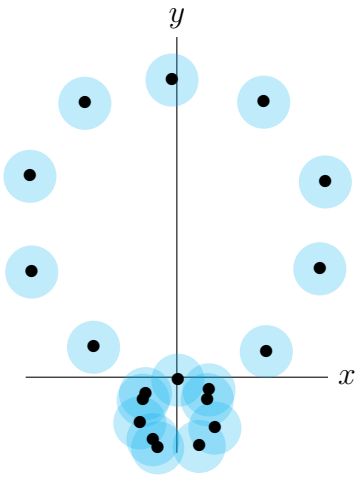
$$\boldsymbol{X} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}.$$

Since the points are isolated, there is no interesting topology to work with; we need to connect the points to one another to produce interesting topological structures. One way

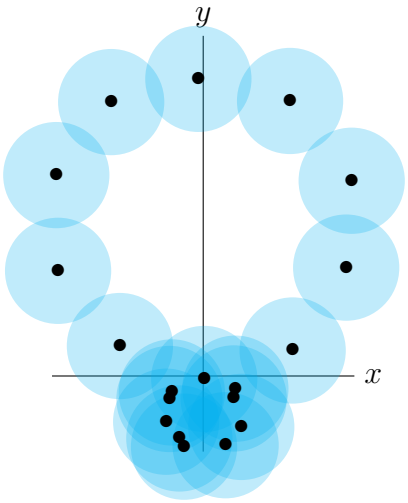
to accomplish this is to pick a positive value ϵ and connect all points that are within ϵ of one another. This will naturally give us a collection of simplices known as the Vietoris–Rips complex of the dataset.



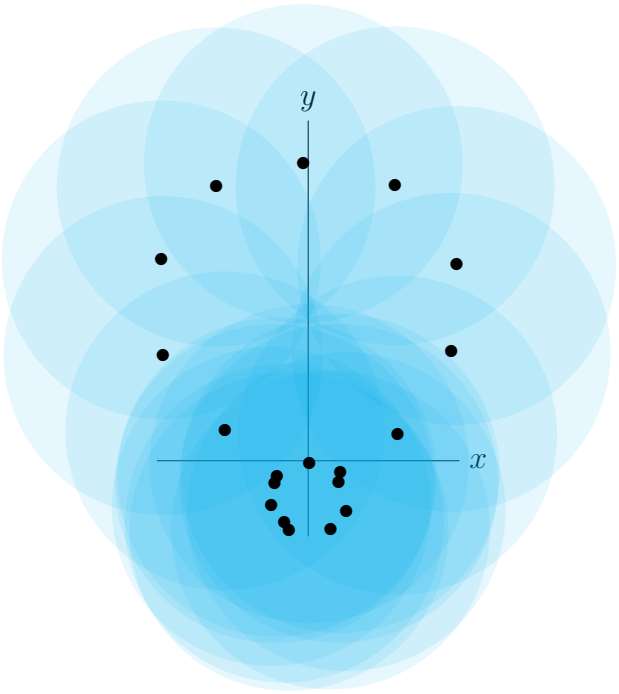
(a) A Simple Point–Cloud



(b) Point–Cloud Connected with Small ϵ



(c) Point–Cloud Connected with Moderate ϵ



(d) Point–Cloud Connected with Large ϵ

Figure 5: Homology with Various Sizes of ϵ

For the point–cloud in Figure 5a, there clearly seem to be two loops of different sizes. If we select a relatively small value of ϵ, the smaller loop will be joined while the points in the

larger loop will remain disconnected. Notice in Figure 5b that a tiny hole still remains in the smaller loop indicating that we would pick up a component in $\mathcal{H}_1$ for this choice of ϵ. With a larger choice of ϵ, the smaller loop is completely filled (and so it contributes nothing to $\mathcal{H}_1$) while the bigger loop is now completely connected, adding a component to the $\mathcal{H}_1$. If we make ϵ too large, all interesting topological features vanish. It would seem that no single value of ϵ allows us to capture all interesting topological features of the data.

Persistent Homology remedies this by allowing ϵ to vary between zero and infinity. It records the value of ϵ when an interesting topological feature appears and the value when it eventually disappears as ϵ increases. The result is a *persistence diagram* recording the birth and death of topological features. Notice that the entire bottom half of the Persistence Diagram is empty (since a feature must be born before it can die).

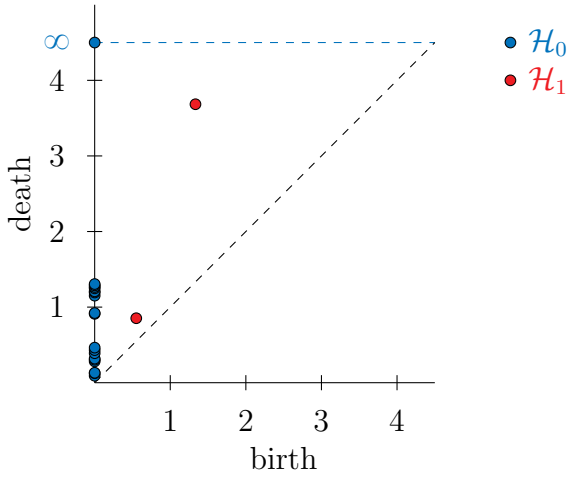


Figure 6: Persistence Diagram for Figure 5a

The points corresponding to $\mathcal{H}_0$ keep track of the number of connected components. When $\epsilon = 0$, there are as many connected components as distinct points in the cloud. As ϵ is increased, the points are merged together until they ultimately form one single component (which technically persists for all values of ϵ no matter how large).

The two points corresponding to $\mathcal{H}_1$ capture the two loops in the data. The smaller loop is born roughly at $\epsilon \approx 0.6$ and persists until $\epsilon \approx 0.85$. The other point corresponds to the larger loop which is born at $\epsilon \approx 1.3$ and dies at $\epsilon \approx 3.7$.

Another representation of Persistent Homology is a *Tilted Persistence Diagram* which encodes points as

$$(\text{birth, persistence}) = (\text{birth, death} - \text{birth}).$$

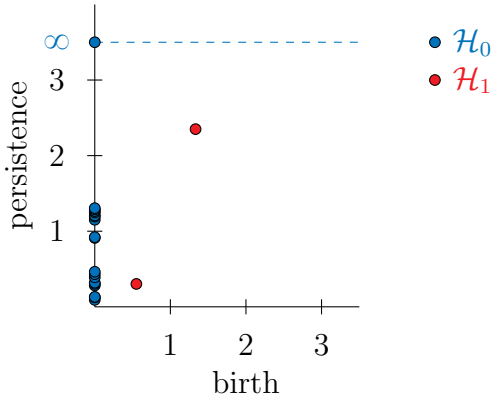


Figure 7: Tilted Persistence Diagram for Figure 5a

For real data, there are often a lot of noise points in the persistence diagrams (as transitory topological features come into being and quickly die). Strong topological features are well above the main diagonal in a standard Persistence Diagram (or well above the birth axis on a Tilted Persistence Diagram).

Computing persistent homology is no easy task (see Part C of [3] for details), and the complexity of computing $\mathcal{H}_d$ increases dramatically with d.[6] For that reason, most use cases are limited to $\mathcal{H}_0, \mathcal{H}_1$, and $\mathcal{H}_2$ unless there is serious need for higher dimensional analysis. While there are a number of packages available for computational homology, the most well known is Ripser which was first developed in 2015.[11] While the original version is written in C++, it has been ported into many languages commonly used in data science – including Python, Julia, and Matlab.

[6] In fact, the runtime can be $\mathcal{O}(N^d)$ where N is the number of data points and d is the maximum homological dimension.

3.3 Persistence Images

There are a number of metrics that can be used to directly compare persistence diagrams. The two most commonly used are the *bottleneck* and *Wasserstein* distances.[3, Section VIII.2] In both cases, the metric is computed by attempting to find the optimal bijection between points in the respective diagrams.[7] As the number of points in the diagrams increases, these metrics become increasingly expensive to compute.

A more efficient means of comparing persistence diagrams is to convert them into *persistence images* which are effectively pixelations of the underlying persistence diagram. [12] Once computed, persistence images can be treated vectors in $\mathbb{R}^n$ and used in almost any machine learning pipeline.

The process begins by taking a persistence diagram D and converting it to its tilted representation $T(D)$ as described above. Next, a *persistence surface* is constructed from all points $u = (b, p)$ in the tilted persistence diagram.

$$\rho_D(x, y) = \sum_{u \in T(D)} w(u)\phi_u(x, y)$$

The weight function w ensures that the persistence image is stable against small perturbations of the persistence diagram. Since points close to the horizontal axis in the tilted diagram are likely due to noise in the data, we want to discount those points in the image. The authors of [12] recommend

$$w(u) = w(b, p) = \begin{cases} 0, & p \leqslant 0 \\ \frac{p}{P}, & 0 < p \leqslant P \\ 1, & p > P \end{cases}$$

where P is the maximum persistence of any feature (excluding the single point of $\mathcal{H}_0$ at

[7] If the two diagrams do not have the same number of features, points can be added on the main diagonal since these represent features with no actual persistence.

infinity). The shape function ϕ_u is typically taken to be a symmetric Gaussian centered at u (with some choice of standard deviation σ).

$$\phi_u(x,y) = \frac{1}{2\pi\sigma^2} e^{-[(x-b)^2-(y-p)^2]/2\sigma^2}$$

Once the persistence surface has been computed, we can easily obtain a pixelation by dividing the tilted persistence diagram into an $n \times n$ grid and integrating the persistence surface over each cell. If the collection of cells is $\mathcal{C} = \{c_{ij}\}$, then the persistence image is the matrix of pixel values

$$I(D)_{ij} = \iint_{c_{ij}} \rho_D(x,y)\, dxdy.$$

To summarize, we have the following pipeline to produce a vectorized version of a persistence diagram.

$$\text{data} \xrightarrow[\text{dim}]{} D \longrightarrow T(D) \xrightarrow[w,\sigma]{} \rho_D \xrightarrow[n]{} I(D)$$

To create the persistence diagram, we first have to select the maximum topological dimension. Once the tilted diagram has been constructed, we must choose a weight function w and a standard deviation σ for the persistence surface. Finally, we have to choose a grid size n for the pixelation.[8] According to results cited in [12], persistence images are fairly robust to changes in these parameters.

[8] The default setting in Julia's `Ripserer` package is a 5×5 grid.[13]

4 HDP–HMM with Topological Emissions

In this chapter, we detail the overall pipeline for the Hierarchical Dirichlet Process Hidden Markov Model with Topological Emissions (HDP–HMMte). We begin with the data pre–processing steps to turn a time series into a sequence of persistence images. The development of the HDP–HMM mechanism itself follows the development of Zhou et al. in [9] but using the weak limit approach on the advice of Fox et al. [8] The entire pipeline was developed in the Julia programming language.

4.1 Data Pre–Processing

The majority of the data under consideration are presented as times series with total length T: $\boldsymbol{X} = (x_0, x_2, x_3, \cdots, x_{T-1})$ where each x_t is an individual data point from the time series. To convert this time series into a sequence of persistence images, we first select a window size ω and a slide rate $\jmath$. We can then partition the time series into a number of (possibly) overlapping frames in the obvious way.

$$
\begin{aligned}
\boldsymbol{F}_0 &= (x_0, x_1, \cdots, x_{\omega-1}) \\
\boldsymbol{F}_1 &= (x_\jmath, x_{\jmath+1}, \cdots, x_{\jmath+\omega-1}) \\
&\vdots \\
\boldsymbol{F}_i &= (x_{i\jmath}, x_{i\jmath+1}, \cdots, x_{i\jmath+\omega-1}) \\
&\vdots \\
\boldsymbol{F}_F &= (x_{F\jmath}, x_{F\jmath+1}, \cdots, x_{F\jmath+\omega-1})
\end{aligned}
$$

Each frame consists of ω points from the original time series, and the final frame number is given by the largest value of F so that $F\jmath + \omega - 1 \leqslant T - 1$:

$$F = \left\lfloor \frac{T - \omega}{\jmath} \right\rfloor.$$

For each frame $\boldsymbol{F}_i$, we use the `Ripserer` package [13] in Julia to compute a persistence diagram. This requires us to choose a maximum topological dimension, d. For this thesis, homology groups were largely limited to $\mathcal{H}_0$ and $\mathcal{H}_1$. For time series data of higher dimension (i.e. when $\dim(x_t) \gg 3$), it might be sensible to increase the maximum topological dimension – though computation time is exponential in the topological dimension. For the applications below, a maximum dimension of $d = 1$ seemed perfectly adequate.

Once the persistence diagrams have been computed, `Ripserer` has a method to convert them into persistence images as described in Chapter 3.3. The default settings for the weight function and variance of the Gaussian used to create the persistence surface were maintained throughout. For the size of the persistence image, the typical choice was to use a 10×1 representation for 0–th dimensional persistent homology (since all points lie on the vertical line $b = 0$ in the persistence diagram) and a 5×5 image for all other dimensions. The resulting 5×5 matrices are then stacked to produce 25×1 vectors representing the persistence diagram. All of these parameters can be varied, but choosing very large sizes for the persistence images will create issues when using them in the Hidden Markov Model. `Ripserer` returns the resulting converter so that both the training and test sets can be converted identically. Once the conversion is complete, each frame is represented by a collection of vectors representing the persistence images: one for each topological dimension from 0 up to the maximum homological dimension d.

$$\boldsymbol{F}_i = (\boldsymbol{P}_{i0}, \boldsymbol{P}_{i1}, \cdots \boldsymbol{P}_{id})$$

Throughout this paper, $d = 1$ for almost all applications.

The final choice in pre–processing is how to incorporate the various persistence images

into data useful for machine learning purposes. There are three obvious strategies we could adopt.

1) Stack all topological dimensions from the persistence images into a single vector.

2) Treat each topological dimension as an independent emission for a given state in the Hidden Markov Model.

3) Create a single persistence image which is a (weighted) combination of all topological dimensions.

The first option is likely to cause issues when we attempt to compute covariance matrices to characterize the emission parameters. While the matrices we compute are positive definite in theory, in practice they typically have eigenvalues close to 0 – resulting in numerical problems when they are inverted. This is occasionally an issue in the second approach but one that is relatively easy to correct (by adding a small value ϵ to all entries in the main diagonal). We use the second approach throughout this study. To summarize these considerations, we make the (questionable) assumption that data from different topological dimensions are roughly independent, and we compute a separate persistence image for each dimension. Using the independence assumption, we multiply the likelihoods from each topological dimension to obtain an overall likelihood for the entire persistence image. An interesting future research project would be to compare this approach to one where all topological data is combined into a single persistence image in some way that preserves the topological dimension of each point.

4.2 The Disentangled Sticky Hierarchical Dirichlet Process Hidden Markov Model

After the pre–processing phase, both the training and test sets consist of a sequence of frames with each frame containing a set of vectors representing persistence images; that is,

each frame contains d vectors where d is the maximal topological dimension for persistent homology.

$$\boldsymbol{F} = (\boldsymbol{F}_0, \boldsymbol{F}_1, \cdots, \boldsymbol{F}_F)$$
$$\boldsymbol{F}_i = (P_{i0}, P_{i1}, \cdots, P_{id})$$

Our goal is to learn a sequence of discrete state variables $\boldsymbol{Z} = (z_1, z_2, \cdots, z_F)$ from a hidden Markov process and an associated set of emission parameters for each state $\{\boldsymbol{\theta}_s\}_{s\in\mathbb{Z}}$ so that frame $\boldsymbol{F}_i$ can be thought of as a sample from a random variable with parameters $\boldsymbol{\theta}_{z_i}$.

Borrowing from [9], a graphical model of the disentangled sticky HDP–HMM (dsHDP–HMM) is shown in Figure 8. The major groups of hyperparameters for the model are ρ_1 and ρ_2 (which determine the self–transition parameters κ_j), γ and α (which determine the normal HMM transition matrix $\boldsymbol{\pi}$), and H (the random variable selecting the emission parameters $\boldsymbol{\theta}_j$). Since we are using a weak limit of the dsHDP–HMM, we must also set an upper bound on the total number of states, L. If a significant portion of the L states are occupied after training, we can simply increase L and train again. For the studies in this thesis, $L = 20$ was adequate for all purposes.

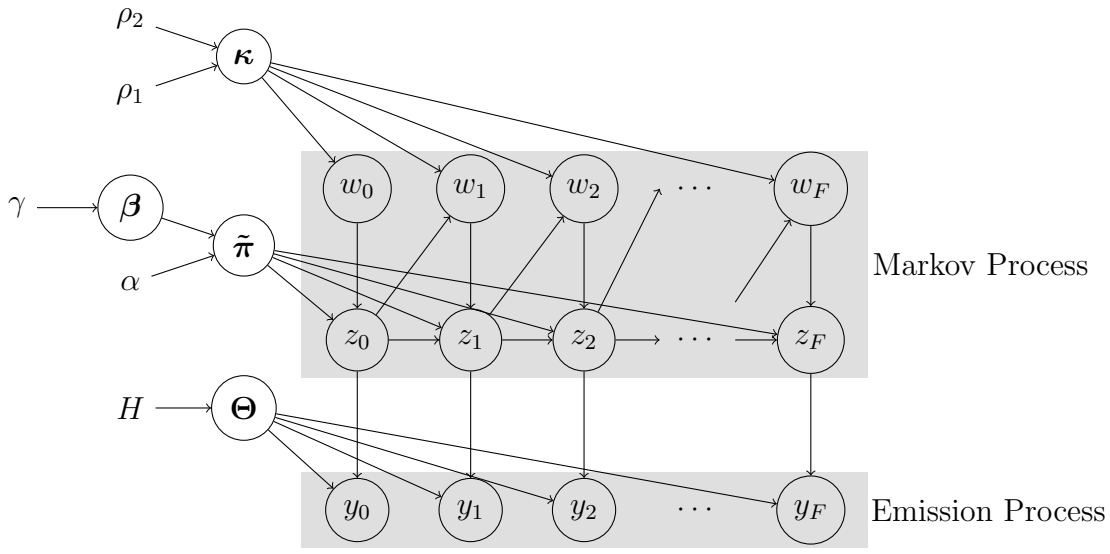


Figure 8: Graphical Model of the dsHDP–HMM

One round of the sampler for the dsHDP–HMM proceeds in five major stages.

1) Update the state variables $\boldsymbol{Z} = (z_t)_{t=0}^{F}$ and auxiliary variables $\boldsymbol{W} = (w_t)_{t=0}^{F}$.

2) Update the self–transition parameters $\boldsymbol{\kappa} = (\kappa_s)_{s=1}^{L}$.

3) Update the Markov parameters $\boldsymbol{\beta}$ and $\tilde{\boldsymbol{\pi}}$.

4) Update the emission parameters $\boldsymbol{\Theta} = (\boldsymbol{\theta}_s)_{s=1}^{L}$

5) Update the hyperparameters ρ_1, ρ_2, γ, and α.

These stages are repeated for some preset number of rounds. At the end of sampling, the set of parameters with the largest log–likelihood is taken to be the best model for the training data. Since we are updating parameters through a Bayesian process at each step, we are effectively taking a (biased) random walk in the parameter space. As such, we cannot expect the log–likelihood to monotonically increase to a local maximum (and indeed, there may be multiple local maxima). But we can expect it to enter a neighborhood of a local maximum rather rapidly.

Once trained, we can extract the underlying Markov Model by keeping only the populated states and their emission parameters and possibly an extra state to catch emissions from novel data categories. This allows us to evaluate the trained HMM on test data to see how well the model generalizes. This process gives us some idea how to use the dsHPD–HMM as a *quasi*–online change point detection method. After an initial round of training using previously gathered data, the model can be used to detect real–time changes in data. If the reserved "new state" in the trained model is ever populated, we can update the HMM using some portion of the old training data along with the data producing the new state.

4.3 Updating the States

In theory, we should be able to initialize the states, $\boldsymbol{Z} = (z_t)_{t=0}^{F}$, randomly and allow the algorithm to find the optimal assignments. In order to speed up convergence, we use two

different procedures to initialize the state variables in a more efficient way. The first involves running K–Means on the dataset $\boldsymbol{F}$ for a range of K values (with a minimum K of 2 and a maximum of L). We choose the K with best clustering score (typically silhouette) and label each frame with the cluster it belongs to. The second option is to run a variant of the backward messaging algorithm described below. We will compare the performance of both of these options in various scenarios in the next chapter.

Given $\boldsymbol{\kappa}$, $\tilde{\boldsymbol{\pi}}$, and $\boldsymbol{\Theta}$ along with the upper bound on the total number of states L, we can run a variant of the backward message passing algorithm to update the states as discussed in [8]. As mentioned in Section 2.3, auxiliary Bernoulli random variables, $\boldsymbol{W} = (w_0, w_1, \cdots, w_F)$, can be used to determine whether state transitions are from the normal Markov process or from the extra self–transition process controlled by $\boldsymbol{\kappa}$. For $t > 0$, we have

$$\begin{aligned} w_t &\sim \text{Bernoulli}(\boldsymbol{\kappa}_{z_{t-1}}) \\ z_t &\sim w_t \delta_{z_{t-1}} + (1 - w_t)\tilde{\boldsymbol{\pi}}_{z_{t-1}} \end{aligned}$$

which can be interpreted as first flipping a biased coin with probability of success given by the value of $\boldsymbol{\kappa}$ associated to the previous state. If the toss is successful, we stay in the same state with probability 1. If the toss fails, we use the Markovian process to transition to the new state (which also includes a possibility of self–transition). The first state, z_0, is handled by the backward messaging scheme alone since there is no prior state to transition from (and the value of w_0 is set to 0).

Aside from the first state ($t = 0$), we sample pairs (z_t, w_t) jointly with the following scheme. For states $s = 1, 2, \cdots, L$

$$\begin{aligned} (z_t = s, w_t = 0) &\sim \left(1 - \kappa_{z_{t-1}}\right) \cdot \tilde{\boldsymbol{\pi}}_{z_{t-1},s} \cdot \mathbb{P}(\boldsymbol{F}_t | \boldsymbol{\theta}_s) \cdot m_{t+1,t}(s), \\ (z_t = s, w_t = 1) &\sim \delta_{z_{t-1},s} \kappa_{z_{t-1}} \cdot \mathbb{P}(\boldsymbol{F}_t | \boldsymbol{\theta}_s) \cdot m_{t+1,t}(s). \end{aligned}$$

Notice that if $w_t = 1$, we must have $z_t = z_{t-1}$ with probability 1 (accounting for the Kronecker

delta in that case). For the likelihoods associated to the data, we assume each topological dimension is sampled from an independent Gaussian random variable.

$$\mathbb{P}(\boldsymbol{F}_t|\boldsymbol{\theta}_s) = \prod_{i=1}^{d} \mathbb{P}(P_{td}|\boldsymbol{\mu}_{sd}, \boldsymbol{\Sigma}_{sd})$$

Here $\boldsymbol{\theta}_s = ((\boldsymbol{\mu}_{sd}, \boldsymbol{\Sigma}_{sd}))_{i=1}^{d}$ is a collection of mean vectors and covariance matrices for each topological dimension under consideration. As it turns out, the likelihood computations are a major bottleneck for the computations. In order to speed up comparisons, we also attempt to use vector norms to give a notion of *persistence similarity* which should be computationally more tractable. That is, we replace the true likelihood with

$$\mathbb{P}(\boldsymbol{F}_t|\boldsymbol{\theta}_s) \longrightarrow \prod_{i=1}^{d} \frac{1}{\epsilon + \|P_{td} - \boldsymbol{\mu}_{sd}\|_p}$$

where p is the index of the vector norm (typically $p = 2$) and $\epsilon > 0$ is a small cutoff parameter to prevent infinite similarity. We compare the efficiency and accuracy of this choice in early experiments.

The numbers $m_{t+1,t}(s)$ are the backward messages from time step $t+1$ to t. These are given recursively by $m_{F+1,F}(s) = 1$ and

$$\begin{aligned} m_{t+1,t}(s) = &\sum_{j=1}^{L} \mathbb{P}(z_{t+1} = j|\boldsymbol{\pi}, z_t = s, w_{t+1} = 0)\mathbb{P}(w_{t+1} = 0|z_t = s, \boldsymbol{\kappa})\mathbb{P}(\boldsymbol{F}_{t+1}|\boldsymbol{\theta}_j)m_{t+2,t+1}(j) \\ &+ \sum_{j=1}^{L} \mathbb{P}(z_{t+1} = j|\boldsymbol{\pi}, z_t = s, w_{t+1} = 1)\mathbb{P}(w_{t+1} = 1|z_t = s, \boldsymbol{\kappa})\mathbb{P}(\boldsymbol{F}_{t+1}|\boldsymbol{\theta}_j)m_{t+2,t+1}(j) \end{aligned}$$

for all $t < F$.

One issue with directly implementing the backward message pass is that the products of probabilities in $m_{t+1,t}(s)$ can lead to underflow for very long sequences. There are a number of ways to remedy this situation, but the easiest one is to rescale all backward messages at the same time step so that they are of size one. Since this scales all probabilities for (z_t, w_t)

by the same quantity, it will not impact the distribution of the states.

$$\begin{aligned}
\widetilde{m}_{F+1,F}(s) &= m_{F+1,F}(s) = 1 \\
\widetilde{m}_{t+1,t}(s) &= \sum_{j=1}^{L}\sum_{w=0}^{1} \mathbb{P}(z_{t+1}=j|\boldsymbol{\pi}, z_t = s, w_{t+1}=w) \\
&\qquad \cdot \mathbb{P}(w_{t+1}=w|z_t=s,\boldsymbol{\kappa})\mathbb{P}(\boldsymbol{F}_{t+1}|\boldsymbol{\theta}_j)\widehat{m}_{t+2,t+1}(j) \\
G_{F+1,F} &= L \\
G_{t+1,t} &= \sum_{\ell=1}^{L} \widetilde{m}_{t+1,t}(\ell) \\
\widehat{m}_{F+1,F}(s) &= \frac{1}{L} \\
\widehat{m}_{t+1,t}(s) &= \frac{\widetilde{m}_{t+1,t}(s)}{G_{t+1,t}}
\end{aligned}$$

By induction, we can show that

$$m_{t+1,t}(s) = \left(\prod_{\tau=t+1}^{T} G_{\tau+1,\tau}\right)\widetilde{m}_{t+1,t}(s) = \left(\prod_{\tau=t}^{T} G_{\tau+1,\tau}\right)\widehat{m}_{t+1,t}(s).$$

Since the product of G factors is constant for fixed t, we can drop them altogether and use $\widehat{m}_{t+1,t}$ in place of $m_{t+1,t}$ in the backward message algorithm which alleviates the underflow issue.

4.4 Updating the Self–Transition Parameters

Following [9], the vector $\boldsymbol{\kappa} = (\kappa_1, \kappa_2, \cdots \kappa_L)$ of self–transition probabilities is chosen initially by sampling the components independently according to

$$\kappa_s \sim \text{Beta}(\rho_1, \rho_2).$$

This Beta random variable samples a random probability with expected values of

$$\mathbb{E}[\kappa_s] = \frac{\rho_1}{\rho_1 + \rho_2},$$

and so we can think of ρ_1 and ρ_2 as initial guesses on the relative number of self–transitions versus Markov transitions, respectively. The initial choices for these hyperparameters are discussed in Section 4.7. The posterior distribution on $\boldsymbol{\kappa}$ given the auxiliary variables $\{w_t\}$ is just

$$\kappa_s = \text{Beta}\left(\rho_1 + \sum_{\tau, z_{\tau-1}=s} w_\tau,\ \rho_2 + \sum_{\tau, z_{\tau-1}=s} (1 - w_\tau)\right)$$

since we only need to augment ρ_1 and ρ_2 with the actual counts of these transition types (which is precisely what w_t indicates).

4.5 Updating the Markov Parameters

We initialize the vector $\boldsymbol{\beta}$ via

$$\boldsymbol{\beta} \sim \text{Dirichlet}\underbrace{\left(\frac{\gamma}{L}, \frac{\gamma}{L}, \cdots, \frac{\gamma}{L}\right)}_{L}$$

where L is the upper bound on the number of states and γ is a hyperparameter initially set as discussed in Section 4.7. To find the posterior, we first sample the $L \times L$ auxiliary matrix $\boldsymbol{M}$. The process for sampling this matrix is a variant of the Chinese Restaurant Franchise formulation of Hierarchical Dirichlet Processes proposed by Teh et al. in [7].

Let the $L \times L$ matrix $\boldsymbol{N}$ count the number of transitions between states from the Markov process; that is, n_{ij} counts the number of transitions from state i to state j. For self–transitions, we only increment n_{ii} if the corresponding auxiliary variable $w_t = 0$ (as otherwise the self–transition is from the process controlled by $\boldsymbol{\kappa}$). For each pair of states (i, j), we sample m_{ij} by the following scheme. Initialize m_{ij} to 0 and set a counter $c = 0$. For each

$\ell = 1, \cdots, n_{ij}$ sample a Bernoulli random variable x by

$$x \sim \text{Bernoulli}\left(\frac{\alpha\beta_j}{c + \alpha\beta_j}\right).$$

Increment c, and if $x = 1$, increment m_{ij}. Let $m_{\bullet j}$ be the sum of M over the entries in column j.

$$m_{\bullet j} = \sum_{s=1}^{L} m_{sj}$$

We can then update the vector $\boldsymbol{\beta}$ via

$$\boldsymbol{\beta} \sim \text{Dirichlet}\left(\frac{\gamma}{L} + m_{\bullet 1}, \frac{\gamma}{L} + m_{\bullet 2}, \cdots, \frac{\gamma}{L} + m_{\bullet L}\right)$$

Finally, after updating $\boldsymbol{\beta}$, we can update the i–th row of the Markov transition matrix by

$$\tilde{\boldsymbol{\pi}}_i \sim \text{Dirichlet}\left(\alpha\beta_1 + n_{i1}, \alpha\beta_2 + n_{i2}, \cdots, \alpha\beta_L + n_{iL}\right).$$

4.6 Updating the Emission Parameters

Since we are assuming the data for each topological dimension are generated from a multivariate Gaussian distribution, we need to update the mean vector $\boldsymbol{\mu}_{sd}$ and the covariance matrix $\boldsymbol{\Sigma}_{sd}$ for each state s and each topological dimension d. To make the discussion more tractable, we focus only on one topological dimension (as the process is similar for each).

Following Fox et al. [8, §C.4], if we place an Inverse–Wishart prior on $\boldsymbol{\Sigma}_s$ and an associated normal distribution on $\boldsymbol{\mu}_s$, then the posterior distributions will be the same by Bayesian conjugacy. This requires us to select a degree of freedom $\nu > n + 1$ where n is the dimension of the data and an $n \times n$ scale matrix $\boldsymbol{\Delta}$. There are various ways to select these initially, but the obvious choices are to set $\nu = n + 2$ (the minimum integer ν allowed) and $\boldsymbol{\Delta}$ equal to the overall covariance of the data.

Assuming such priors, let $\mathcal{P}_s$ be the collection of emissions associated to state s

$$\mathcal{P}_s = \{P_{td} | z_t = s\},$$

and $\boldsymbol{\theta}_s^{\text{old}} = \left(\boldsymbol{\mu}_s^{\text{old}}, \boldsymbol{\Sigma}_s^{\text{old}}\right)$ be the prior emission parameters. We first draw a sample of the new covariance matrix by the following scheme.

$$\begin{aligned}
\bar{\nu}_s &= \nu + |\mathcal{P}_s| \\
\bar{\nu}_s \bar{\boldsymbol{\Delta}}_s &= \nu \boldsymbol{\Delta} + \sum_{\mathcal{P}_s} \left(P_{td} - \boldsymbol{\mu}_s^{\text{old}}\right) \left(P_{td} - \boldsymbol{\mu}_s^{\text{old}}\right)^T \\
\boldsymbol{\Sigma}_s &\sim \mathcal{IW}(\bar{\nu}_s, \bar{\nu}_s \bar{\boldsymbol{\Delta}}_s)
\end{aligned}$$

Once we have updated the covariance matrix, we can update the mean.

$$\begin{aligned}
\bar{\boldsymbol{\Sigma}}_s &= \left(\boldsymbol{\Sigma}_s^{\text{old},-1} + |\mathcal{P}_s| \boldsymbol{\Sigma}_s^{-1}\right)^{-1} \\
\bar{\boldsymbol{\mu}}_s &= \bar{\boldsymbol{\Sigma}}_s \left(\boldsymbol{\Sigma}_s^{\text{old},-1} \boldsymbol{\mu}_s^{\text{old}} + \boldsymbol{\Sigma}_s^{-1} \sum_{\mathcal{P}_s} P_{td}\right) \\
\boldsymbol{\mu}_s &\sim \mathcal{N}\left(\bar{\boldsymbol{\mu}}_s, \bar{\boldsymbol{\Sigma}}_s\right)
\end{aligned}$$

An important issue arises when using the Inverse–Wishart distribution; namely, the scale matrix must be positive definite. While this is theoretically true for the formulas above, many of the matrices end up being nearly singular (i.e. they have at least one eigenvalue very close to zero). Due to constraints on machine precision, this can cause the Inverse–Wishart function to fail on occasion. The simplest fix is to check for positive definiteness before calling the distribution, and if this check fails, add some small parameter $\epsilon > 0$ to the main diagonal. In practice, we can set a small ϵ as a hyperparameter and increment the main diagonal until the scale matrix passes the positive definiteness check.

4.7 Updating the Hyperparameters

We update each of the hyperparameters α, γ, ρ_1, and ρ_2 according to the steps outlined in [9, §C.1].

4.7.1 Updating α

We initially choose α using

$$\alpha \sim \text{Gamma}(a, 1/b)$$

where the parameters are further hyperparameters. We simply choose $a = 1$ and $b = 0.01$ since the update process should ultimately converge to a relevant value of α for the model. This gives a fairly broad distribution with a mode of 0 but an expected value of 100.

The posterior of α is also a Gamma random variable with updated parameters. Given the matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ from updating the Markovian parameters, we have the following sampling scheme.

$$\begin{aligned}
n_{k\bullet} &= \sum_{\ell=1}^{L} n_{k\ell} \\
s_k &\sim \text{Bernoulli}\left(\frac{n_{k\bullet}}{n_{k\bullet} + \alpha^{\text{old}}}\right) \\
r_k &\sim \text{Beta}(\alpha^{\text{old}} + 1, n_{k\bullet}) \\
\tilde{a} &= a + \sum_{i,j} m_{ij} - \sum_{\ell=1}^{L} s_\ell \\
\tilde{b} &= b - \sum_{\ell=1}^{L} \ln(r_\ell) \\
\alpha &\sim \text{Gamma}(\tilde{a}, \tilde{b})
\end{aligned}$$

4.7.2 Updating γ

We initially choose γ using

$$\gamma \sim \text{Gamma}(c, 1/d).$$

We choose values $c = 2$ and $d = 1$ and allow the model to converge to an appropriate value through the update process. This Gamma distribution is much tighter than the one for α with a mode of 1 and an expected value of 2. Given the matrix $\boldsymbol{M}$ from updating the Markovian parameters (and a small parameter $\epsilon > 0$), we have the following sampling scheme.

$$\begin{aligned}
\zeta &\sim \text{Beta}\left(\gamma^{\text{old}} + 1,\ \sum_{i,j} m_{ij}\right) \\
p_M &= \frac{c + L - 1}{c + L - 1 + \sum_{i,j} m_{ij}(d - \ln(\zeta + \epsilon))} \\
x &\sim \text{Bernoulli}(p_M) \\
\gamma &\sim \begin{cases} \text{Gamma}\left(c + L,\ 1/(d - \ln(\zeta + \epsilon))\right), & x = 1 \\ \text{Gamma}\left(c + L - 1,\ 1/(d - \ln(\zeta + \epsilon))\right), & x = 0 \end{cases}
\end{aligned}$$

4.7.3 Updating ρ_1 and ρ_2

The hyperparameters for $\boldsymbol{\kappa}$ are chosen using auxiliary variables ϕ and η.

$$\begin{aligned}
\rho_1 &= \frac{\phi}{\eta^3} \\
\rho_2 &= \frac{1 - \phi}{\eta^3}
\end{aligned}$$

Initially, $\phi \sim \text{Uniform}(0, 1)$ and $\eta \sim \text{Uniform}(0, 2)$. Updating these auxiliary variables involves numerically estimating their posterior distribution over a grid on the rectangle $[0, 1] \times [0, 2]$ (since their Bayesian conjugate is not a well–known distribution). For each pair (ϕ_i, η_j) in the parameter space, we compute the corresponding values $(\rho_{1,ij}, \rho_{2,ij})$ and

the quantities

$$
\begin{aligned}
W_k^+ &= \sum_{t:z_t=k} w_t, \\
W_k^- &= \sum_{t:z_t=k} (1-w_t), \\
\Gamma_{ij} &= L \ln\left(\frac{\Gamma(\rho_{1,ij}+\rho_{2,ij})}{\Gamma(\rho_{1,ij})+\Gamma(\rho_{2,ij})}\right) + \sum_{\ell=1}^{L} \ln\Gamma\left(\rho_{1,ij}+W_\ell^+\right) + \sum_{\ell=1}^{L} \ln\Gamma\left(\rho_{2,ij}+W_\ell^-\right) \\
&\qquad - \sum_{\ell=1}^{L} \ln\Gamma\left(\rho_{1,ij}+\rho_{2,ij}+W_\ell^+ + W_\ell^-\right).
\end{aligned}
$$

Letting $\Gamma_{\max} = \max\{\Gamma_{ij}\}$, we compute the probabilities associated to the grid element (i,j) using softmax

$$p_{ij} = \frac{e^{\Gamma_{ij}-\Gamma_{\max}}}{\sum_{k,\ell} e^{\Gamma_{k\ell}-\Gamma_{\max}}},$$

and we pick a sample from the grid based on these probabilities. Note that subtracting $\Gamma_{\max}$ is not necessary, but it often helps to avoid overflow errors. Once the pair $(\phi_{\text{new}}, \eta_{\text{new}})$ have been sampled, we convert these values to ρ_1 and ρ_2 as above.

4.8 Log–Likelihood for the dsHDP–HMM

The overall log–likelihood function for the process considers all possible pairings of the data with the identified classes and their emission parameters along with the probabilities of state transitions. We compute two main quantities recursively, the auxiliary matrix $\boldsymbol{A}$ of size $(F+1)\times L$ (where F is the total number of frames in the dataset) and the F–dimensional vector $\boldsymbol{C}$ (which are the normalization factors for each row of $\boldsymbol{A}$). For the first row of $\boldsymbol{A}$, we set

$$\boldsymbol{A}_{1s} = \mathbb{P}\left(\boldsymbol{F}_1 | \boldsymbol{\theta}_s\right) \qquad (1 \leqslant s \leqslant L)$$

which are just the likelihoods that the first datum belongs to the various classes. For rows $2 \leqslant t \leqslant F+1$, we set

$$\begin{aligned}
\tilde{\boldsymbol{A}}_{ts} &= \sum_{\ell=1}^{L} \boldsymbol{A}_{(t-1)\ell} \cdot \boldsymbol{\pi}_{\ell s} \cdot \mathbb{P}\left(\boldsymbol{F}_t | \boldsymbol{\theta}_s\right), \\
\boldsymbol{C}_t &= \sum_{\ell=1}^{L} \tilde{\boldsymbol{A}}_{t\ell}, \\
\boldsymbol{A}_{ts} &= \tilde{\boldsymbol{A}}_{ts} / \boldsymbol{C}_t.
\end{aligned}$$

Effectively, each row computes the likelihood of a given transition from all possible previous states to all possible next states using the learned emission parameters and transition matrix. Note that $\boldsymbol{\pi}_{s\ell}$ is the *full* transition matrix (including the self–transition mechanism)

$$\boldsymbol{\pi}_{s\ell} = \kappa_s \delta_{s,\ell} + (1 - \kappa_s) \widetilde{\boldsymbol{\pi}}_{s\ell}.$$

The overall log–likelihood of the model is given by

$$\ln \mathcal{L}(\boldsymbol{F}) = \frac{1}{F} \sum_{t=1}^{F} \ln \boldsymbol{C}_t.$$

4.9 State Matching and Change Point Comparison

Since this study is primarily interested in change point detection, it is not necessary to match actual and predicted state labels (as the changes in state are sufficient to identify change points). Also, HDP–HMMte is completely unsupervised as the known labels are never used during training; this means the learned state labels are completely arbitrary. However, matching known and predicted states as closely as possible facilitates graphical interpretation of the results.

In order to match states, the Kuhn–Munkres Algorithm (or Hungarian Algorithm) is employed at the end of the training phase. [14] This algorithm is designed to find the

optimal matching between vertices in a complete bipartite graph. Assuming there are an equal number of vertices in both the left and right sets, the brute force approach is to check all $n!$ pairings between the sets of vertices and select the one with the least total edge weight. Kuhn–Munkres solves the problem with an algorithm that runs in $\mathcal{O}(n^3)$ time.[15] The algorithm begins with the matrix of edge weights between left and right vertices and then constructs an initial partial match via a greedy approach. This partial matching is then expanded into a full matching through a series of augmentations and reassignments. To use the algorithm, we only need to supply the weight matrix $\boldsymbol{W}$ which for our purposes is given by

$$\boldsymbol{W}_{ij} = \left|\#\{\text{frames with true state} = i\} - \#\{\text{frames with predicted state} = j\}\right|.$$

The algorithm returns the matching between true and predicted states that minimizes the number of mismatches. This matching is turned into a dictionary which is then used to relabel all states, transition probabilities, and emission parameters before the model is run on the test data.

The primary comparison we need to make is between actual and predicted change points (i.e. frames where a change of state occurs). A modified greedy algorithm is employed to match these sets of change points. First, the matrix of absolute differences between true and predicted change points is computed, and the elements of the matrix are chosen from smallest to largest until either the list of actual or predicted change points has been exhausted. While this approach is the obvious one, it can run into a temporal anomaly.

As a simple example, suppose the actual change points are $T_a = \{1, 5, 10, 24\}$ while the predicted ones are $T_p = \{2, 8, 13, 24\}$. The matrix of absolute differences is given as follows.

	2	8	13	24
1	1	7	12	23
5	3	3	8	19
10	8	2	3	14
24	10	16	11	0

Greedily choosing the smallest entries gives the following matching of change points.

Actual	Predicted	Difference
1	2	1
5	13	8
10	8	2
24	24	0

Clearly, we should not match the actual change point of 5 with the predicted change point of 13 since this comes after our prediction of 8 for the true change point at 10. The algorithm proceeds by first performing the greedy matching as above. It then searches for and removes these temporal inconsistencies. Any extra true change points are marked as missed, and any remaining predicted change points are listed as extra. In addition to giving an average discrepancy between matched change points, we can also compute recall and precision as follows.

$$\text{recall} = \frac{\#\{\text{matched change points}\}}{\#\{\text{true change points}\}}$$
$$\text{precision} = \frac{\#\{\text{matched change points}\}}{\#\{\text{predicted change points}\}}$$

5 Experiments with Synthetic Data

In this chapter, we benchmark the Hierarchical Dirichlet Process Hidden Markov Model with Topological Emissions (HDP–HMMte) algorithm against three different scenarios. The first set of trials use two–dimensional time series data where there is a sharp change in topology at known times. These experiments effectively serve as a proof of concept for the method. The second set of experiments examines the ability of the method to detect changes in variance of point sets in three dimensions. Finally, HDP–HMMte is tasked with finding parameter changes in a dynamical system resulting in a significant change in the underlying topology.

5.1 Two–Dimensional Time Series Data with Distinct Changes in Topology

For the first experiment with the HDP-HMMte, we look at time series where the points are chosen to lie around curves with either 1, 2, or 3 loops (with a small noise component added). Sample images are shown in Figure 9. Both the training and test sets consist of point clouds chosen randomly according to one of these three basic shapes. Each state has a self–transition probability of at least 80% and roughly equal probabilities of switching to either of the two other states. Since this should be the ideal type of data for the method, we use these experiments to test the more important aspects of the model.

First, we examine the performance of the model using the standard Gaussian likelihood to compare states versus using similarity (i.e. truncated inverse distance) of the persistence images interpreted as vectors. That is, we consider the effect of the replacement

$$\prod_{i=1}^{d} \mathbb{P}(P_{t,d} | \boldsymbol{\mu}_{s,d}, \boldsymbol{\Sigma}_{s,d}) \longrightarrow \prod_{i=1}^{d} \frac{1}{\epsilon + \|P_{t,d} - \boldsymbol{\mu}_{s,d}\|_2}$$

on the learning rate and accuracy. Four trials were conducted for each choice; two of the

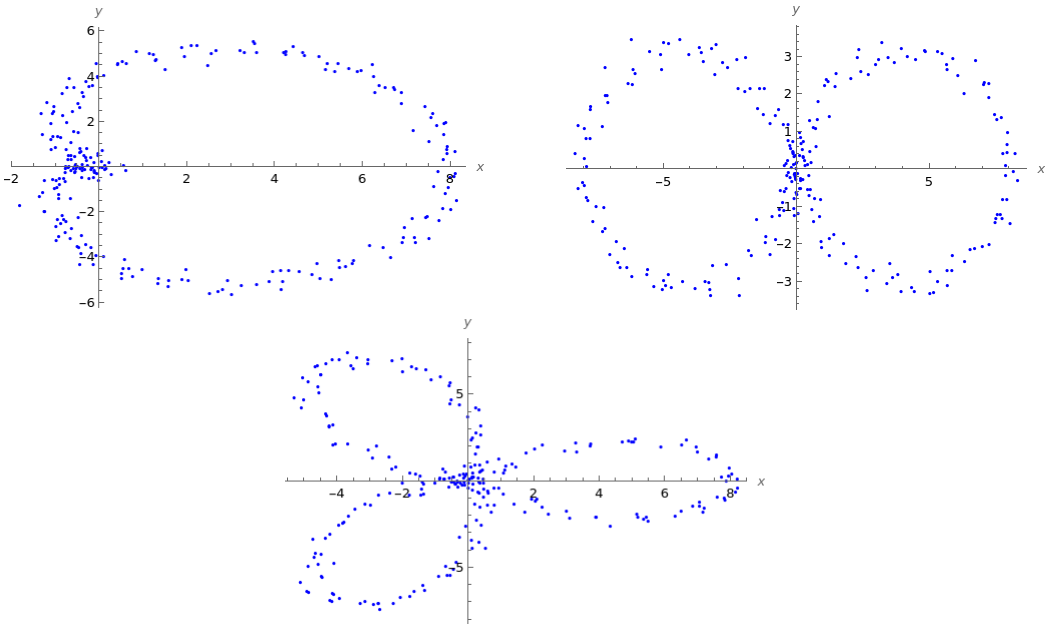


Figure 9: Sample Time Series Data Showing Different Numbers of Loops

trials used the raw persistence images while the other two used Z–scored images. All trials used K–Means to initialize the states, and training was run for 2000 iterations with the hyperparameters initialized as in Section 4.7.

In all of these trials, similarity gave a clear advantage over Gaussian likelihood. The statistics for the two trials using raw persistence images are shown below. For brevity, only the F1–Scores for change point detection and the overall state prediction accuracy are shown. As can be seen in the plots of predicted versus actual states on the training data,

	Likelihood				**Similarity**			
	Train		Test		Train		Test	
	CP F1	State Acc.	CP F1	State Acc.	CP F1	State Acc.	CP F1	State Acc.
Trial 1	15.82%	50.90%	88.00%	15.63%	88.89%	97.80%	100.0%	98.20%
Trial 2	14.36%	60.32%	23.73%	65.93%	100.0%	98.80%	92.31%	98.20%

Figure 10: Gaussian Likelihood vs. Similarity for Raw Data

the Gaussian likelihood actually finds all true change points, but it finds so many additional changes of states that the true predictions are overwhelmed (leading to the low F1–Score). The results were similar when the persistence images were Z–scored before both training and testing. Both Gaussian likelihood and similarity should be largely insensitive to a shift

(a) Gaussian Likelihood (trial 2)

(b) Similarity (trial 2)

Figure 11: State Comparison for Raw Training Data

of mean (since they involve difference of vectors). Rescaling the data could be useful in certain circumstances, but none of the trials in this thesis seemed to benefit significantly from Z–scoring.

Looking at the time spent exploring various numbers of states, the models trained using

	Likelihood				**Similarity**			
	Train		Test		Train		Test	
	CP F1	State Acc.	CP F1	State Acc.	CP F1	State Acc.	CP F1	State Acc.
Trial 1	48.28%	92.59%	66.67%	96.59%	96.30%	98.20%	92.31%	98.00%
Trial 2	10.94%	42.89%	12.33%	45.89%	96.30%	98.20%	98.60%	92.31%

Figure 12: Gaussian Likelihood vs. Similarity for Z–Scored Data

similarity spent 77.00% of all iterations with 3 occupied states, 21.84% with 4 occupied states, 1.05% with 5 occupied states, and only about 0.1% of iterations in any other occupancy number. Using Gaussian likelihood, the model spent roughly equal numbers of iterations with occupancies of between 9 and 13 states and spent very little time with models having close to 3 states. All of this seems to point to the conclusion that using similarity helps the algorithm converge to better models far faster than Gaussian likelihood. As trial 1 for the Z–scored data shows, Gaussian likelihood can reach a decent model rather quickly, but the number of iterations needed to hone in on a better portion of the parameter space could be quite large.

Not only were the results far better using similarity, the overall run times were significantly improved. Over the four runs using Gaussian likelihood, the average time to complete the 2000 iterations for training was just under 45 minutes (meaning each iteration required 1.34 seconds on average). Using similarity, the average run time dropped to roughly 3.5 minutes (with each iteration taking about 0.11 seconds on average) – roughly a 12 fold increase in speed. Given these results, similarity of the persistence images was used for all trials moving forward.

The next set of trials were designed to determine the impact of state initialization methods on the final results. The first run used K–Means to set the initial states with a maximum value of K set to 15. The value of K with the best silhouette score was used to initialize the states. The second run was initialized using a variant of the Backward Message Passing algorithm. As before, training was limited to 2000 iterations. The results are summarized in Figures 13 and 14. Looking at the results, there is no clear reason to choose one initialization

method over another. Both led to a trained model with exactly the same performance on the test data. And neither seems to offer a massive speed–up in convergence as the maximum log-likelihoods were found after roughly the same number of iterations.

From these figures, it is also clear that HDP–HMMte found the majority of the change points in both training and test sets and predicted those changes very close to the correct times. In fact, the average discrepancy between a found change point and the actual change point for the training data was roughly 0.5 ± 0.5 frames (meaning most change points were detected within 1 frame of the actual change). For the test data, the accuracy was similar (0.6 ± 0.5 frames).

Init. Method	Recall	Precision	F1	Likelihood	Iteration
K–Means	100.0%	87.50%	93.33%	2.1481	970
Backward Msgs.	100.0%	100.0%	100.0%	2.4044	969

(a) Summary Statistics

(b) Initialized by K–Means

(c) Initialized by Backward Message Passing

Figure 13: Initialization Comparison for Loop Training Data

5.2 Detection of Variance Changes in Time Series Data

For the second set of experiments, times series data were generated by randomly generating points in three dimensions using a spherical Gaussian distribution with mean $(0, 0, 0)$

Init. Method	Recall	Precision	F1
K–Means	85.71%	100.0%	92.31%
Backward Msgs.	85.71%	100.0%	92.31%

(a) Summary Statistics

(b) Initialized by K–Means

(c) Initialized by Backward Message Passing

Figure 14: Initialization Comparison for Loop Test Data

and different variances (that is, with covariance matrices of the form $\Sigma = \sigma^2 I_3$ for different choices of σ). Two sets of experiments were run using slightly different sets of variances. In all cases, the window size was set to 20 data points, but the sliding rate was varied between 5, 10, and 20 and points to investigate the effect of overlap on change point detection. The

State	St. Deviation
1	0.25
2	4
3	2
4	1

(a) Variance Experiment 1

State	St. Deviation
1	0.5
2	1
3	2

(b) Variance Experiment 2

Figure 15: States and Variances for Experiments

algorithm was run for 2000 iterations with the hyperparameters initialized as in Section 4.7.

The results of experiment 1 for the three different slide rates can be found in Figure 16. Clearly, the choice of slide rate seems mostly immaterial for the set of variances used. The difference between the slide rate 10 (the best overall F1–score) and slide rate 20 (the worst) on the test data is noticeable, but both models found all actual changes points (with minimal delay) while the model using a slide rate of 20 found one extraneous change point.

	Training Data			Test Data		
Slide Rate	Recall	Precision	F1	Recall	Precision	F1
20	100.0%	100.0%	100.0%	100.0%	83.33%	90.91%
10	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
5	100.0%	100.0%	100.0%	100.0%	90.91%	95.24%

Figure 16: Results for Variance Change Experiment 1

For the second experiment, we can see more variation in model performance versus the slide rate. In particular, having a slide rate of 5 seems to pick up a number of extraneous transitions in both the training and test sets as can be seen in Figure 19. It may well be that with a larger number of iterations, the model with slide rate of 5 could have found a better fit. At the very least, we can see that the choice of slide rate is a parameter that needs to be tuned for the time series data under consideration. If the rate is too small, the algorithm may pick up extraneous changes from the overlapping frames. On the other hand, a large slide rate might miss transition behavior that could be of interest. In later experiments, we largely keep the slide rate either equal to the window size or half the window size as these seem to perform well in many cases.

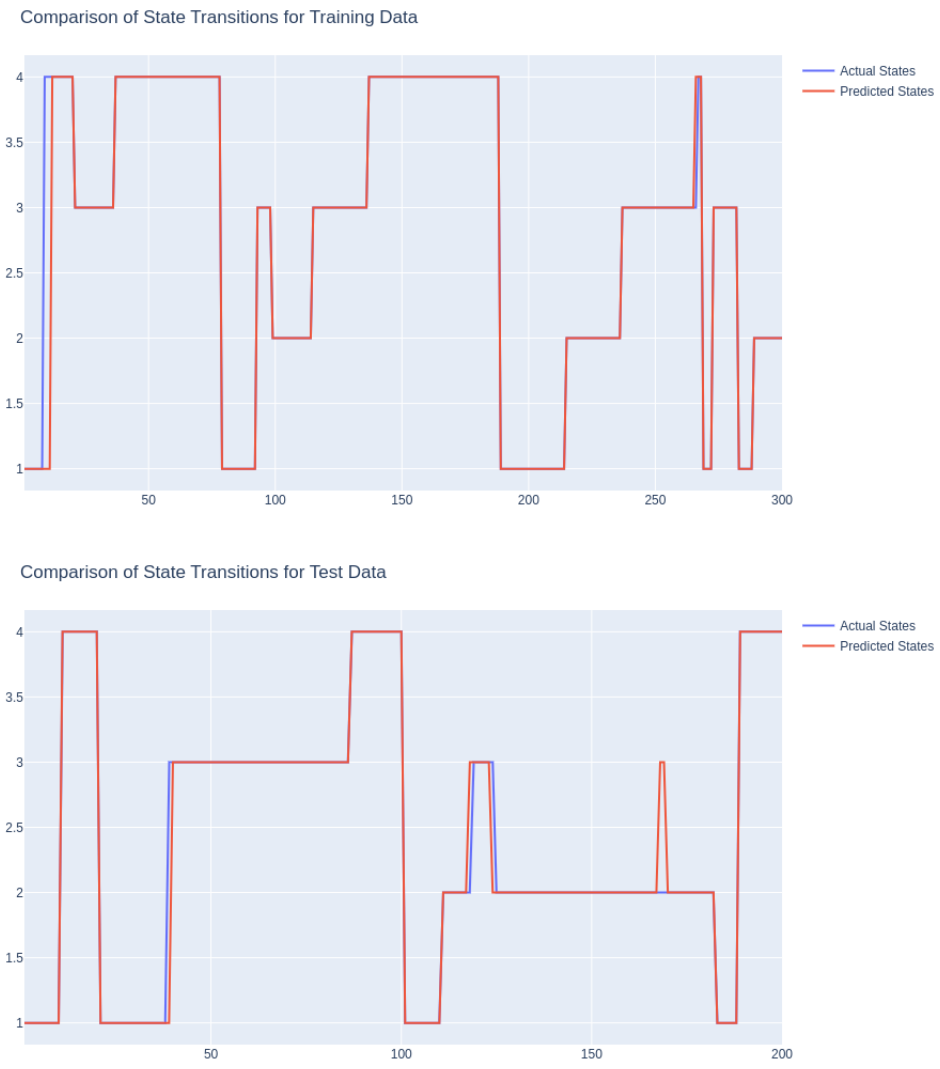


Figure 17: Results for Experiment 1 with Slide Rate 20

	Training Data			Test Data		
Slide Rate	Recall	Precision	F1	Recall	Precision	F1
20	91.67%	100.0%	95.65%	100.0%	100.0%	100.0%
10	100.0%	92.31%	96.0%	100.0%	92.31%	96.0%
5	100.0%	33.33%	50.0%	100.0%	75.00%	85.71%

Figure 18: Results for Variance Change Experiment 2

5.3 Parameter Changes in Dynamical Systems

In 1963, Edward Norton Lorenz introduced a set of coupled, nonlinear differential equations meant to be a simple model of fluid flow in the atmosphere.[16, §7.11] In terms of

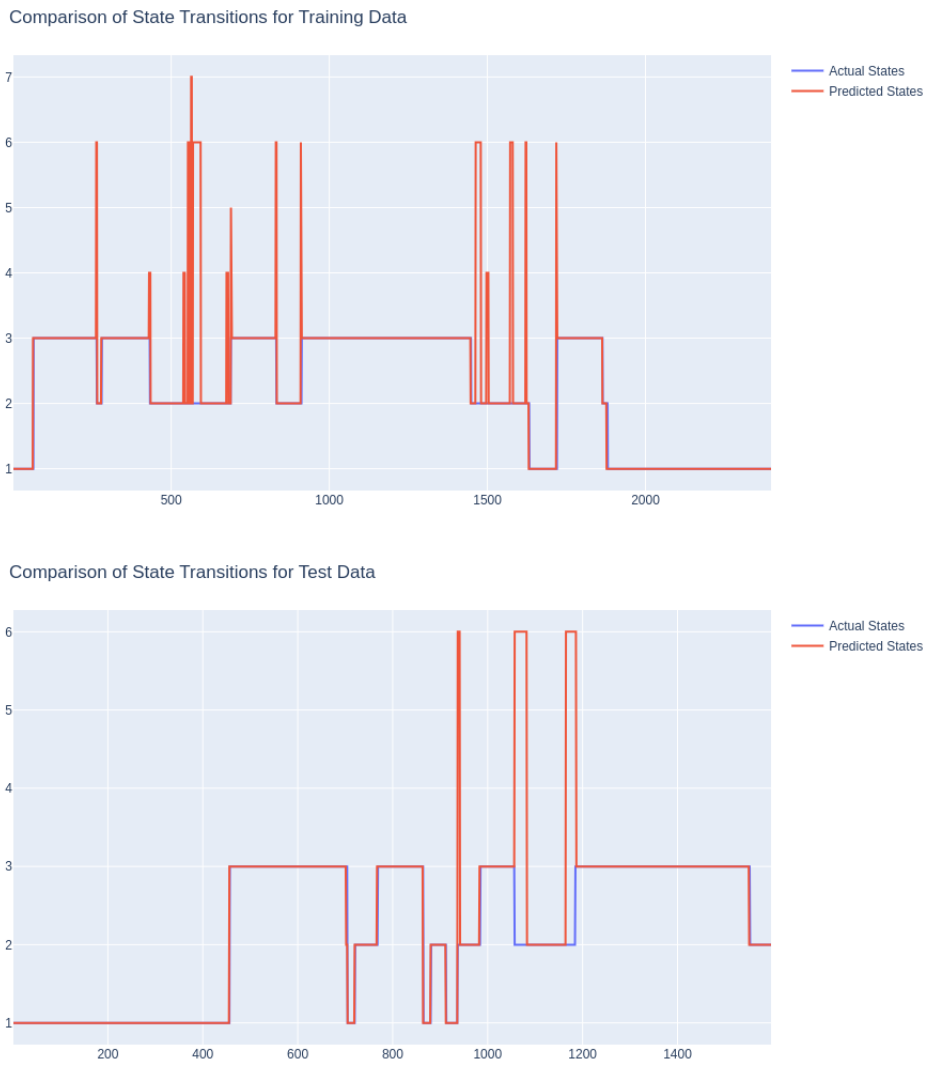


Figure 19: Results for Experiment 2 with Slide Rate 5

generic variables $x = x(t), y = y(t)$, and $z = z(t)$, the system is given by

$$\begin{aligned} \frac{dx}{dt} &= -\sigma x + \sigma y \\ \frac{dy}{dt} &= \rho x - y - xz \\ \frac{dz}{dt} &= -\beta z + xy \end{aligned}$$

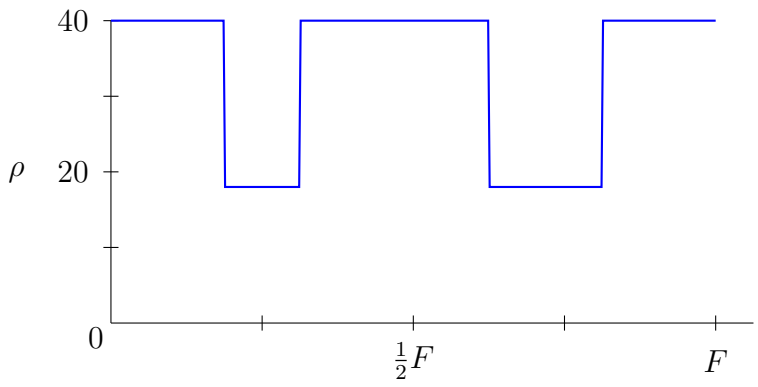
where β, ρ, and σ are parameters of the model. Lorenz discovered that the system displayed chaotic behavior for the choice of parameters $\beta = 8/3, \rho = 28$, and $\sigma = 10$; that is, the

solutions to the system of differential equations above are extremely sensitive to small changes in initial data. This chaotic behavior persists for a wide range of the parameter values. Despite the sensitive dependence on initial conditions, the chaotic solutions to the Lorenz system all lie on a complicated (roughly) two–dimensional surface known as the *Lorenz Attractor*.[9]

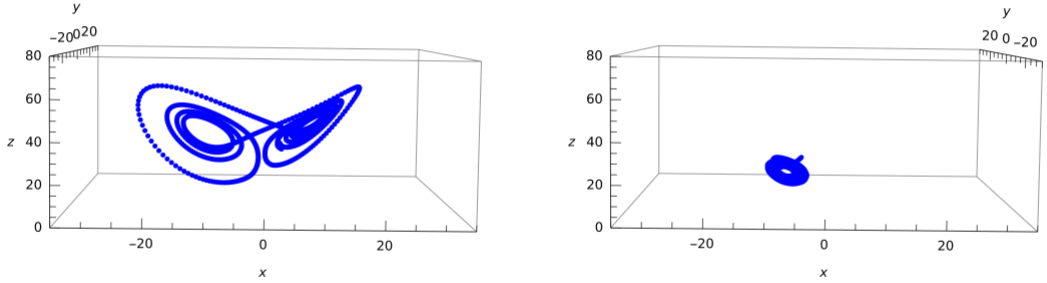
For this study, parameters $\beta = 8/3$ and $\sigma = 10$ were fixed while ρ was varied to produce changes in the topology of the solutions. For $14 \leqslant \rho < 470/19 \approx 24.74$, solutions to the Lorenz system will converge to one of two equilibrium values (and so no chaotic behavior is observed). For values of ρ larger than this critical value, the solutions travel around the Lorenz Attractor in complicated orbits. [16, pp. 310, 311] For these experiments, we start with $\rho = 40$ (the chaotic regime) and fairly rapidly (but continuously) lower the value to $\rho = 18$ (the stable regime). We repeat this cycle twice for the training data. A similar (single) cycle is used to test the HDP–HMMte model. The training data was divided into windows of 600 data points that moved in increments of 100 points per frame (giving 392 total frames for the training data). The chaotic regime was labeled as State 1, the stable regime State 3, and a brief transition region was labeled as State 2. K–Means was used to initialize the states, and training was allowed to run for 1000 iterations. As the results in Figure 21 show, HDP–HMMte easily finds the transitions between regimes for both the training and test datasets. In fact, the best log–likelihood for the run was found just after 3 iterations.

When the method was initialized without K–Means, the model settled on a slightly different interpretation of the data (with a slightly higher log–likelihood). It still reliably distinguishes the chaotic from the stable regime. However, this instance of the model splits the chaotic regime into two distinct topological varieties. Figure 23 shows the difference in topology between states 1 and 2 for the training data (near frame 19). Notice that the inner loops for the left portion of the Lorenz Attractor have momentarily disappeared for the

[9]The attractor is actually a fractal, making its precise dimensionality difficult to compute.



(a) ρ vs. Time (frame) for Training Data



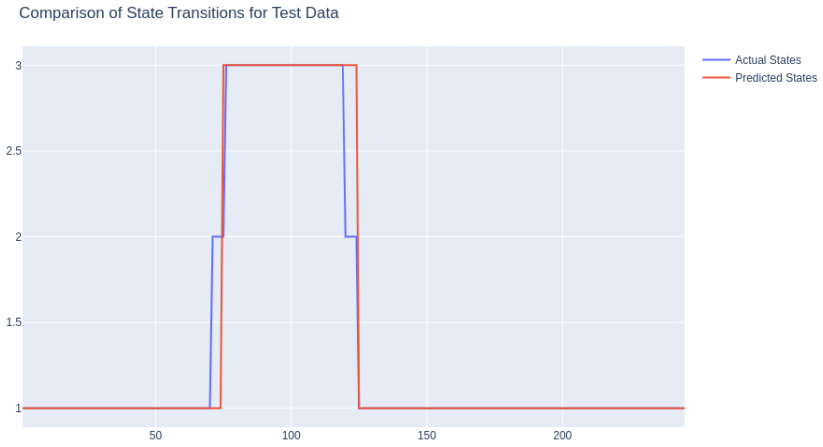
(b) Chaotic Regime

(c) Stable Regime

Figure 20: Parameter Change and Sample Orbits for the Lorenz Trials

frame corresponding to State 2 – primarily an artifact of the choice of window size and slide rate. This again points to the importance of choosing appropriate values for the window and slide parameters. If the window is too small (and so the number of points for the homology computations is relatively few), important features may be missed in the topological analysis. On the other hand, a very large window may well include topological features from distinct states of the system (muddling the change point detection). Similar considerations hold for the rate at which the window slides across the time series data.

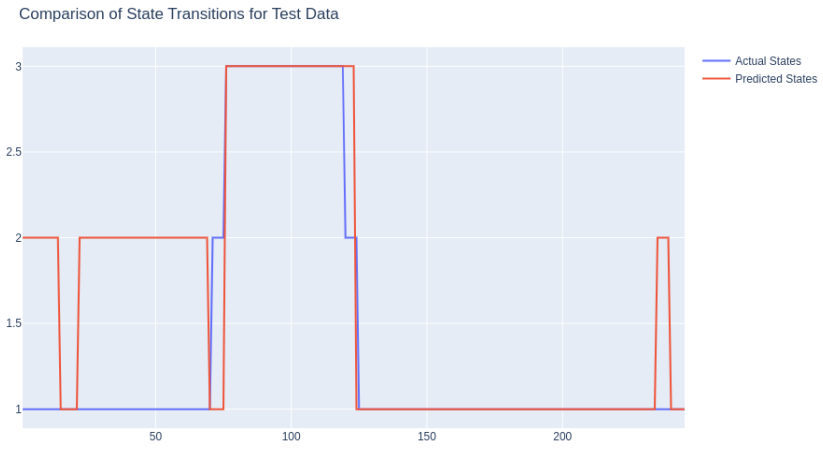
(a) Training Data



(b) Test Data

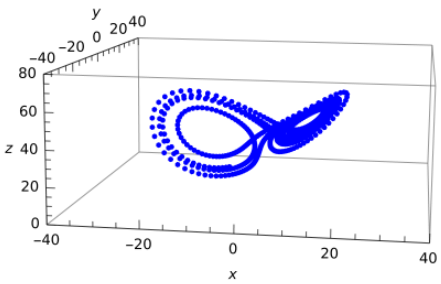
Figure 21: Results for Parameter Change in the Lorenz Dynamical System (initialized by K–Means)

(a) Training Data

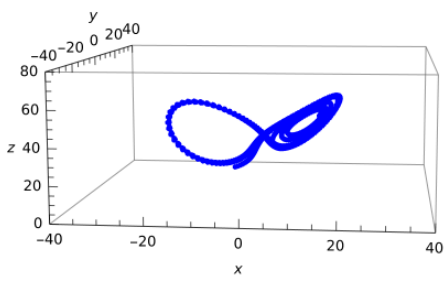


(b) Test Data

Figure 22: Results for Parameter Change in the Lorenz Dynamical System (initialized without K–Means)



(a) Sample State 1 from Training Data



(b) Sample State 2 from Training Data

Figure 23: Difference in Topology for States 1 and 2 (modeled initialized without K–Means)

6 Experiments with Real–World Data

In this chapter, we examine the efficacy of HDP–HMMte for change point detection using data collected from real experiments. This chapter collects results from three types of experiments conducted on two distinct types of machine learning tasks. The first task looks at human motion classification. For an initial experiment, we use data from the Human Activity Recognition (HAR) dataset to determine if topological data analysis can distinguish between different actions from the same person and similar actions from different people as recorded by cellphone accelerometer data. Due to the data limitations in the HAR dataset, the second experiment uses the Microsoft Research Cambridge–12 (MSRC–12) dataset which consists of multiple recordings of people performing various body movements as recorded by a network of twenty sensors. Finally, we turn to image classification using topological data analysis by considering the problem of detecting the onset of a solar flare from images of the sun's surface recorded by NASA's Solar Dynamics Observatory (SDO) spacecraft.

6.1 The Human Activity Recognition (HAR) Dataset

The Human Activity Recognition dataset [17] consists of observations of 30 volunteers performing six different activities while wearing a smartphone. Data from the phone's accelerometer and gyroscope were captured, giving a 6–dimensional vector for each measurement (three linear acceleration components and three angular velocity components). The researchers collecting the observations pre–processed them by applying noise filtering and sampling the smoothed data into windows of size 128 measurements (corresponding to 2.56 seconds of time) with a slide rate of 64. The researchers also created numerous additional features for the set by transforming the raw 128 measurements per window to an augmented 561 feature vector for each window. These extra features include discrete Fourier transforms and various numerical derivatives of the original data. For our purposes, we use the original 6–dimensional measurements (found in the Inertial Signals folder of the dataset). Since these

data are already windowed, we opt for a window size of 128 and a slide rate of 128 (as the overlap is already built into the data).

As is, the HAR dataset is not well-suited for HDP–HMMte since the participants were asked to perform the six activities in a fairly consistent order. In order to have a dataset of reasonable size and with a decent number of transitions between states, the experiments here were limited to three of the activities: Standing, Walking, and Walking Downstairs. In the original design, the dataset was filtered by activity label, and a random selection of the three selected activities were chosen (obviously, natural transitions between the activities were destroyed in the process). The initial runs found far more than three states, and a closer examination led to the conclusion that the method was able to distinguish between the different volunteers for some of the activities.

To test this idea, two experiments were run. First, a single volunteer (Subject 1) was chosen and their data for the three activities isolated. There were two separate measurements for each of the activities; standing had two segments each just over one minute long; walking had two segments each over two minutes long; walking downstairs had one segment just over one minute long and one just under 20 seconds. These segments were selected randomly and the corresponding time series data created. Unfortunately, the test data provided by HAR is for a completely separate set of volunteers. So, a short test set was constructed by randomly sampling from the same motions. This has fairly limited value except to check that the HDP–HMMte can genuinely distinguish between the activities (and is not memorizing a specific sequence of states). The HDP–HMM was trained with 1000 iterations using the usual hyperparameter initializations (as detailed in Section 4.7).

As can be seen in Figure 24, the HDP–HMMte model easily distinguishes the three different activities (identifying a total of 4 topologically distinct states). The recall on the training data is 100.0% with a precision of 83.33% (so an F1–score of 90.91%). The only extraneous state transitions were two brief jumps to a fourth state when the dataset transitioned between standing and walking downstairs (states 1 and 3). On average, all true

change points were detected within about 1 to 2 frames of the actual value. The performance on the test series was similarly accurate.

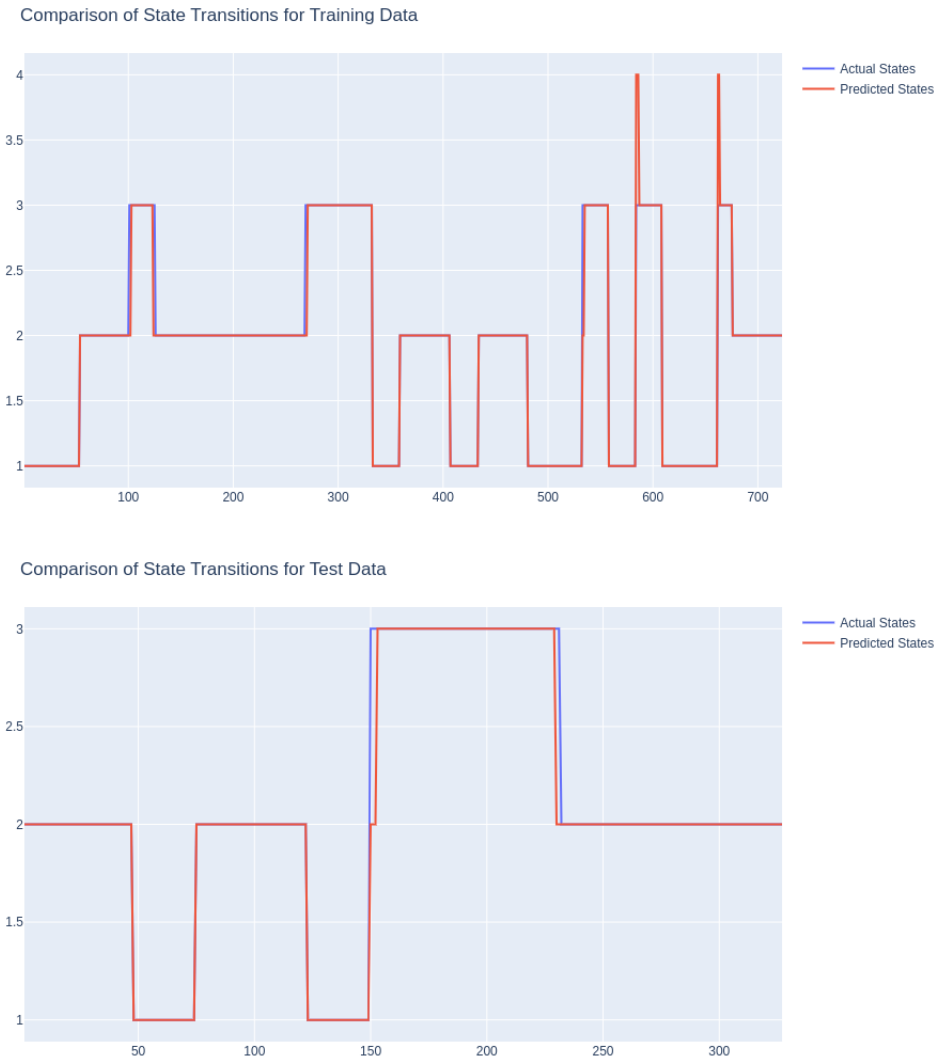


Figure 24: Results for Single Subject HAR Data

To test the idea that the HDP–HMMte is able to distinguish between different subjects, two subjects were chosen (Subjects 1 and 3) and their data for Standing, Walking, and Walking Downstairs were extracted and randomized as for the single subject trial. There are now six possible states for the model as cataloged in Figure 25.

On the training data, the model identified seven distinct states and had a change point recall of 100.0% but a precision of only 54.26% (giving an F1–score of 70.34%). All true

State	Subject	Activity
1	1	Standing
2	1	Walking
3	1	Walking Downstairs
4	3	Standing
5	3	Walking
6	3	Walking Downstairs

Figure 25: State Labels for Two Subject HAR Data

change points were discovered within an average of 0.9 ± 2.5 frames of their actual values. The results for the test set were similar. Diving into the state labels deeper showed that

Figure 26: Results for Two Subject HAR Data

the model was able to distinguish the two subjects 80.02% of the time on the training data (that is, it correctly identified the subject for 2844 out of the 3554 frames). Of the 710 misidentified frames, 414 of them labeled state 4 as state 1. That means that over 58% of the misidentifications were data derived from the subject standing in place (when presumably the accelerometer and gyroscope readings are close to zero). Looking in more detail, there were 1144 frames labeled standing (either state 1 or 4). A total of 729 of them were attributed to both the correct subject and correct activity. So despite the higher level of uncertainty, the model was able to distinguish the two different subjects 63.72% of the time even when they were standing still.

The extraneous seventh state found by the model was exclusively associated to the activity of walking downstairs. Among the states associated to this activity (totaling 661 frames in the training data), 62.33% of the states were correctly identified as the correct subject and activity, 25.72% were identified as state 7, and the remaining 11.95% were an assortment of 5 different misidentifications. If we associate state 7 with subject 1 (the more commonly associated subject), then the model is able to distinguish subjects walking downstairs 81.39% of the time. However, state 7 is only split about 60% to 40% in favor of subject 1. Presumably, the heavy shocks from walking down a flight of stairs are overwhelming the finer details allowing the model to distinguish between persons. If there were landings between flights, that might also explain the 26 frames mistakenly identified as walking (though with the correct subject identification).

6.2 The Microsoft Research Cambridge–12 (MSRC–12) Dataset

To explore human activity detection further, we turn to the Microsoft Research Cambridge 12 (MSRC–12) dataset [18] which consists of records of specific body motions from multiple participants. Each record is a time series of a single participant recording a specific motion repeatedly. The motion is captured by an array of 20 body sensors capturing position data (so each point in the time series is 60 dimensional). For this study, three specific

motions were selected: bowing, kicking, and throwing.

For the first trial, a single participant was selected (participant 28) and the recordings of the three motions were split into individual files based on the times when each motion began and ended. The window size was set at 90 data points since the average motion was just over 3 seconds long (and the recording rate was 30 Hz). Two sliding rates were considered: 90 data points (so that windows had no overlap) and 45 data points. All data were scaled and subjected to principal component analysis. For the single person trial, the full range of 60 PCA dimensions were retained. Roughly 20% of the motions were reserved for test data. Both training and test time series were generated by creating random sequences of the motions. The homology components were restricted to $\mathcal{H}_0$ and $\mathcal{H}_1$. Adding the $\mathcal{H}_2$ component was considered, but most windows had empty (or nearly empty) persistence images for this dimension.

As with the HAR dataset, the HDP–HMMte model is quite successful at distinguishing between the motions (successfully identifying 3 distinct states). Using a matching window size and slide rate of 90 data points, the model had a recall of 93.75%, a precision of 100.0%, and an F1–score of 96.77% for change points on the training data. For the change points correctly recalled, the change was found within 1 frame of the actual change point, on average. The results on the test data were reasonably similar: change point recall was 90.91%, precision 83.33%, and F1–score 86.96%. Correctly recalled change points were found within 6 frames of the true change (with an average of about 2 frames).

Looking at correct state prediction, we see that with the exception of a quick transition of the form $1 \to 3 \to 2$ near the end of the training data, the overall accuracy is quite good (just over 99%, in fact). The accuracy on the test data is only 93% overall with significant confusion between states 1 (bowing) and 2 (kicking). The confusion matrix for the test data is shown in Figure 28.

When we switch to a slide rate of 45 data points (so that successive windows overlap by half), we get slightly worse performance. The model found four states in total and had a

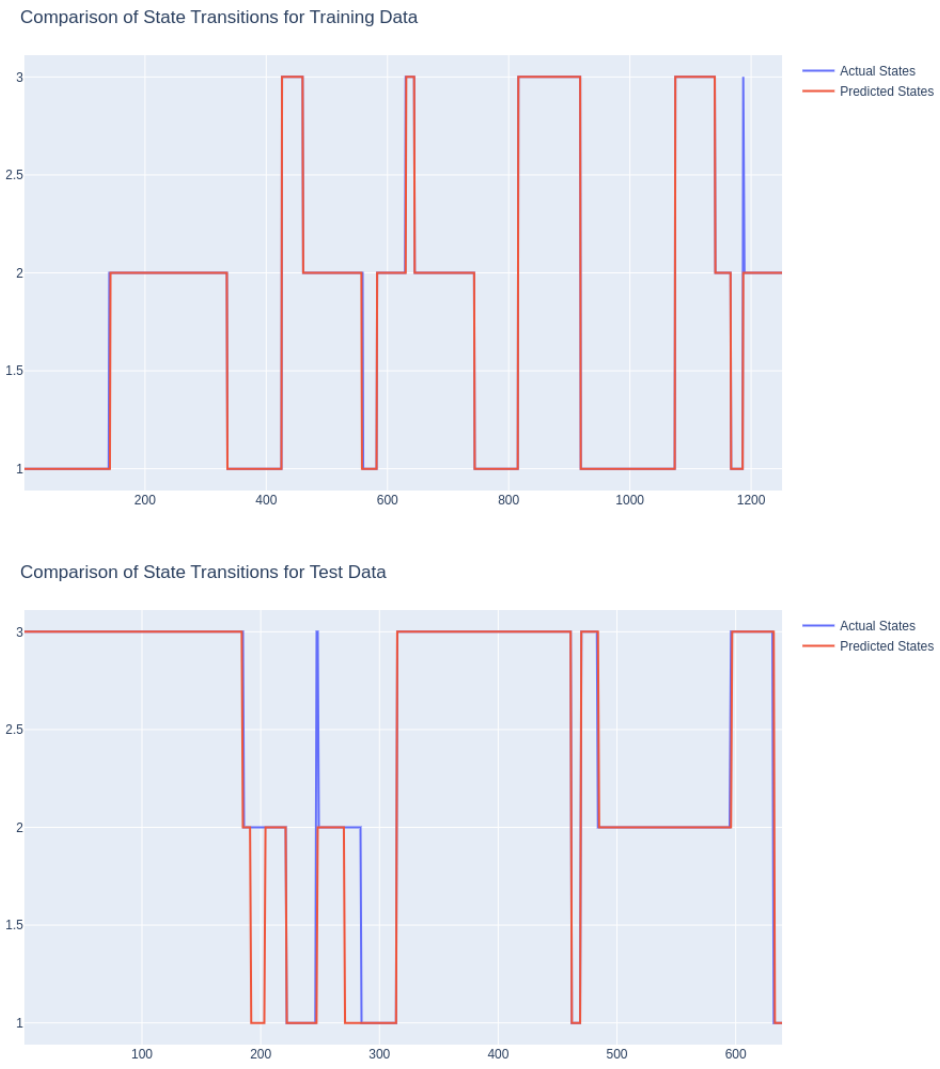


Figure 27: Results for Single Subject MSRC–12 Data (window = slide = 90)

		Predicted		
		1	2	3
Actual	1	70	0	1
	2	26	157	1
	3	1	3	380

Figure 28: Confusion Matrix for Single Subject MSRC–12 Test Data (window = slide = 90)

recall of 93.75%, a precision of 88.23%, and an F1–score of 90.91% for change points on the training data. For the change points correctly recalled, the change was found within 2 frame

of the actual change point, on average. Notice that the spurious state 4 only occurs once.

On the test data, the recall for change points was 90.91%, precision 71.43%, and F1–score 80.00%. Correctly recalled change points were found typically within 10 frames (with an average of about 3). Since there are double the number of frames at this slide rate, the change point accuracy is on par with the results for a slide rate of 90. Also note that the extraneous state 4 was not detected.

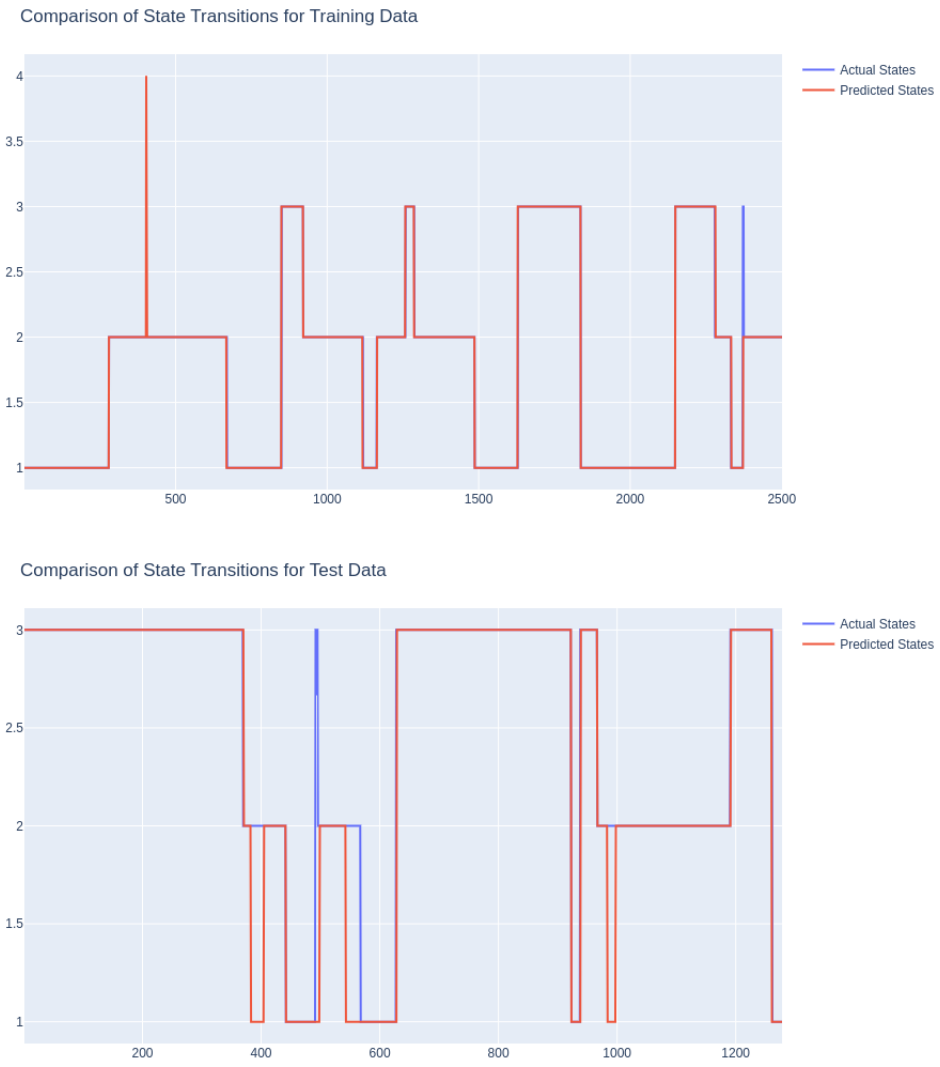


Figure 29: Results for Single Subject MSRC–12 Data (window = 90, slide = 45)

For the second round of experiments, two participants from the database were selected (participants 22 and 28) and their data for the three motions were separated into training

		Predicted		
		1	2	3
Actual	1	142	0	0
	2	64	303	1
	3	7	1	760

Figure 30: Confusion Matrix for Single Subject MSRC–12 Test Data (window = 90, slide = 45)

and test sets. Random times series of the motions were generated for both training and test sets. The states are labeled as shown in Figure 31.

State	Subject	Action
1	22	Bow
2	22	Kick
3	22	Throw
4	28	Bow
5	28	Kick
6	28	Throw

Figure 31: State Labels for Two Person MSRC–12 Trials

The first run was with the full 60–dimensional data (again after scaling and PCA analysis) with a window size and slide rate of 90. This particular run seemed to get caught in a fairly poor region of the parameter space as can be seen in Figure 32. The model identified seven distinct states, and the recall for change points was 74.68%, precision 88.06%, and F1–score 80.82% on the training data. The overall accuracy for state prediction was only 66.62%. The results for the test data were not very different.

The method had far better outcomes for a slide rate of 45 though it still finds seven distinct states (rather than six). On the training data, the recall for change points was 92.41% with a precision of 78.49% giving an F1–score of 84.88%. All correctly identified change points were typically found within 4 frames of true (with an average of about 1 frame difference). The performance on the test data was a little worse: recall of 86.67%, precision of 45.61% and an F1–score of 59.77% (recalled change points were found on average within 7 frames).

Figure 32: Results for Two Subject MSRC–12 Data
(window = 90, slide = 90, PCA dimension 60)

		Predicted						
		1	2	3	4	5	6	7
Actual	1	167	0	1	0	0	0	2
	2	33	0	1	0	2	61	0
	3	2	7	151	0	0	0	1
	4	7	0	0	0	0	0	298
	5	4	0	0	123	176	2	6
	6	5	0	4	0	0	9	2

Figure 33: Confusion Matrix for Two Subject MSRC–12 Test Data
(window = 90, slide = 90, PCA dimension 60)

Figure 34: Results for Two Subject MSRC–12 Data
(window = 90, slide = 45, PCA dimension 60)

		Predicted						
		1	2	3	4	5	6	7
Actual	1	329	0	1	1	0	2	7
	2	0	7	0	0	0	188	0
	3	0	0	322	0	0	1	0
	4	289	0	1	319	0	0	0
	5	0	0	0	161	454	7	0
	6	0	34	5	0	0	0	0

Figure 35: Confusion Matrix for Two Subject MSRC–12 Test Data
(window = 90, slide = 45, PCA dimension 60)

On the training data, the correct state was predicted for 94.61% of the frames. The extra predicted state 7 was exclusively associated to the true state 1; merging state 7 with state 1 bumps the accuracy on the training data to 95.49%. If look at how often the model predicted the correct *action* (by merging states 1 and 4, 2 and 5, and 3 and 6), the model correctly identifies the motion 97.61% of the time on the training data (going up to 98.48% by merging states 1, 4, and 7). It also correctly identifies the participant 96.40% of the time. On the test data, overall accuracy was 67.25% while the correct action was predicted 81.16% of the times. Frames were attributed to the correct participant 75.19% of the time as well.

A second set of trial was run with a PCA dimension of 12. This level of dimensionality reduction was chosen since the 12–th singular value was just under one–tenth the size of the largest. Using a slide rate of 90, change points in the training data had a recall rate of 87.34%, precision of 90.79% and an F1–score of 89.03%. The overall accuracy of state prediction was 92.88% even though the model still finds seven distinct states in the data. For the test data, recall of change points was 90.00% and precision was 75.00%, giving an F1 = 81.82%. The overall accuracy of state prediction on the test data was 87.69%. The correct action was predicted 89.94% of the time on the test data; the correct person was identified 95.77% of the time. The spurious state seven was not identified in the test data.

Changing the slide rate to 45 yielded similar results (still exhibiting seven distinct states in both training and test data). Change points in the training data had a recall rate of 91.14%, precision of 75.79% and an F1–score of 82.76%. The overall accuracy of state prediction was 91.50%. For the test data, recall of change points was 96.67% and precision was 55.77%, giving an F1–score = 70.73%. The overall accuracy of state prediction on the test data was 87.74%. The correct action was predicted 88.16% of the time on the test data; the correct person was identified 95.02% of the time.

It seems reasonable to conclude that reducing the PCA dimension helped the classification. It could be that the low dimensional topological features we are extracting are far noisier in the original high dimensional point cloud, and reducing the dimensionality (at

Figure 36: Results for Two Subject MSRC–12 Data
(window = 90, slide = 90, PCA dimension 12)

		Predicted					
		1	2	3	4	5	6
Actual	1	166	0	1	1	0	2
	2	7	86	2	0	0	2
	3	0	0	161	0	0	0
	4	16	0	0	289	0	0
	5	0	6	0	73	231	1
	6	0	17	1	2	0	0

Figure 37: Confusion Matrix for Two Subject MSRC–12 Test Data
(window = 90, slide = 90, PCA dimension 12)

Figure 38: Results for Two Subject MSRC–12 Data
(window = 90, slide = 45, PCA dimension 12)

		Predicted						
		1	2	3	4	5	6	7
Actual	1	335	0	1	4	0	0	0
	2	15	171	0	4	1	0	4
	3	0	0	323	0	0	0	0
	4	2	0	0	554	1	0	52
	5	0	0	0	135	484	0	3
	6	0	34	2	3	0	0	0

Figure 39: Confusion Matrix for Two Subject MSRC–12 Test Data
(window = 90, slide = 45, PCA dimension 12)

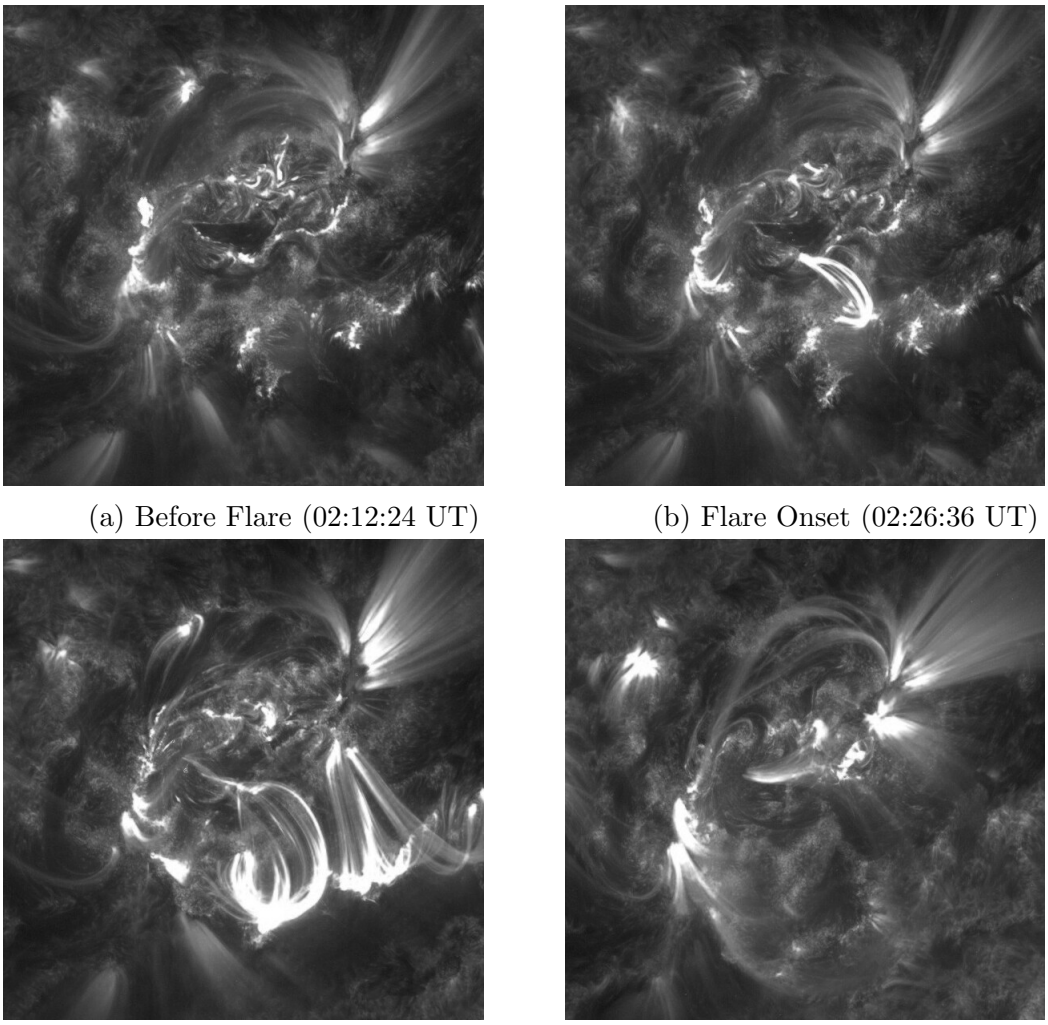
least via PCA) may be helping to increase the persistence of interesting topological features. What is extraordinary is that using the lowest two topological dimensions for persistent homology gives such decent results for data that is actually 60 dimensional.

6.3 Solar Flare Detection

As a final experiment, we look at time series classification of images using the HDP–HMMte model. Following the results of [19], topological data analysis should be a reliable tool for spotting the onset of solar flares by processing images from NASA's Solar Dynamics Observatory (SDO) spacecraft. For this experiment, 113 images were downloaded from `https://sdo.gsfc.nasa.gov/` between March, 14 2015 (16:56:24 UT) and March 15, 2015 (21:56:24 UT) (roughly one frame every 15 minutes). These dates are centered on a large X class flare [10] and associated partial halo coronal mass ejection which began on March 15, 2015 roughly at 02:10 UT. The ejecta from the flare arrived on Earth roughly two days later causing a massive geomagnetic storm in the upper atmosphere. The SDO spacecraft records images across a wide spectrum, but only images at a wavelength of 171 angstroms were used for this study. [11] All frames were cropped to track a particular sunspot (designated region 2297 on the sun's surface) which was the center of activity for the flare, and the frames were converted to gray scale for processing. Figure 40 shows sample images from the event. In addition, a second set of images was produced with the gray scale values negated (swapping black and white values) to determine if negative images are better for feature extraction.

For processing the images into persistence diagrams, the steps outlined in Image Classification with Cubical Persistent Homology [20] from the `Ripserer.jl` GitHub page were followed. Cubical homology is simply a variant of traditional homology using rectangular instead of triangular simplices. Given that images are just rectangular arrays of square pixels, this is a standard choice for image processing.[21][p. 39] After computing the $\mathcal{H}_0$ and $\mathcal{H}_1$ components from the images, the resulting diagrams were transformed into persistence images in the usual way. This gave a time series of persistence images of total length 113 with the onset of the solar flare at frame 38. For the purposes of change point detection, the frames were labeled as state 1 before this frame and state 2 after.

[10] Flares are classified in increasing intensity as A, B, C, M, and X.
[11] 171 angstroms corresponds to the upper end of the ultraviolet spectrum.



(a) Before Flare (02:12:24 UT)

(b) Flare Onset (02:26:36 UT)

(c) Flare Peak (04:11:00 UT)

(d) After Flare (21:56:24 UT)

Figure 40: Images of Solar Flare from March 15, 2015

The results of the first set of experiments are shown in Figure 42. Note that both gray-scale and reverse gray-scale frames find a transition roughly four frames before onset (roughly one hour prior) but fail to distinguish the actual start of the flare. It also looks as though the negative images produced slightly less noisy behavior during the early frames (4 states for the negative images as opposed to 5 states for the originals). For the original images, there are nearly distinct sets of states for the period one hour before the flare begins (states 1, 2, and 5) and the period including the flare (states 2, 3, and 4). State 2 shows up briefly before flare onset, and does not reappear until well after the flare has reached its peak. A similar phenomenon occurs with the negative images, as state 4 shows up at the beginning

of the recording period and well after the peak of the flare activity.

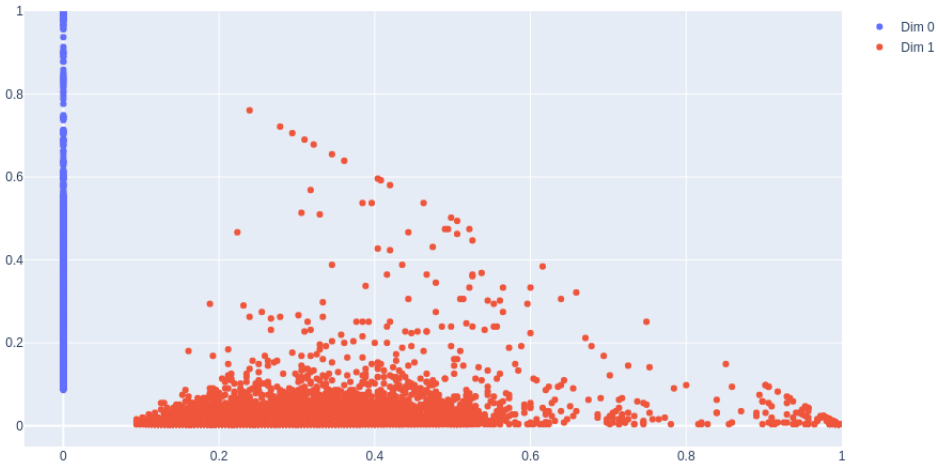
(a) Images

(b) Negative Images

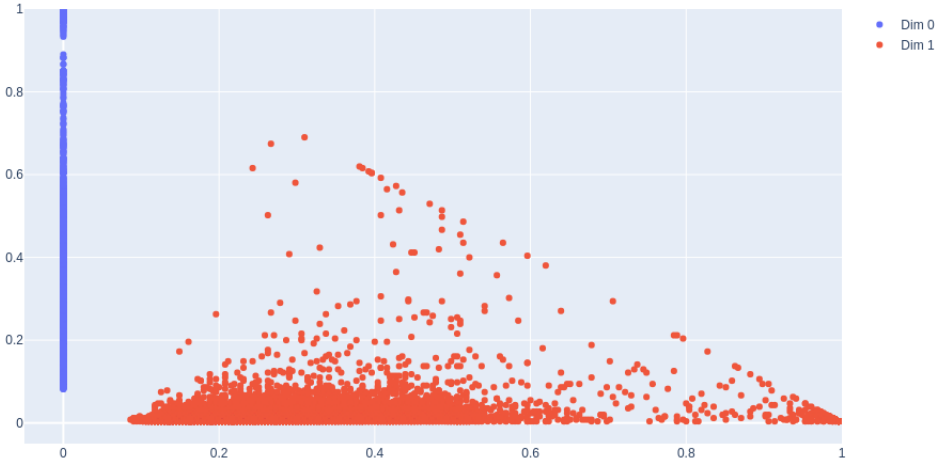
Figure 41: Flare Prediction Results

Looking at the tilted persistence diagrams in Figure 42 shows a potential issue with this method of image classification – a huge number of noise points near the bottom of the diagram. Although conversion to persistence images should lessen the impact of these noise points, the sheer number of them could be affecting classification.

To test this theory, the persistence diagrams were recomputed using a cutoff that removed



(a) Before Onset



(b) Onset

Figure 42: Tilted Persistence Diagrams for Flare Images

all points with persistence less than 0.1. With these cutoff versions of the persistence images, prediction was more reliable. In Figure 43, notice that both the original and negative images show a unique state in the early part of the series of frames (with both having a total of 6 different states). Both the original and negative images still find a transition to new behavior roughly 5 frames before onset, but the start of the flare is now clearly visible (a transition from state 5 to state 6 for the original images and from state 3 to state 4 for the negative

(a) Images

(b) Negative Images

Figure 43: Flare Prediction Results using Cutoff

images). Due to the arbitrary labeling of states, the two diagrams appear more different than they actually are. As can be seen in Figure 44, relabeling states for the negative images by $3 \to 5$, $4 \to 6$, $5 \to 4$, and $6 \to 3$ produces a graph far closer to the one from the original images. Given that all trials found an interesting transition in the topology of the images approximately four frames before onset (roughly an hour before the flares), this method might prove to be a reliable tool for early flare prediction.

Figure 44: Flare Prediction Results using Cutoff Negative Images with Relabeled States

7 Conclusion

The results of this study show that combining topological data analysis with hidden Markov models gives a powerful tool for detecting change points in time series data. The addition of a Hierarchical Dirichlet Process on top of the usual Markov structure allows the model to learn the number of states in a natural way from the data itself (obviating the need to test a variety of models with different numbers of states). This is especially useful when the number of states and transitions is unknown upfront. Even when there is some prior knowledge about the number of states, the HDP–HMMte model could find interesting transitions in the data that might otherwise go unnoticed.

Perhaps the most interesting finding from the various experiments is that replacing Gaussian likelihood with a similarity measurement (using a truncated vector norm) seems to greatly speed up the learning process. An interesting future project would be to see to what extent this is influenced by the assumption of independence of topological dimensions. Recall that the algorithm as currently implemented treats the data from each topological dimension as essentially independent of the data from other dimensions. Multiplying the likelihoods might be underestimating how similar the persistence images are to one another. Combining the dimensions into a single persistence image (perhaps using a weighted average) might improve the performance using likelihood. However, this would have the drawback of forgetting the topological dimension each point is associated with. A slightly better approach might be to model a combined persistence image as a mixture of Gaussians (using a separate mixture for each topological dimension). Another idea would be to manually cut out noise points when constructing the persistence diagrams. At least for the solar flare data, this seemed to make a significant difference to performance.

Another interesting project would be to explore changes to the standard likelihood function for identifying the best overall model. As Figure 45 shows, there are typically a variety of models with roughly equivalent log–likelihood but very different numbers of occupied states. Adding a term penalizing models with larger numbers of states might better inform model

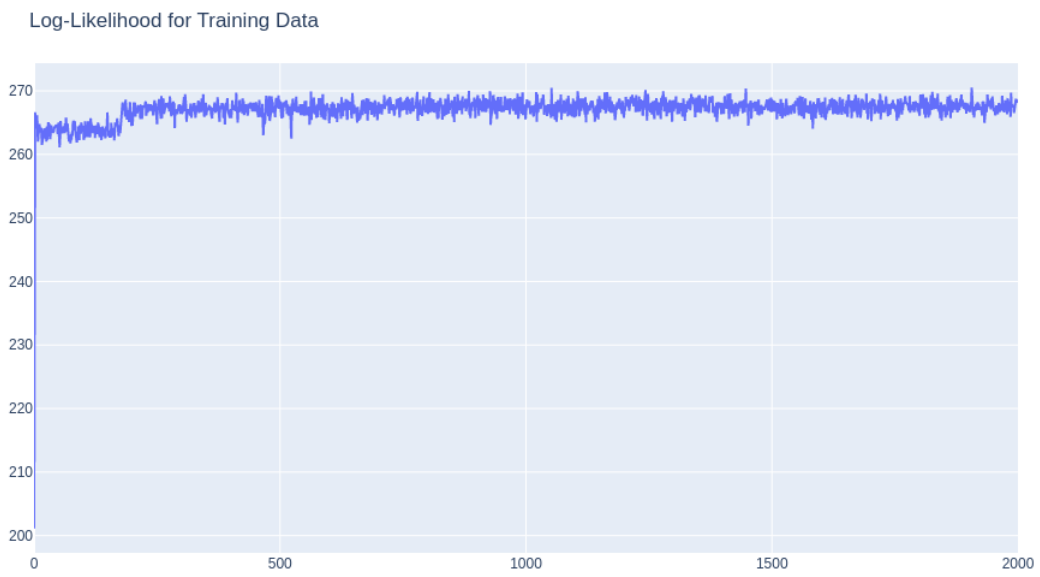


Figure 45: Typical Log–Likelihood Graph (from Variance Change Experiment 1)

selection.

Finally, it was clear from many of the experiments that the choice of window size and slide rate were important in detecting changes in the underlying topology. It seems that a slide rate of half the window size worked well generally, but it is not immediately clear why this should be or whether it would hold true in general. A detailed study looking at the impact of both window size and slide rate for data with clear topological features at known scales would help clarify this critical issue.

Bibliography

[1] Truong, C., Oudre, L., and Vayatis, N. "Selective Review of Offline Change Point Detection Methods." *Signal Processing*, 167: 107299, 2020.

[2] Adams, R.P. and MacKay, D.J.C. "Bayesian Online Changepoint Detection." *arXiv preprint arXiv:0710.3742v1.*

[3] Edelsbruner, H. and Harer, J. *Computational Topology: An Introduction.* American Mathematical Society. Providence, RI, USA (2010).

[4] Ross, S. *Introduction to Probability Models (8th ed.).* Academic Press. New York, NY, USA (2003).

[5] Jurafsky, D. and Martin, J. *Speech and Language Processing (3rd ed.).* https://web.stanford.edu/~jurafsky/slp3/. (Accessed on October 10, 2024).

[6] Ferguson, T.S. "A Bayesian Analysis of Some Nonparametric Problems." *The Annals of Statistics*, 1: pp. 209–230, 1973.

[7] Teh, Y.W., Jordan, M.I., Beal, M.J., and Blei D.M. "Hierarchical Dirichlet Processes." *J. American Statistical Association*, 101(476): pp.1566–1581, 2006.

[8] Fox, E.B., Sudderth, E.B., Jordan, M.I., Willsky, A.S. "The Sticky HDP–HMM: Bayesian Nonparametric Hidden Markov Models with Persistent States." *MIT Laboratory for Information and Decision Systems Technical Report*, P-2777: 2009.

[9] Zhou, D., Gao, Y., and Paninski, L. "Disentangled Sticky Hierarchical Dirichlet Process Hidden Markov Model." In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I, pp. 612-–627. Springer.

[10] Hatcher, A. *Algebraic Topology.* Cambridge University Press. New York, NY, USA (2001).

[11] Bauer, U. "Ripser: Efficient Computation of Vietoris–Rips Persistence Barcodes." *J. Appl. Computational Topology*, 5(3): pp. 391–423 (2021).

[12] Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., and Zeigelmeier, L. "Persistence Images: A Stable Vector Representation of Persistent Homology." *J. Machine Learning Research*, 18: pp. 1–35 (2017).

[13] Čufar, M. `Ripserer.jl`. https://mtsch.github.io/Ripserer.jl/dev/.

[14] Munkres, J. "Algorithms for the Assignment and Transportation Problems." *J. Soc. Industrial and Applied Mathematics*, 5(1): pp. 32 – 38 (1957).

[15] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms, 4ed.* The MIT Press. Cambridge, MA, USA (2022).

[16] Robinson, C. *Dynamical Systems: Stability, Symbolic Dynamics, and Chaos*. CRC Press. Boca Raton, FL, USA (1995).

[17] Anguita D., Ghio A., Oneto L., Parra X., and Reyes-Ortiz, J.L. "A Public Domain Dataset for Human Activity Recognition Using Smartphones." *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013*: Bruges, Belgium, 24-26 April 2013.

[18] Fothergill, S., Mentis, H., Kohli, P., and Nowozin, S. "Instructing People for Training Gestural Interactive Systems." *ACM SIGCHI Conference on Human Factors in Computing Systems*: Chicago, Illinois, USA (2012).

[19] Zheng, X., Mak, S., Xie., L., and Xie, Y. "PERCEPT: A New Online Change-Point Detection Method using Topological Data Analysis." *Technometrics*: 65(2), 162–178.

[20] https://mtsch.github.io/Ripserer.jl/dev/generated/malaria/

[21] Kaczynski, T., Mischaikow, K., and Mrozek, M. *Computational Homology.* Springer–Verlag. New York, NY, USA (2004).

Abstract

Brent Oneil Joseph Young

BS, University of North Carolina – Chapel Hill

MS, University of North Carolina – Wilmington

Ph.D., Rutgers University – New Brunswick

Change Point Detection via Hierarchical Dirichlet Process Hidden Markov Models with Topological Emissions

Thesis directed by Yijun Zhao, Ph.D.

This study examines the utility of Hierarchical Dirichlet Process Hidden Markov Models with topological data for emissions (HDP–HMMte) for detecting change points in time series data. The model is benchmarked against a number of synthetic data sets where there are clear changes in the underlying data (including detection of variance changes in multidimensional data). Once it is established that the model can reliably detect changes in topology, a number of studies are performed using real data. In particular, the model is shown to be effective at human motion differentiation – both in detecting activity changes from a single actor and in distinguishing different actors performing the same action. The model is also used to classify time series of images. Specifically, the model was able to detect the onset of a major solar flare event from March 2015 using images from NASA's Solar Dynamics Observatory (SDO).

Vita

Brent Oneil Joseph Young, son of Ricky and Betty Jo Young, was born on April 13, 1977 in Dunn, North Carolina, USA. After graduating in 1995 from South Johnston High School in Four Oaks, NC, he attended the University of North Carolina – Chapel Hill ultimately receiving a Bachelor of Science in Physics in 1999. After graduation, he did a year of volunteer work with Jesuit Volunteer Corps in Dallas, TX before taking a job as an industrial chemist for two years.

In 2003, he returned to academia by enrolling at the University of North Carolina – Wilmington where he earned a Master of Science in Mathematics in 2005. He was then accepted to the graduate program in Mathematics at Rutgers University – New Brunswick and received a Ph.D. in Mathematics in 2011. After a two year postdoctoral position at the University of Cologne in Germany, he took a job as an Associate Professor of Mathematics at Wilkes University in Wilkes–Barre, PA. In 2021, he was granted tenure and promoted to Associate Professor.