Norwegian Institute of Science and Technology

# Clustering animal behavior

TMA4500

Specialization project in Industrial Mathematics
autumn 2022, supervised by Associate Professor
Benjamin Adric Dunn

**Ulrik Bernhardt Danielsen**

# Contents

# List of Figures

## Abstract

This specialization project serves as a preparation for my master thesis in Industrial Mathematics at NTNU. My aim is to extend and justify a clustering methodology grouping motion recordings of free roaming rats into distinct behaviours. Using time frequency analysis to extract information about repeating movement patterns, and dimensionality reduction techniques such as principal component analysis and t-distributed stochastic neighbor embedding, we can separate behaviors in a two dimensional space. These detected behaviors can then be used in further research, e.g., investigating connections between neural recordings and behavior, or behavioral effects by various stimulation.

# Chapter 1

# Introduction

## 1.1 Motivation

The human brain is an incredibly complex structures that researchers have been trying to understand for a long time. One way to gain information about how the brain operates is to study its neurons. Neurons are cells which can communicate with each other through synapses. This communication are electric signals and can be recorded. At Kavli Institute for Systems Neuroscience at NTNU researchers are interested in relating these neural spike recordings to the behavior in rats. This in turn begs the question of how rats behave. Manually labelling video recordings of rats running around seems a tedious and unfruitful endeavor. Additionally it introduces bias in our prior assumptions of how the rats behave, and which activities they engage in. Thus, a methodology for automatically detecting distinct behaviours is needed.

## 1.2 Previous work

Through recent advances in machine learning and computer vision, many attempts at decomposing animal behavior into small modules has been made. Manually labelling recordings of *Drosophila melanogaster*, the fruit fly, before applying machine learning prediction techniques shows performance on par with human labelling [1]. Unsupervised clustering techniques have also been developed for the fruit fly, using dimensionality reduction techniques to separate distinct behaviors in a two dimensional space [2]. Research has not only been done on the fruit fly. The MoSeq group at Harvard university uses 3D video capturing on free ranging mice. Modelling the behavior modules as vector autoregressive processes, and changes between modules as a hidden Markov model, allows researches to extract information about rat body language and behavior [3]. Recording the behavior in 3D as opposed to 2D plays a big part of these methods success [4] As opposed to using depth cameras to provide the 3D kinematic recording, use of motion sensors tracking separate body movements have also been used [5].

# Chapter 2

# Theory

## 2.1 Time series analysis

We define time series as a realization $y_t = \{y_{t_1}, y_{t_2}, \ldots, y_{t_n}\}$ of a stochastic process $Y(\omega, t)$, where $\omega \in \Omega$, $\Omega$ being the sample space, and $t \in \mathbb{Z}$, $\mathbb{Z}$ being the chosen index set [6]. It is an ordered series of random variables which can be described completely by its joint probability function

$$F_{t_1,\ldots,t_n}(x_1, \ldots, x_n) = \Pr\{y_{t_1} \leq x_1, \ldots, y_{t_n} \leq x_n)\}.$$

The mean and variance function of a time series $y$ are defined as

$$\mu_t = E(y_t) \tag{2.1}$$

and

$$\sigma_t^2 = E(y_t - \mu_t)^2.$$

Given two random variables in the series $y_{t_1}$ and $y_{t_2}$, we define the covariance function and correlation function as

$$\gamma(t_1, t_2) = E[(y_{t_1} - \mu_{t_1})(y_{t_2} - \mu_{t_2})] \tag{2.2}$$

and

$$\rho(t_1, t_2) = \frac{\gamma(t_1, t_2)}{\sqrt{\sigma_{t_1}^2}\sqrt{\sigma_{t_2}^2}}. \tag{2.3}$$

### 2.1.1 Stationarity

A time series $y_t$ is $n$th-order stationary if for any shift $h$ and indexes $t_1, t_2, \ldots, t_n$ if

$$F_{y_{t_1},\ldots,y_{t_n}}(x_1, \ldots, x_n) = F_{y_{t_1+h},\ldots,y_{t_n+h}}(x_1, \ldots, x_n). \tag{2.4}$$

If (2.4) holds for all $n$, the time series is called *strictly* stationary. We also define a $n$th-order *weakly* stationary time series $y_t$ if the first $n$ joint moments are finite and time invariant. Specifically we define the second-order weakly stationary, i.e. with constant and time invariant mean function (2.1), and where the covariance function (2.2) is solely a function of the time difference,

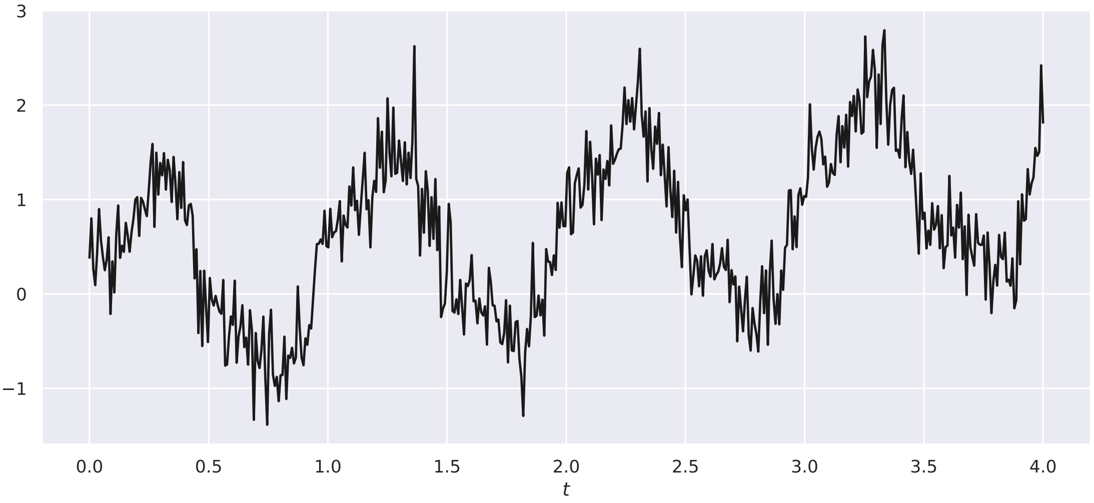


Figure 2.1: Example of a non-stationary time series with $t_0 = 0, t_n = 4$. The discrete data points are connected to better visualize the movement through time.

as *covariance* stationary. When the covariance function between $t_1, t_2$ can be written as a function of the time difference $h = |t_1 - t_2|$, i.e. $\gamma(t_1, t_2) = \gamma(h) = \gamma_h$, we call it an *autocovariance* function. The same is true for the correlation function (2.3), which when is a function of the time difference is called an *autocorrelation* function (ACF). Figure 2.1 shows an example of a time series. As the mean seem to increase with $t$ it is non-stationary.

We also define the common estimates for the mean and covariance functions. They are called the sample mean and sample covariance, and are written as

$$\bar{y_t} = \frac{1}{n} \sum_{i=1}^{n} y_{t_i}, \tag{2.5}$$

$$\hat{\gamma}_h = \frac{1}{n} \sum_{i=1}^{n-h} (y_{t_i} - \bar{y_t})(y_{t_i+h} - \bar{y_t}), \tag{2.6}$$

respectively.

### 2.1.2 Detrending

Many methods for analysing and processing time series requires stationarity [7]. If the series is non-stationary, we can split the it into one stationary and one non-stationary part called the *trend*. Mathematically we write it as

$$y_t = \mu_t + x_t,$$

where $x_t$ denotes the stationary part and $\mu_t$ the trend. The process of finding $\mu_t$ and then computing $x_t = y_t - \mu_t$ is called *detrending*. Detecting the trend can be done in many ways, for instance using regression techniques or smoothing. The simplest way is to assume a linear trend, $\mu_t = \beta_0 + \beta_1 t$ and estimate the parameters using least squares. In figure 2.2 the linear regression fit is shown, showing an upwards in the time series.

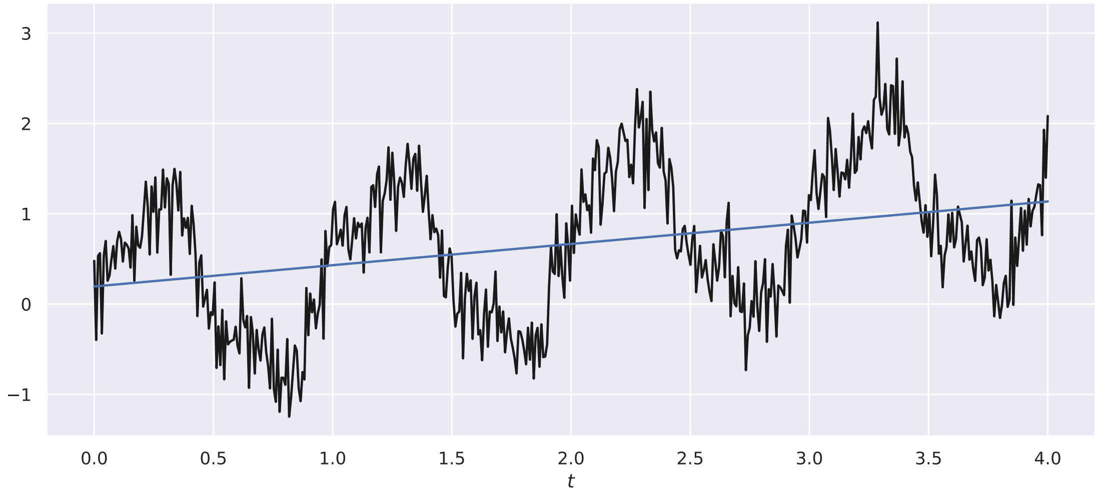


Figure 2.2: Time series from figure 2.1 with an estimated linear trend shown in blue. It is clear that there exists an upward trend in the data.

## 2.2 Fourier analysis

Let $Z_1, Z_2, \ldots, Z_n$ be a sequence of numbers. For simplicity in notation we assume $n$ to be an odd number. It can be shown that the sequence can be represented as a linear combination of complex exponentials

$$Z_t = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} c_k e^{\frac{i2\pi kt}{n}}. \tag{2.7}$$

This comes from the fact that the set

$$\left\{ e^{\frac{i2\pi kt}{n}} \middle| k \in \left[ -\frac{n-1}{2}, \frac{n-1}{2} \right] \right\}$$

consists of $n$ orthogonal functions [6]. I.e., that

$$\sum_{i=1}^{n} e^{\frac{i2\pi kt}{n}} e^{-\frac{i2\pi jt}{n}} = \begin{cases} n, & k = j \\ 0, & k \neq j \end{cases}.$$

The coefficients $c_k$ are given by

$$c_k = \frac{1}{n} \sum_{t=1}^{n} Z_t e^{-\frac{i2\pi kt}{n}}.$$

It is clear that (2.7) is periodic with period $n$, meaning $Z_{t+jn} = Z_t, j = 0, \pm 1, \pm 2, \ldots$. Thus the Fourier series is able to capture periodic sequences. The smallest positive integer $n$ for which $Z_{t+n} = Z_t$ is called the fundamental period, with corresponding fundamental frequency $2\pi/n$. For the components $k = \pm j, j = 1, 2, \ldots, (n-1)/2$ the frequencies are multiples of the fundamental frequency, $\omega_k = k(2\pi/n)$. The set of frequencies making up the series is called the *spectrum*. As a consequence the coefficients $c_k$ can be viewed as weighting the importance of the contributions for the different frequencies making up the full sequence. This is formalized by the definitions of energy and power,

$$\text{energy} = \sum_{t=2}^{n} Z_t^2 = n \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} |c_k|^2, \tag{2.8}$$

$$\text{power} = \frac{\text{energy}}{n} = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} |c_k|^2. \tag{2.9}$$

Let $p_k$ be the contribution to the power from frequency $k = 0, 1, \ldots, (n-1)/2$. As $\omega_k$ and $\omega_{-k}$ corresponds to the same frequency the contribution is given as $p_0 = c_0^2, p_k = 2|c_k|^2, k = 1, \ldots, (n-1)/2$. The values $p_k$ are called the power spectrum of the series.

### 2.2.1 Discrete-Time Fourier Transform

We have seen that all sequences of length $n$ can be viewed and parameterized as Fourier series with period $n$. Moving to non-periodic sequences essentially amounts to taking the limit of the series as $n$ approaches infinity. Formally we now let $Z_t$ be a finite discrete function of $t$, where $Z_t = 0$ when $|t| > M$ for some integer $M$. Choosing $n = 2M + 1$ the function

$$Y_{t+jn} = Z_t, \; t \in \left[-\frac{n-1}{2}, \frac{n-1}{2}\right], \; j \in \mathbb{Z}$$

is periodic with period $n$. It's Fourier series is

$$Y_t = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} c_k e^{\frac{i2\pi kt}{n}}.$$

As $Y_t = Z_t$ when $t \in [-(n-1)/2, (n-1)/2]$, and $Z_t = 0$ when $|t| > (n-1)/2$, the coefficients $c_k$ can be written as the infinite sum

$$\begin{aligned} c_k &= \frac{1}{n} \sum_{t=-\infty}^{\infty} Z_t e^{\frac{-i2\pi kt}{n}} \\ &= \frac{2\pi}{n} f\left(\frac{2\pi k}{n}\right), \end{aligned}$$

where

$$f(\omega) = \frac{1}{2\pi} \sum_{t=-\infty}^{\infty} Z_t e^{-i\omega t}.$$

If we now take the limit $Z_t = \lim_{n\to\infty} Y_t$ the summation becomes an integral over the length $2\pi$ [6]. This gives the relation

$$Z_t = \int_{-\pi}^{\pi} f(\omega) e^{i\omega t} d\omega, \quad t \in \mathbb{Z} \tag{2.10}$$

$$f(\omega) = \frac{1}{2\pi} \sum_{t=-\infty}^{\infty} Z_t e^{-i\omega t}, \quad -\pi \leq \omega \leq \pi, \tag{2.11}$$

where $f(\omega)$ in (2.11) is called the discrete-time Fourier transform of $Z_t$. As opposed to the periodic case (2.7) where the periodic sequence was made up by a finite number of frequencies, the non-periodic sequence is an integral over a continuum of frequencies $\omega$. We call $|f(\omega)|$ the spectrum of the sequence, and the function $g(\omega) = 2\pi|f(\omega)|^2$ the energy spectrum. The energy spectrum definition comes from Parseval's relation

$$\sum_{t=-\infty}^{\infty} |Z_t|^2 = 2\pi \int_{-\pi}^{\pi} |f(\omega)|^2 d\omega. \tag{2.12}$$

It is worth noting that the relation in (2.12) only holds when the sequence $Z_t$ is absolutely summable, i.e., that

$$\sum_{t=-\infty}^{\infty} |Z_t| < \infty. \tag{2.13}$$

## 2.3 Spectral density estimation

Let $y_t$ be a stationary time series where the autocovariance function $\gamma_h$ from (2.2) is absolutely summable. Then we can write $\gamma_h$ as a Fourier transform pair

$$f(\omega) = \frac{1}{2\pi} \sum_{h=-\infty}^{\infty} \gamma_h e^{-i\omega h}, \tag{2.14}$$

$$\gamma_h = \int_{-\pi}^{\pi} f(\omega) e^{i\omega h} d\omega. \tag{2.15}$$

It can be shown [7] that the spectrum $f(\omega)$ in (2.14) is real-valued and non-negative. Furthermore, as $\mathrm{Var}(y_t) = \gamma_0$, we get the interpretation

$$\mathrm{Var}(y_t) = \int_{-\pi}^{\pi} f(\omega) d\omega,$$

i.e., that $f(\omega)$ is the contribution to the variance for frequency $\omega$. We often want to locate these important frequencies, and thus an important task is to estimate this spectrum.

### 2.3.1 Periodogram

Again we consider a time series sample $y_1, y_2, \dots, y_n$ where $n$ is chosen to be odd for simplicity. It can be written as a real Fourier representation

$$y_t = a_0 + \sum_{k=1}^{\frac{n-1}{2}} (a_k \cos(\omega_k t) + b_k \sin(\omega_k t)),$$

where $\omega_k = 2\pi k/n, k = 0, 1, \dots, (n-1)/2$ and the coefficients are given as

$$a_0 = \bar{y}_t, \quad a_k = \frac{2}{n} \sum_{t=1}^{n} y_t \cos(\omega_k t), \quad b_k = \frac{2}{n} \sum_{t=1}^{n} y_t \sin(\omega_k t).$$

We then define the periodogram as

$$I(\omega_k) = \begin{cases} n a_0^2, & k = 0 \\ \frac{n}{2}(a_k^2 + b_k^2), & k = 1, \dots, (n-1)/2. \end{cases} \tag{2.16}$$

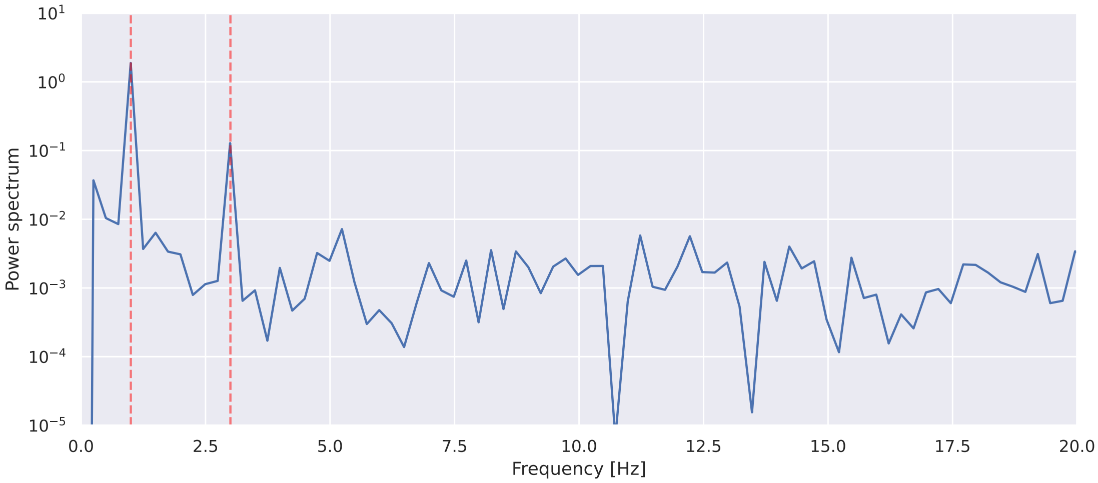


Figure 2.3: Periodogram for the time series plotted in figure 2.1. The time series is detrended assuming a linear trend. Observe that the two highest peaks in the periodogram are at the underlying frequencies 1Hz and 3Hz shown with dashed red lines.

The periodogram is of interest as it has has a large value if the frequency $\omega_k$ is of importance in the series. A scaled periodogram $\frac{2}{n}I(\omega_k)$ estimates the sample variance of the sinusoid component at frequency $\omega_k$ [7]. A periodogram for the time series example in figure 2.1 are shown in figure 2.3. Note that the time series is linearly detrended for the periodogram is computed. The two highest peaks match the underlying generating function which can be revealed to be $\sin(2\pi t) + \frac{1}{4}\cos(6\pi t) + t/3$.

### 2.3.2 Sample spectrum

One intuitive way of estimating the spectrum is to replace the autocovariance with the sample autocovariance from equation (2.6). I.e., we define the sample spectrum for a realization $y_1, y_2, \dots, y_n$

$$\hat{f}(\omega) = \frac{1}{2\pi} \sum_{k=-(n-1)}^{n-1} \hat{\gamma}_k e^{-i\omega k}. \tag{2.17}$$

At the Fourier frequencies $\omega_k$ it is related to the periodogram through [6]

$$\hat{f}(\omega_k) = \frac{I(\omega_k)}{4\pi}.$$

Although $\hat{f}(\omega_k)$ is asymptotically unbiased, meaning $\lim_{n\to\infty} E(\hat{f}(\omega)) = f(\omega)$, is lacks consistency in the variance as $n$ tends to infinity, i.e.,

$$\lim_{n\to\infty} \operatorname{Var}(\hat{f}(\omega_k)) \neq 0.$$

### 2.3.3 Spectral Window

The fact that the variance does not decrease with the sample size produces quite jagged and noisy spectrum estimations [7]. To account for this we introduce the spectral window which smooths the spectrum. Mathematically the smoothed spectrum is written as

$$\hat{f}_{\mathcal{W}}(\omega_k) = \sum_{j=-m}^{m} \mathcal{W}_n(\omega_j)\hat{f}(\omega_k - \omega_j), \tag{2.18}$$

where $\omega_k = 2\pi k/n,\ k = 0, 1, \dots, (n-1)/2$ and $m$ is a function of $n$, typically $m \ll n$. The value of $m$ decides how many points in the neighborhood of $\omega_k$ should be included in the smoothing. Furthermore, the function $\mathcal{W}_n(\omega_j)$ is chosen to have the following properties,

$$\begin{aligned}
&\sum_{j=-m}^{m} \mathcal{W}_n(\omega_j) = 1,\\
&\mathcal{W}_n(\omega_j) = \mathcal{W}_n(-\omega_j),\\
&\lim_{n\to\infty} \sum_{j=-m}^{m} \mathcal{W}_n^2(\omega_j) = 0.
\end{aligned} \tag{2.19}$$

We view the smoothed spectrum as a weighted average of the sample spectrum (2.17) in a window around the target frequency $\omega_k$. How the weights are distributed in the window is governed by $\mathcal{W}_n(\omega_j)$, giving it the name spectral window. Because of the property in (2.19) we have

$$\begin{aligned}
\mathrm{Var}(\hat{f}_{\mathcal{W}}(\omega_k)) &\approx \sum_{j=-m}^{m} \mathcal{W}_n^2(\omega_j)(f(\omega_k))^2\\
&= (f(\omega_k))^2 \sum_{j=-m}^{m} \mathcal{W}_n^2(\omega_j) \overset{n\to\infty}{=} 0,
\end{aligned}$$

assuming $f(\omega)$ is approximately constant in the window. We can thus reduce the variance by increasing the points included in the window. However, when doing so we introduce bias.

### 2.3.4 Lag window

It can be shown [6] that the spectral window forms a Fourier transform pair with a lag window $W_n(k)$, i.e,

$$W_n(k) = \int_{-\pi}^{\pi} \mathcal{W}_n(\omega)e^{i\omega k}d\omega, \quad k = 0, \pm1, \dots, \pm M.$$

The lag window is a weighting function applied to the sample autocovariance

$$\hat{f}_{\mathcal{W}} = \frac{1}{2\pi} \sum_{k=-M}^{M} W_n(k)\hat{\gamma}_k e^{-i\omega k},$$

with $W_n = W(k/M)$, $W$ being a bounded even continuous function

$$\begin{aligned}
&|W(t)| \le 1,\\
&W(0) =,\\
&W(t) = W(-t),\\
&W(t) =,\ t > 1.
\end{aligned}$$

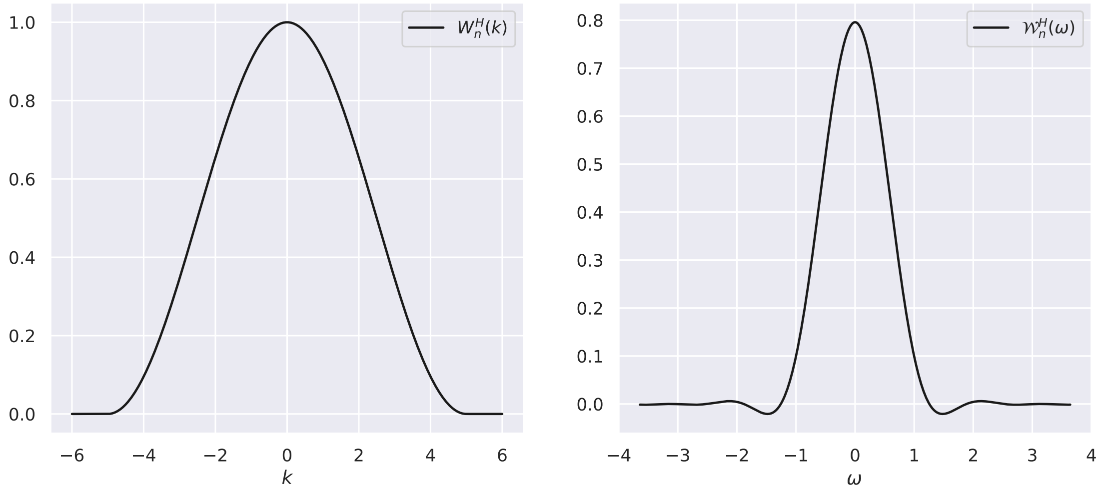


Figure 2.4: On the left the hanning window given by equation (2.20). On the left the corresponding spectral window, i.e., its Fourier transform.

Figure 2.4 shows the popular *hanning* window given by

$$W_n^H = \begin{cases} \frac{1}{2}(1+\cos(\frac{\pi k}{M})), & |k| \leq M \\ 0, & |k| > M, \end{cases} \tag{2.20}$$

with $M = 5$.

## 2.4 Time frequency analysis

The methods mentioned so far are only suitable for extracting information about the frequencies of a signal. If the process is stationary this is often all we need. However, if the process is non-stationary and the frequencies change over time, the periodogram is unable to capture this. The original time series contains all the information in the time-scale, and the Fourier transform contains all the information in the frequency scale. We need something in between.

### 2.4.1 Short-Time Fourier Transform

A simple and intuitive way to gain information about the changes in frequency is to divide the time interval into sections, and compute the Fourier transform on each section. Then we can plot how the periodogram changes over time. If we divide the time interval using a window function this method is called the short-time Fourier Transform (STFT). Mathematically we define it for a discrete sequence $y = y_1, y_2, \ldots, y_n$, which is windowed by $W_n(t)$ around time $\tau$, as

$$S_y(\omega, \tau) = \frac{1}{2\pi} \sum_{t=-\infty}^{\infty} y_t W_n(t-\tau) e^{-i\omega t}. \tag{2.21}$$

The estimated spectrum $|S_y(\omega, \tau)|^2$ is now a function of both time and frequency, and is commonly called the spectrogram [8], which can be plotted to show the changes in frequencies over time.

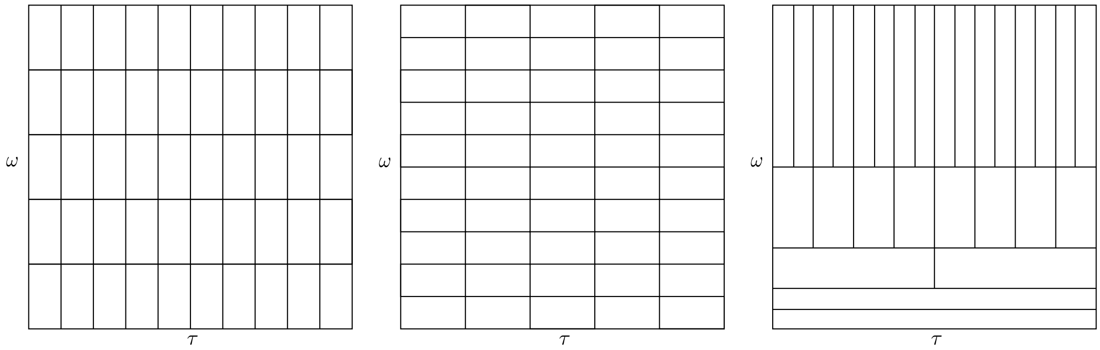


Figure 2.5: *Left/middle:* The figures illustrates the tradeoff in resolution for time and frequency for the short-time Fourier transform from equation (2.21), increasing the resolution in time decreases the resolution in frequency and vice versa. *Right:* Illustrate the corresponding resolution for the continuous wavelet transform from equation (2.22).

Unfortunately the STFT has some severe drawbacks; it can be shown that there exists a limit for the possible precision achieved when measuring both frequency and time simultaneously [9]. There exists a tradeoff in time and frequency resolution, as shown in figure 2.5. This is very intuitive as frequency has to be measured over a certain time period. Decreasing the width of the window function makes detecting smaller frequencies harder, and vice versa. Thus we need to know the approximate frequency scales in the data before performing the STFT, or we can try several window widths. In the case where frequencies exists in the data on several different scales, the STFT in equation (2.21) becomes unsuitable.

### 2.4.2 Wavelet transform

The wavelet transform works in a very similar way as the STFT, but circumvents the problem of choosing a fixed width of the window by introducing scaling. Let $\psi(t)$ be a window function (possibly complex valued) which we will call the mother wavelet. One way of scaling the mother wavelet by a factor $s$ is

$$\psi_s(t) = \frac{1}{s^{1/2}}\psi\left(\frac{t}{s}\right).$$

If we also allow a time shift around $\tau$ we arrive at the wavelets

$$\psi_{s,\tau}(t) = \frac{1}{s^{1/2}}\left(\frac{t-\tau}{s}\right).$$

This culminates in the definition of the continuous wavelet transform (CWT) of a function $f$ [9]

$$\tilde{f}(s,\tau)\int_{-\infty}^{\infty}\psi^*_{s,\tau}(t)f(t)dt, \tag{2.22}$$

where $\psi^*_{s,\tau}$ is the complex conjugate of the function $\psi_{s,\tau}$.

Again considering the discrete sample $y_1, y_2, \ldots, y_n$ with $\delta t = y_{t+1} - y_t,\ t = 1, \ldots, n$, its continuous wavelet transform is given by [10]

$$\tilde{f}_n(s,\tau) = \sum_{t=1}^{n} y_t\psi^*\left(\frac{\delta t}{s}(t-\tau)\right), \tag{2.23}$$

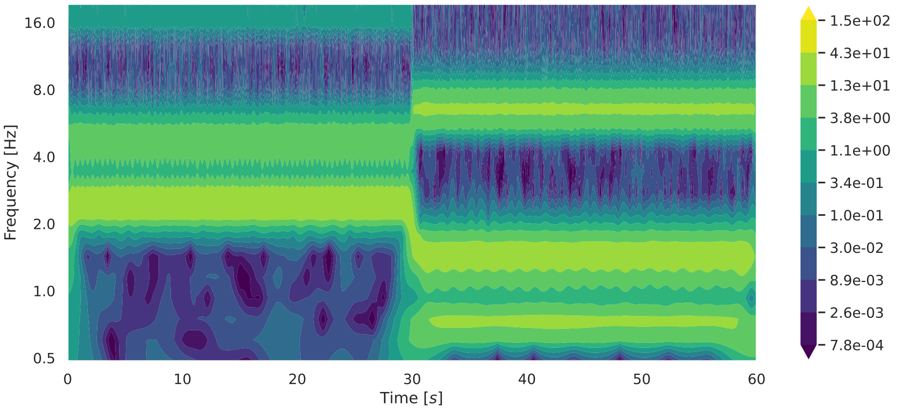


Figure 2.6: An example of a scaleogram of the continuous wavelet transform of function (2.25) with added Gaussian noise.

where $\tau$ now is a time index $\tau \in \{1, 2, \dots, n\}$. It is computationally efficient [10] to represent the CWT as the inverse Fourier transform

$$\begin{aligned}
\tilde{f}_n(s,\tau) & \sum_{t=1}^{n} \hat{y}_k \hat{\psi}^*(s\omega_k) e^{i\omega_k n \delta t}, \\
\hat{y}_k &= \frac{1}{n} \sum_{t=1}^{n} y_t e^{-\frac{i2\pi k t}{n}}, \\
\omega_k &= \begin{cases} \frac{2\pi k}{n\delta t}, & k \leq \frac{n}{2} \\ -\frac{2\pi k}{n\delta t}, & k > \frac{n}{2} \end{cases}.
\end{aligned}$$

As always we are interested in the power spectrum, which for the CWT is defined as $|\tilde{f}_n(s,\tau)|^2$.

There exists many choices for the mother wavelet, and certain important criteria it must satisfy [9]. One popular wavelet is the Morlet wavelet defined as

$$\psi_0(\eta) = \pi^{-1/4} e^{i\omega_0 \eta} e^{-\eta^2/2}, \tag{2.24}$$

where $\omega_0$ is a parameter chosen to fit the criteria.

In figure 2.6 a scaleogram is shown for the function

$$f(t) = \begin{cases} 3\sin(2.4 \cdot 2\pi t) + 2\sin(4.7 \cdot 2\pi t) + \sin(17 \cdot 2\pi t), & t < 30 \\ \sin(0.7 \cdot 2\pi t) + 2\sin(1.4 \cdot 2\pi t) + 3\sin(6.4 \cdot 2\pi t), & t \geq 30, \end{cases} \tag{2.25}$$

with added Gaussian noise. The continuous wavelet transform is computed at 18 frequencies between 0.5Hz and 20 Hz.

## 2.5 Piecewise polynomials

Suppose we have an interval $[a, b]$ divided into $M$ contiguous subintervals. The connecting edges of the subintervals $a = \xi_0, \xi_1, \dots, \xi_{M-1}, \xi_M = b$ are called knots. On each of the intervals

$[\xi_i, \xi_{i+1}], i = 0, \ldots, M-1$ we define a polynomial $p_i(t)$. The function

$$f(t) = \begin{cases} p_0(t), & t \in [\xi_0, \xi_1) \\ p_1(t), & t \in [\xi_1, \xi_2) \\ & \vdots \\ p_{M-1}(t), & t \in [\xi_{M-1}, \xi_M] \end{cases}$$

is called a *piecewise polynomial.*

### 2.5.1 Splines

In the definition of piecewise polynomials no restrictions are made on the polynomials, they are allowed to take any form. As in [11] we define a *spline* $s_k(t)$ of order $k$ on the interval $[a, b]$ as a piecewise polynomial where

$$\begin{aligned} &s_k(t) \in \mathcal{P}^k, \quad t \in [\xi_i, \xi_{i+1}], \quad i = 0, 1, \ldots, M-1 \\ &s_k(t) \in \mathcal{C}^{k-1}[a, b]. \end{aligned}$$

I.e., the spline consists of piecewise polynomials of order $k$ and has continuous derivatives up to order $k-1$. A common choice is letting $k = 3$, providing continuous second derivatives over the interval. This is called *cubic* splines, and are often considered sufficiently smooth for function approximations. It is also common to add curvature constraints at the endpoints, $s_3''(a) = s_3''(b)$, arriving at the *natural* cubic splines.

### 2.5.2 Regression splines

Suppose now we have data points $y_{t_1}, y_{t_2}, \ldots, y_{t_n}$ on $[a = t_1, b = t_n]$. A spline of order $k$ with chosen knots at $a = t_1 = \xi_0, \xi_1, \ldots, \xi_M = t_n = b$ can be parameterized as

$$s_k(t) = \sum_{i=1}^{M+k} \beta_i h_i(t), \tag{2.26}$$

where the functions $h_i$ are the truncated-power basis set

$$\begin{aligned} h_j(t) &= t^{j-1}, \ j = 1, \ldots, k+1, \\ h_{k+1+l}(t) &= (t - \xi_l)_+^k, \ l = 1, \ldots, M-1, \end{aligned}$$

with $(t)_+ = \max\{t, 0\}$ [12]. The parameters $\beta_i$ can be found using least squares. An example of cubic spline regression are shown in figure 2.7.

## 2.6 Dimensionality reduction

Suppose we have $n$ datapoints, each having $p$ numerical features. Mathematically we represent them as the vectors $x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \in \mathbb{R}^p, \ i = 1, 2, \ldots, n$, often represented in the $n \times p$ data matrix $X$, $X_{ij} = x_{ij}, \ i = 1, 2, \ldots, n, \ j = 1, 2, \ldots, p$. Dimensionality reduction is about finding a low dimensional representation of the data still containing most of its properties. There can be many reasons for performing dimensionality reduction, and often it is performed as a form of feature extraction used before predictive modelling or clustering [12]. If the data is sparse or the features highly correlated, reducing the dimensionality could improve computation speed and remove noise. Reducing the data to two dimensions often allows for easier visualization and interpretability of the data.

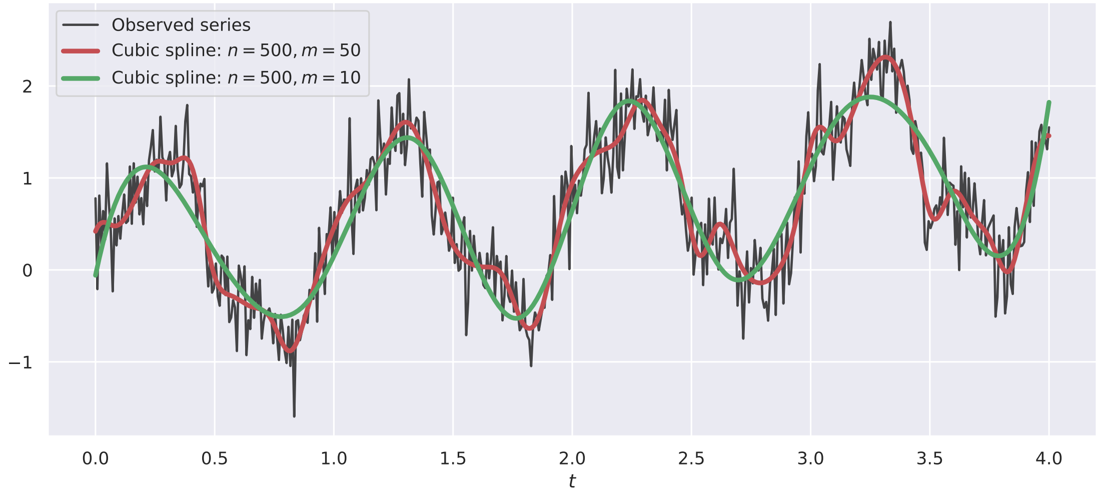


Figure 2.7: Two cubic splines fitted using least square regression on the time series from figure 2.1. Observe that the red spline with 50 knots (including endpoints) fits the data closer than the green spline with 10 knots.

### 2.6.1 Principal component analysis

One of the most commonly used dimensionality reduction methods is called principal component analysis (PCA). There are multiple ways to both derive and interpret the method. Given the $n \times p$ data matrix $X$ we find a sequence of $q \leq p$ orthogonal $p \times 1$ unit vectors $v_1, v_2, \dots, v_q$, each one chosen such that $Z_j = Xv_j$ has maximum variance. On matrix form we write it as $Z = XV_q$, where the columns of $V_q$ are $v_j, j = 1, 2, \dots, q$. The principal components scores—which represents our new features when applying dimensionality reduction—is given by the $q$ columns in $Z$. Let $S$ be the sample covariance matrix of $X$. It can be shown [13] that the vectors $v_j$ that maximises the variance while subject to the mentioned constraints, are the eigenvectors of $S$, with the corresponding eigenvalue $\lambda_j$. Note that $\lambda_i > \lambda_{i+1},\ i = 1, 2, \dots, p-1$. For this reason we need to standardize the columns of $X$ to have mean zero and variance one before applying PCA. The proportion of the total variance explained by the first $q$ principal components is given by

$$\frac{\sum_{i=1}^{q} \lambda_i}{\sum_{j=1}^{p} \lambda_j}.$$

Thus when using PCA for dimensionality reduction we set a variance threshold, and keep the number of principal components keeping enough variance.

### 2.6.2 t-Stochastic Neighbor Embedding

Another dimensionality reduction method specially suited for visualizing data in two dimension, is called t-Stochastic Neighbor Embedding (t-SNE). Again we start with the $n \times p$ data matrix, and wish to embed it into a low dimensional space preserving as much of the original structure in the data as possible. Instead of trying to emulate the Euclidian distances between the data points $x_1, x_2, \ldots, x_n$, we convert them into conditional probabilities

$$p_{j|i} = \frac{\exp\left\{\frac{-||x_i - x_j||^2}{2\sigma_i}\right\}}{\sum_{k\neq i}\exp\left\{\frac{-||x_i - x_k||^2}{2\sigma_i^2}\right\}}, \quad i \neq j,\ p_{i|i} = 0. \tag{2.27}$$

For a point $x_i$, $p_{j|i}$ is large for nearby points $x_j$, the magnitude proportional to a Gaussian distribution centered at $x_i$ with variance $\sigma_i^2$ [14]. The individual variance $\sigma_i^2$ is a tuning parameter to be chosen by the user.

In the low dimensional representation we model the similarities as joint probabilities based on the Student t-distribution with one degree of freedom

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1 + ||y_k - y_l||^2)^{-1}}, \quad i \neq j,\ q_{ii} = 0. \tag{2.28}$$

It is beneficial to define a joint probability distribution in the high dimensional space as with $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$ [14]. To find the low dimensional embedding $Y$, t-SNE minimizes the total Kullback-Leibler divergence between the discrete joint probability distributions

$$KL(P||Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}\log\left(\frac{p_{ij}}{q_{ij}}\right). \tag{2.29}$$

The choice of using the Student-t distribution in equation (2.28) as opposed to a Gaussian (as in the high dimensional space (2.27)), stems from its heavier tails. This allows data points which are relatively dissimilar in the original data, but still close enough to make a difference in the cost function, to be modelled far apart in the low dimensional embedding. I.e., t-SNE focusses on the local structures in the data, and allows clusters to be well separated in the embedding.

Furthermore, we need to choose the tuning parameters $\sigma_i$. They relate to the Gaussian distribution modelling the neighbors of $x_i$. For points with many close by neighbors it makes sense to keep $\sigma_i$ small, while increasing it is beneficial when there are few neighbors. The algorithm solves this by letting the user choose a perplexity parameter

$$\text{Perp}(P_i) = 2^{H(P_i)}, \tag{2.30}$$

where

$$H(P_i) = -\sum_{j=1}^{n} p_{j|i}\log_2(p_{j|i})$$

is the Shannon entropy. Then it finds $\sigma_i$ such that the perplexity satisfies our choice. Van der Maaten and Hinton [14] suggests values between 5 and 50, roughly reflecting the number of neighbors in the algorithm.

## 2.7 Kernel density estimation

Let $x_1, x_2, \ldots, x_n,\ x_i \in \mathbb{R}^p$ be a random sample from a random variable with an underlying unknown density function $f(x)$, which we want to estimate. The general expression for the multivariate kernel density estimator is [15]

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_p\left[H^{-1}(x - x_i)\right], \tag{2.31}$$

where $|H| = |\det(H)|$, $H$ being the $p \times p$ non-singular bandwidth matrix. The function $K_p : \mathbb{R}^p \to \mathbb{R}$ in (2.31) is called the kernel function, and must satisfy

$$\int_{\mathbb{R}^p} K_p[u]du = 1, \int_{\mathbb{R}^p} uK_p[u]du = 0 \in \mathbb{R}^p, \int_{\mathbb{R}^p} uu^T K_p[u]du = I_p,$$

$I_p$ being the $p \times p$ identity matrix [15]. We can view $\hat{f}(x)$ as a mixture of densities centered at the observations $x_1, x_2, \ldots, x_n$. The bandwidth matrix $H$ needs to be selected by the user. Changing the bandwidth matrix has a large impact on the resulting estimated density, and is often chosen to suit the problem at hand. An example of kernel density estimation is shown in figure 2.8.

One common way to select the bandwidth matrix is to minimize the asymptotic mean integrated squared error

$$\text{AMISE} = \frac{1}{nh^p} \int K(u)^2 du + \frac{h^4}{4} \int \{\text{tr}[AA^T \nabla^2] f(u)\} du, \tag{2.32}$$

where $H = hA$ with $A$ having a unit determinant [15]. There is no closed form solution, but (2.32) shows that the bandwidth $h$ should be $\mathcal{O}(n^{-4/(p+4)})$. When using a Gaussian kernel on an multivariate normal distribution, the optimal bandwidth matrix is diagonal with elements

$$H = \left(\frac{4}{p+2}\right)^{1/(p+4)} \Sigma^{\frac{1}{2}} n^{-1/(p+4)}.$$

For $p = 2$, the estimated bandwidth matrix

$$\hat{H} = \hat{\Sigma}^{\frac{1}{2}} n^{-1/(p+4)} \tag{2.33}$$

is known as Scott's rule of thumb [16].

## 2.8 Watershed segmentation

Watershed segmentation is an image processing technique commonly used to separate objects in an image. Assuming a greyscale image it treats it like a topological map where intense areas are considered to have large altitude [17]. Using an estimated gradient the algorithm finds the local minima in the image, and gradually ”flood” the topography with water and mark the lines where the filled basins meet. The watershed segmentation applied to an estimated probability density function is shown in figure 2.8.

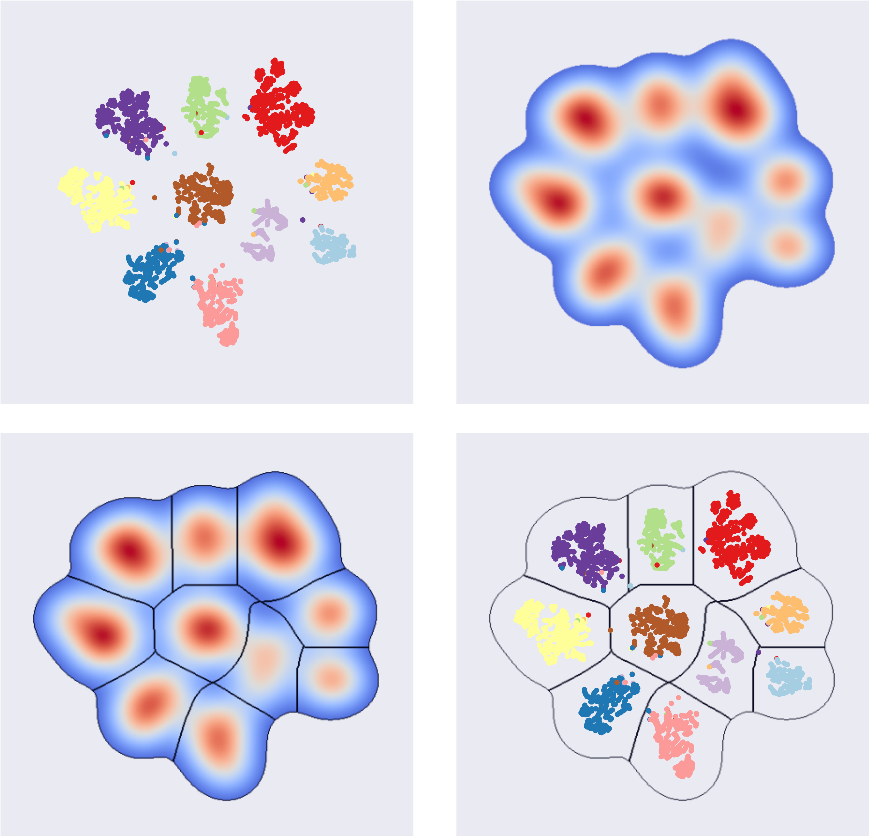


Figure 2.8: Clustering the two-dimensional t-SNE embedding. *Top left*: t-SNE embedding of simulated data, color coded by actual behavior. *Top right*: Kernel density estimation of the t-SNE embedding. *Bottom left*: Watershed segmentation applied to the kernel density estimation. *Bottom right*: Contours of the watershed segmentation with the t-SNE clusters shown inside.

# Chapter 3

# Methodology

The starting point of the analysis is a set $\mathbf{Y} = \{Y_d : d = 1, 2, \ldots, D\}$, where each set $Y_d$ is again a set of independent time series, $Y_d = \{y_1^d(t), y_2^d(t), \ldots, y_n^d(t) : t \in \{t_1, t_2, \ldots, t_{m_d}\}\}$. Each $Y_d$ represents an animal for which $n$ time series are collected tracking parts of its movements. Recording frequency is the same across animals. The goal of the analysis is to cluster these time points into distinct distinguishable actions. Looking at detrended motion recordings as periodic signals, we extract information about the periodicity in the movements at a range of frequencies through time frequency analysis. These extracted features are embedded into two dimensions, where we can use image segmentation techniques on an estimated two dimensional probability distribution.

## 3.1 Feature extraction

### 3.1.1 Detrending

The first step is to analyse each time series separately, modifying them and thus creating $D \times n$ features which we will extract the behavioral information from. Let $y_t = \{y_1, y_2, \ldots, y_m\} = y_i^d(t)$ for some $d \in \{1, 2, \ldots, D\}$ and $i \in \{1, 2, \ldots, n\}$ be one such series.

We detrend the data as $y_t = s_3(t) + x_t$, $s_3(t)$ being the non-linear trend and $x_t$ the detrended time series. The cubic spline $s_3(t)$ is found using least square regression as in equation (2.26) with equally spaced internal knots $\xi_1, \xi_2, \ldots, \xi_M$. We choose the knots to have a fixed frequency, i.e., we choose $\Delta\xi = \xi_{i+1} - \xi_i, i = 2, \ldots, M$. Figure 3.1 shows an example of a detrended time series using cubic splines with interior knots every 240th time point.

### 3.1.2 Time frequency analysis

To make the extracted features comparable, we normalize the time series by dividing it by its standard deviation

$$\sigma_x = \frac{1}{m} \sum_{t=1}^{m} \left( x_t - \frac{1}{m} \sum_{i=1}^{m} x_i \right)^2 .$$

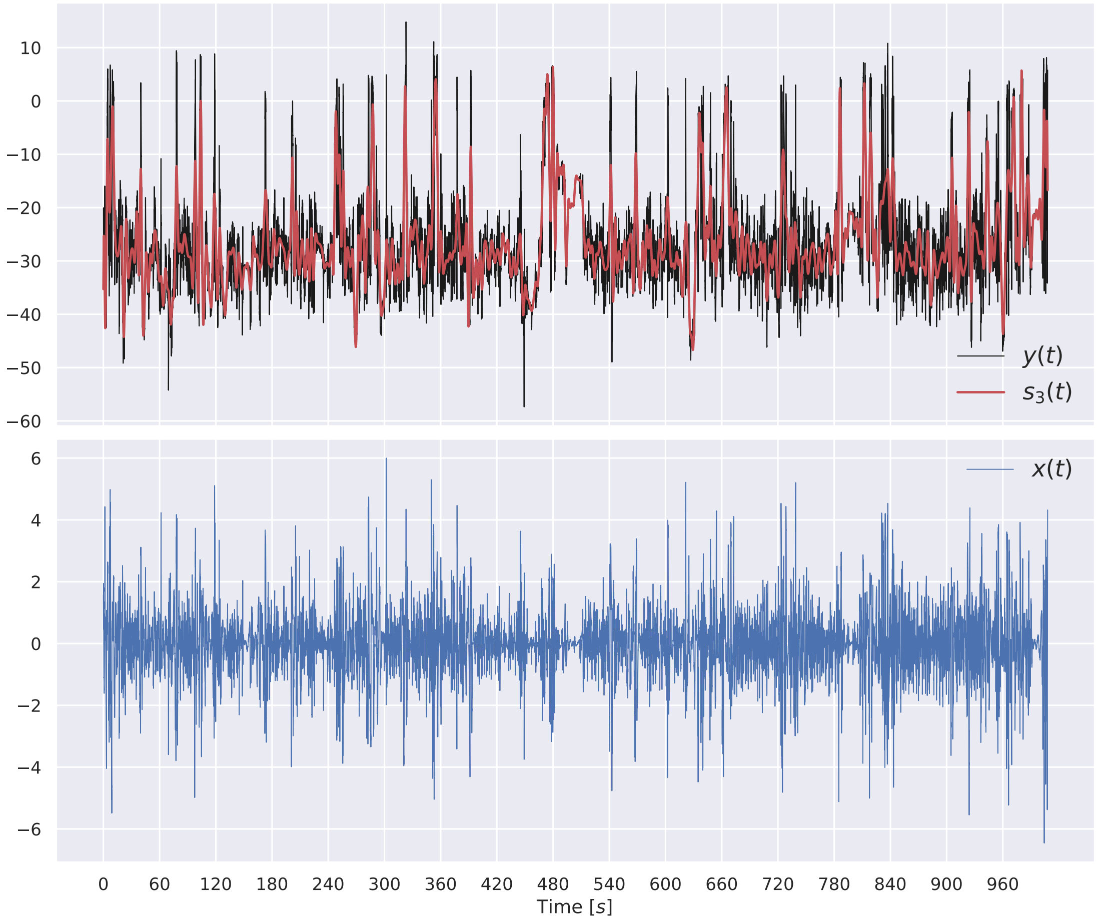


Figure 3.1: Detrended time series using cubic spline regression. The top figure shows the original time series in black with the fitted trend in red. The bottom figure shows the detrended time series.

The detrended time series contain information about how the movements vary around the trend. We assume this movement is repetitive in nature, and can be interpreted as a signal. By applying time frequency analysis to the detrended time series, we obtain additional information about how the frequencies of these movements vary over time. In turn this can contribute to detecting the behavioural clusters.

As we are unaware of at which scales these movements occur beforehand, we opt for a continuous wavelet transform as opposed to the short-time Fourier transform. Let $J$ be the number of predefined frequency scales, ranging from $\omega_{\min}$ to $\omega_{\max}$. The scales $s_1, s_2, \ldots, s_J$ in equation (2.23) are chosen as fractional powers of two—as suggested in [10]—e.g.,

$$s_j = \frac{2^{(j-1)\delta j}}{\omega_{\max}}, \quad j = 1, 2, \ldots, J, \tag{3.1}$$

$$\delta j = \frac{1}{J-1} \log_2 \left( \frac{\omega_{\max}}{\omega_{\min}} \right). \tag{3.2}$$

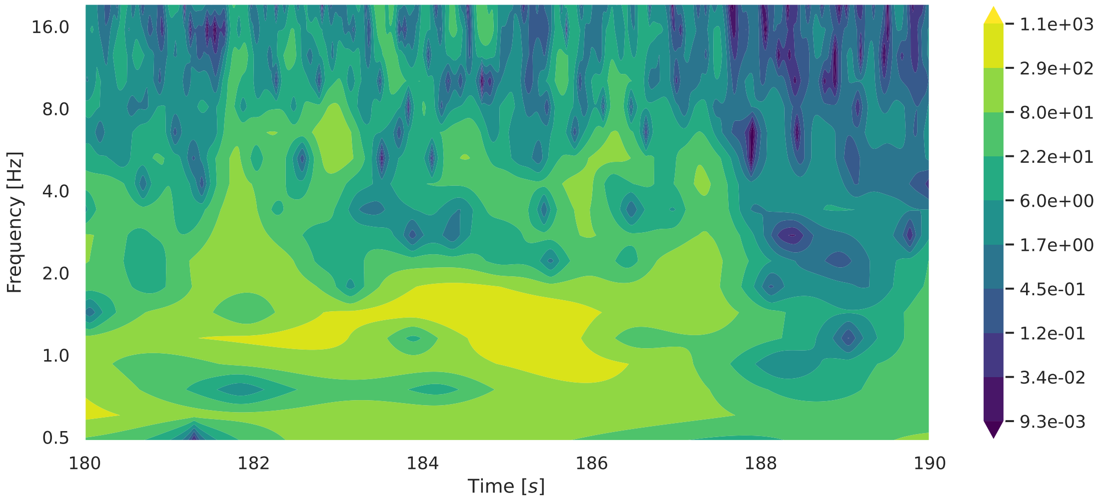


Figure 3.2: Scaleogram of the continuous wavelet transform performed on the detrended time series shown in figure 3.1. The figure shows how the power at different frequency scales changes over time.

Choice of mother wavelet can for instance be based on expected features in the time series [10]. Using equation (2.23) we compute the discrete continuous wavelet transform $\tilde{f}_m(s,\tau)$, and in turn the scaled scaleogram

$$\frac{1}{s_j}\left|\tilde{f}_m(s_j,t)\right|^2, \quad j=1,2,\ldots,J,\ t=1,2,\ldots,m. \tag{3.3}$$

The scaling is done to keep the power comparable across the varying scales [18]. An example plot of such a scaleogram is shown in figure 3.2. To make the estimated power comparable we take the square root before proceeding

## 3.2 Manifold embedding

After performing the time-frequency analysis on all time series for all animals, we concatenate the results into a new feature vector. For animal $d$, we have $m_d$ time points, each containing information about the $n$ trend values, together with the $n \cdot J$ frequency powers. The concatenated feature vector thus have dimension $(m_1 + m_2 + \ldots + m_D) \times (n \cdot (J+1))$. To cluster these time points into behaviours, we first reduce the feature dimension. It is logical that there should exist strong correlations between the various frequencies.

As a first step in the dimensionality reduction we use principal component analysis, keeping features explaining at least 95% of the variance. This both reduces further computational complexity, and removes noise. To embed the data into two dimension, we apply t-distributed stochastic neighbor embedding. It has one hyper parameter, the perplexity (2.30), which is chosen by inspecting some embeddings with values between 10 and 200. As t-SNE has quadratic memory complexity, we only embed as subset of the data. The subset is chosen through downsampling at a predefined frequency such that the number of training points is below 50000.

# Chapter 4

# Discussion

## 4.1 Simulated Data

To exemplify and visualize the impact of various implementation choices, we make use of simulated data inhibiting the properties we assume to be true for animal behaviour. We simulate one animal with five recorded time series features, each over ten minutes with a recording frequency of 120Hz. We create ten distinct behaviors. For each behavior and feature, the underlying feature is generated as a superposition of four sine waves

$$\sum_{i=1}^{4} a_i \sin(\omega_i 2\pi t).$$

The $10 \times 5 \times 4$ frequencies $\omega$ are randomly generated uniformly between 0.5Hz and 20Hz. Similarly, the amplitudes $a$ are randomly generated as a lognormal distribution with parameters $\mu_a = 1, \sigma_a = 0.5$. A Gaussian noise is added with $\mu_n = 0, \sigma_n = 0.2$. The length of the behaviors are modelled to follow an exponential distribution with mean $\lambda = 3$. This is equivalent to generate $10 \cdot 60/3 = 200$ time points uniformly over $[0, 600]$, each interval assigned a random behavior. Figure 4.1 shows one such feature, colored by the underlying behaviour at the first 50 seconds.

## 4.2 Benefits of t-SNE

There are many popular methods for embedding a high dimensional dataset into a lower dimensional space. Principal component analysis is discussed in section 2.6.1, and is a good method to use for variance reduction. It is also commonly used for visualizing data in two or three dimensions. A large caveat is its linearity in the original features, which makes it unable to capture non-linear effects in the data. Another common method is multidimensional scaling (MDS) [12]. It preserves the high dimensional distances as euclidean distances in the low dimensional space. MDS emphasises the global structure in the data, as all points contribute significantly to the loss function. There exists multiple methods extending the MDS methodology, making it more suitable for finding local and non-linear structures in the data. One of these methods is ISOMAP, which first creates a neighborhood graph, on which it computes geodesic distances

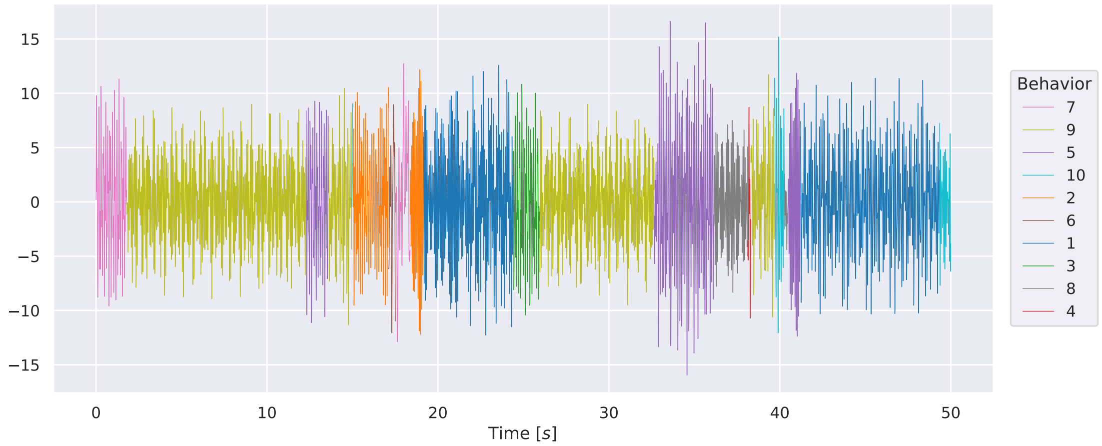


Figure 4.1: Simulated feature vector showing changes in behavior during the first 50 seconds.

between points as the shortest path in the graph [19]. Using these distances it finds the low dimensional embedding via MDS.

As previously stated in section 2.6.2, t-SNE preserves conditional probabilities created from the high dimensional euclidean distances [14]. Due to this it places emphasis on the local structure, and is a natural choice when the goal is to visualize high dimensional clusters in two dimensions. Figure 4.2 illustrates these differences. All methods are able to separate the clusters to some degree, bu t-SNE clearly stands out in its ability to distinguish them.

## 4.3 Tuning the perplexity parameter

The perplexity parameter from equation (2.30) plays a significant role in the t-SNE embedding [14]. It is closely related to the number of neighbors considered when creating the conditional probabilities (2.27). When the perplexity is small all points contribute ”equally little” to the cost function, meaning the algorithm is not able to detect anything. When the perplexity is large it attributes significance to more points, reducing the separation of clusters in the two dimensional embedding. Large datasets should therefore need larges perplexity values. Choosing it between 5 and 50 is suggested. Figure 4.3 shows how the perplexity parameter affects the clustering of the simulated data. Perplexity values 30 and 50 give good results. Note that only 3600 points were used calculating the embedding.

## 4.4 Pre embedding with PCA

We could just embed the points directly into two dimensions through t-SNE alone. However, due to memory complexity limitations in the t-SNE algorithm, we need to select a subset data points which we use to create the embedding. These points can be thought of as the training points in the clustering algorithm. How do we then classify the remaining data? We will discuss two solutions to this problem. Both solutions involves reducing the dimension of the data linearly through principal component analysis prior to the t-SNE embedding. By only including the principal components explaining at least 95% of the variance, we reduce a significant amount of

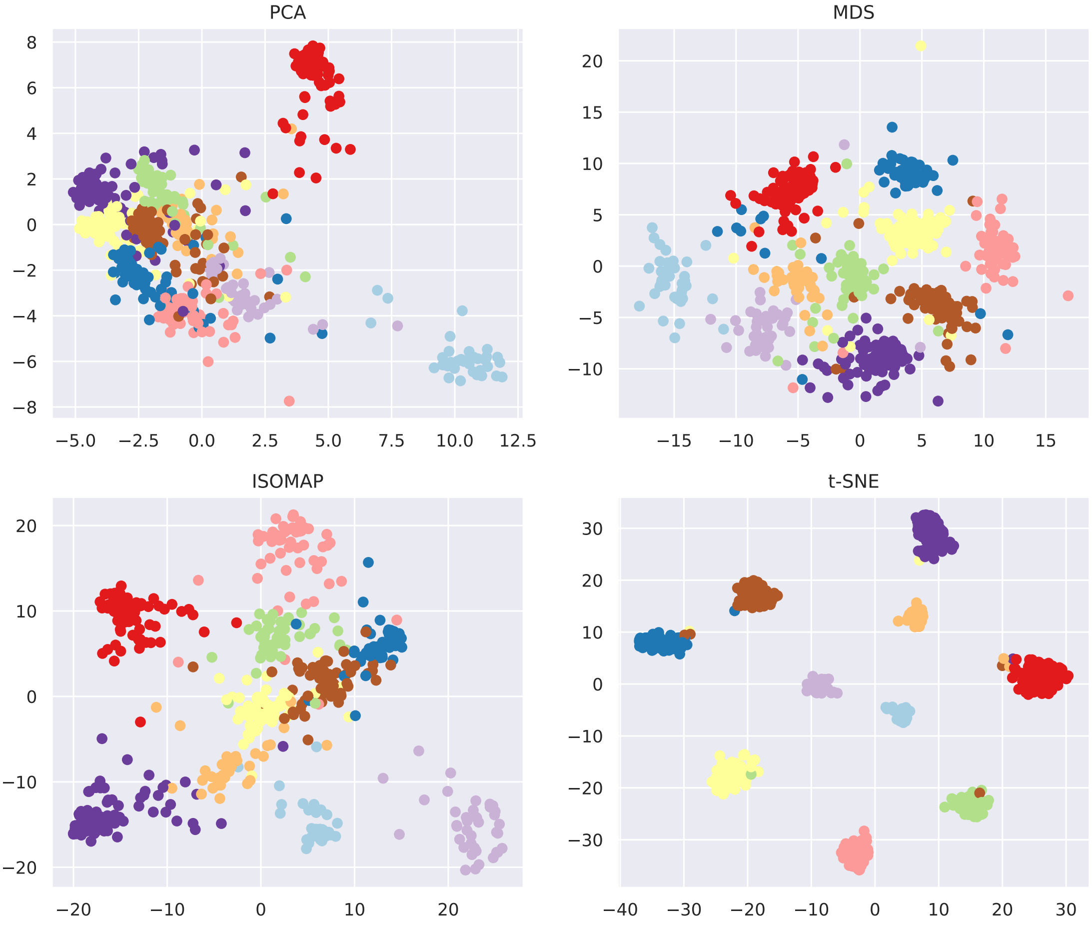


Figure 4.2: Two dimensional embedding of the standardized simulated data using different dimensionality reduction methods. *Top left:* Principal component analysis. *Top right:* Multi-dimensional scaling. *Bottom left:* ISOMAP using $n = 20$ neighbors construction the graph. *Bottom right:* t-SNE with perplexity 30.

noise. At the same time we make it easier to compare data points through euclidean distances in the PCA space.

One idea is to apply kernel density estimation followed by watershed segmentation only on the training points. I.e., we classify the training points first. The remaining points are given the same label as their closest neighbor in the PCA space, using euclidean distance as the metric.

A second idea is to embed all points into the two dimensions provided by t-SNE on the training data. In this way, all points are contributing in the kernel density estimation, and subsequently in creating the behavioral regions. This choice of direction has a significant impact on the resulting classification. As shown in figure 4.4, using all points creates more peaks in the kernel density estimation, leading to a larger number of discovered behaviors.

As already mentioned, the size of the training data is very limited in the t-SNE algorithm. One argument for embedding before the kernel density estimation is that all the data is used for detecting behaviors. When the data samples grow in size, an increasing proportion is left out of

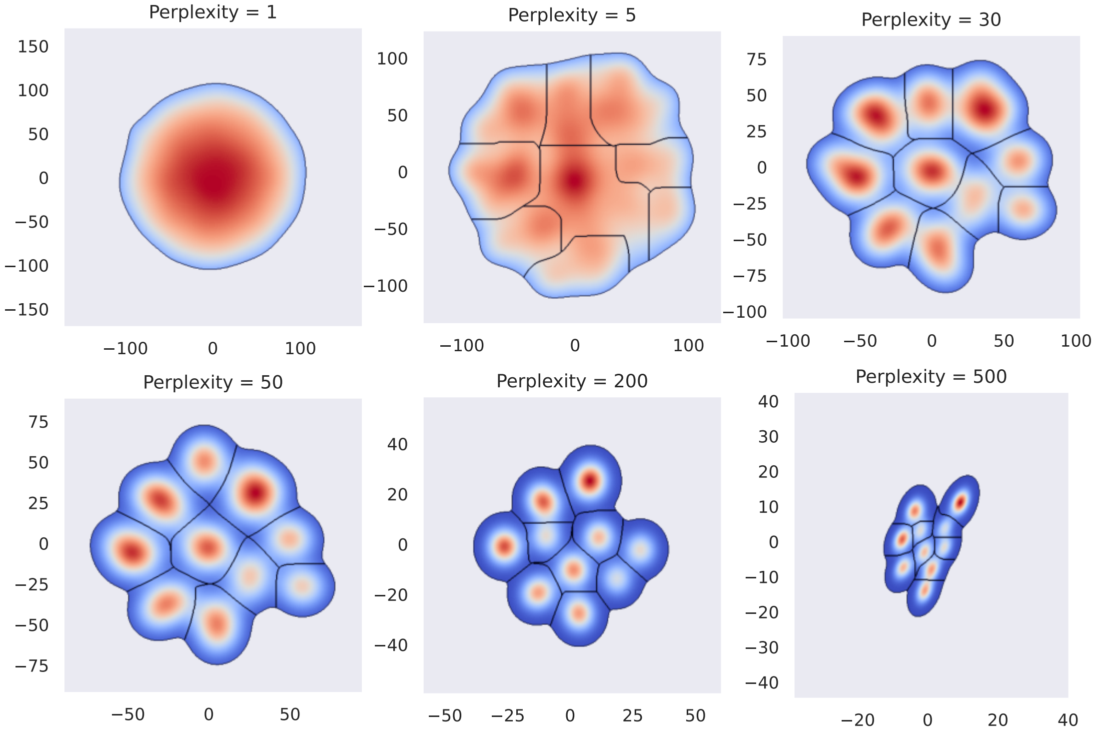


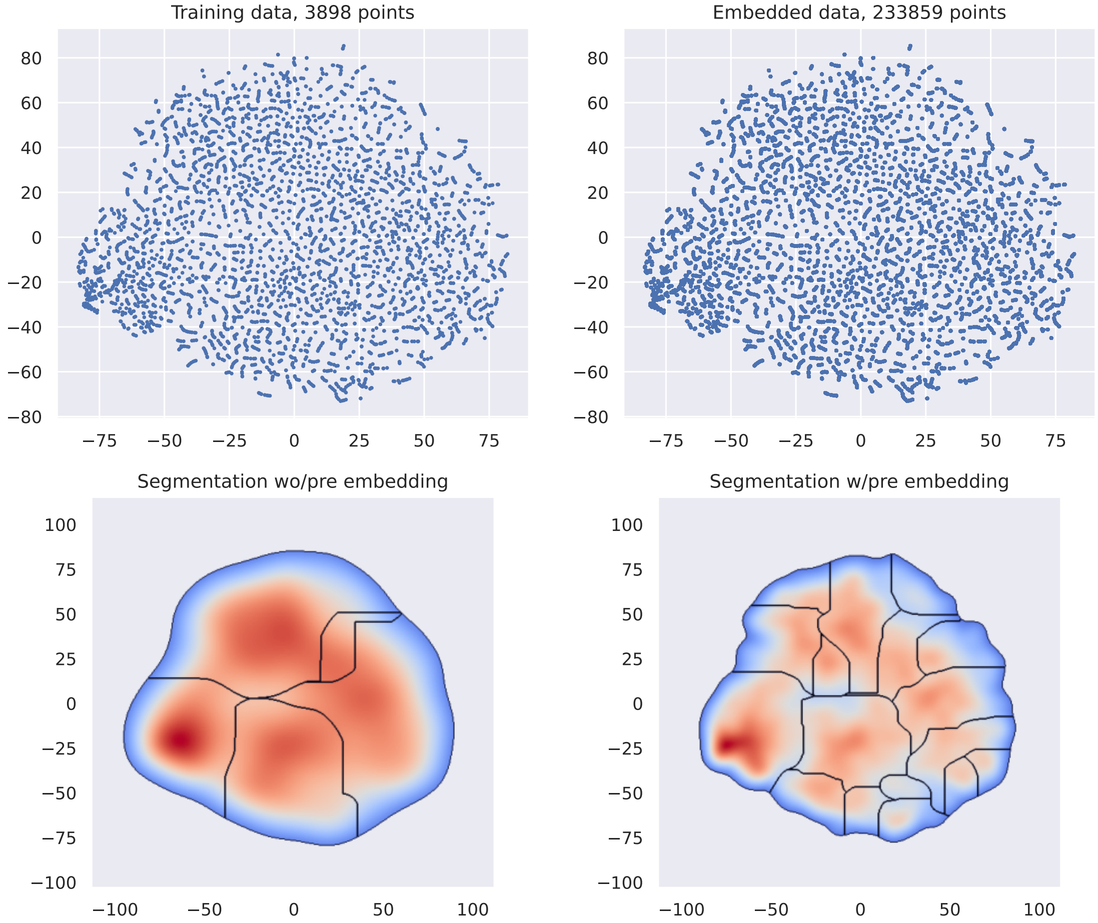
Figure 4.3: Clustering of the simulated data varying the perplexity parameter in t-SNE. The bandwidth for the kernel density estimation is 0.2 for all embeddings.

the training data.

## 4.5 Scaling the spectrogram

It is the power spectrum provided by the wavelet transform which contains all the information we use to separate the behavioral clusters. For each original feature we extract power at a range of frequencies, as shown in the scaleogram 3.2 As these are the features which is to be reduced into two dimensions, we need to prepare them as best as possible for the dimensionality reduction techniques. Firstly we want all frequencies to contribute equally. This creates the need for scaling across frequencies, as the wavelet transform creates larger spectral peaks for lower frequencies [18] [2]. A suggested remedy for this is simply to divide the (squared) spectrum by its corresponding scale [18]. Secondly all features should have equal contributions. This is solved by standardizing the spectrum, which is commonly done before applying principal component analysis anyway.

Finally there remains the question of whether to take the square root of the wavelet spectrum, i.e, should we use $|\tilde{f}_n(s,\tau)|^2$ or $|\tilde{f}_n(s,\tau|$ as a starting point for the dimensionality reduction. Intuitively the squared spectrum discriminates more between frequencies with high and low power. When standardizing the power spectrum before PCA, the original magnitudes are unimportant. However, without standardization taking the square root is essential in keeping the powers comparable, as shown in figure 4.5. The figure shows that standardizing groups the clusters more together in the t-SNE embedding, making it harder to distinguish very dissimilar clusters.


Training data, 3898 points

Embedded data, 233859 points

Segmentation wo/pre embedding

Segmentation w/pre embedding

Figure 4.4: Plots showing the t-SNE embedding and corresponding watershed segmentation on data from two rats, downsampled at 2Hz. *Left:* Only the downsampled training data is used. *Right:* All data is embedded into the two dimensional plane, using the embeddings corresponding to the nearest neighbor in the reduced PCA space.

When tested on actual data, figure 4.6 shows that omitting the standardization creates several outliers in the t-SNE embedding, without providing clearer separation of clusters.

## 4.6 Effects of downsampling

## 4.7 Tuning the bandwidth matrix

The kernel density estimation (2.31) plays a significant role in final the clustering. It smooths the two dimensional embedding creating local peaks which we interpret as individual behaviors. The degree of smoothing is determined by the bandwidth $2 \times 2$ matrix $H$ briefly discussed in section 2.7. Several ways of choosing this parameter have been discussed in literature, most of them based on minimizing the asymptotic mean integrated squared error [15]. As our goal is visualizing clusters, it is better to test several different values, choosing the one which provide the

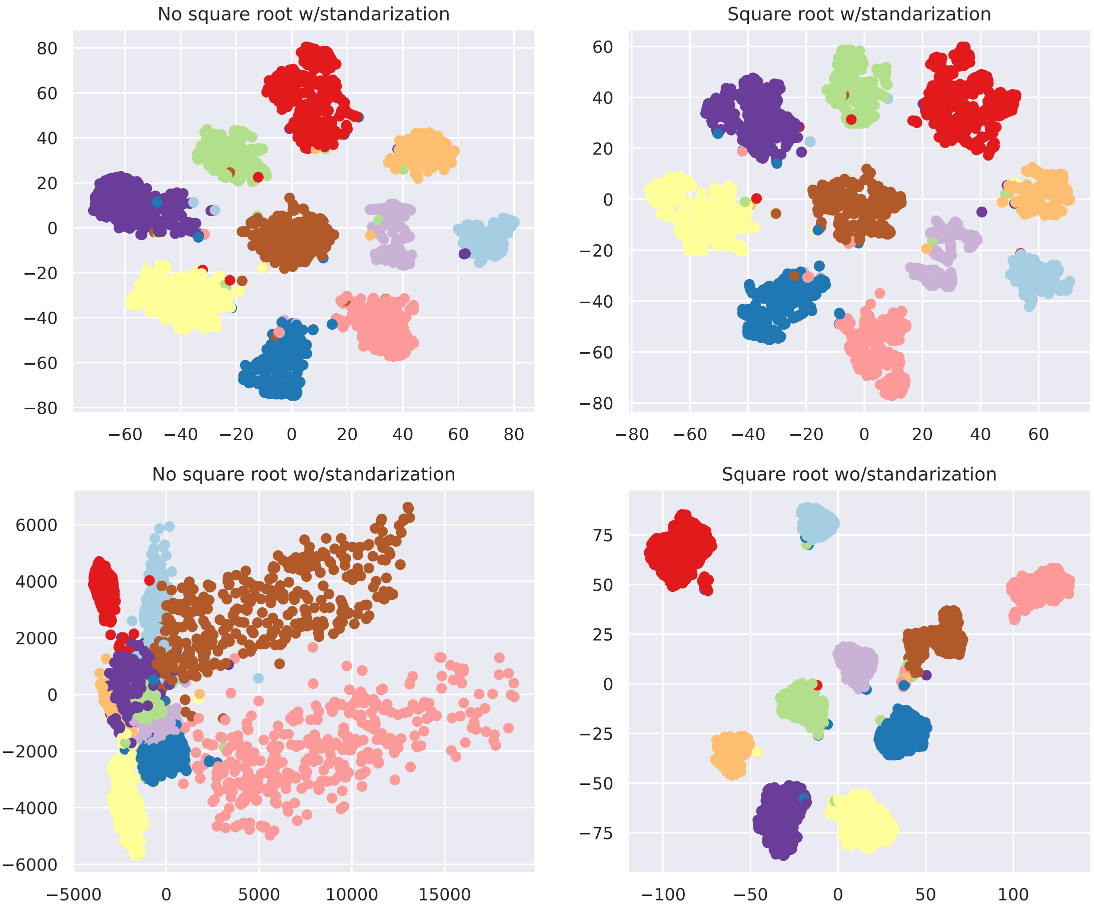


Figure 4.5: t-SNE embedding of the simulated data varying the scaling of the wavelet power spectrum. Points are color coded by their underlying true behavior.

best visualization. To limit the possibilities, we let the scaling be identical in both dimensions, letting $H$ be a diagonal matrix with identical elements.

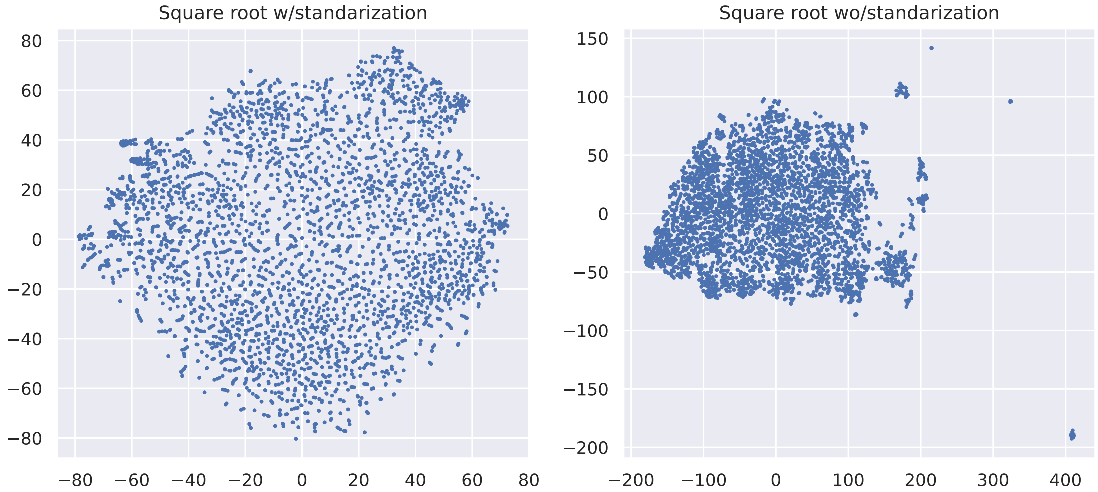


Figure 4.6: t-SNE embedding of motion recordings of two rats downsampled at 2Hz. On the left the wavelet spectrum is standardized before applying PCA. On the right no standardization is performed.

# Chapter 5

# Results

## 5.1 Data

We test the methodology on actual data collected by Kavli Institute for Systems Neuroscience at the Norwegian University of Science and Technology, specifically the Whitlock group. The data can be downloaded through the following figshare. A detailed description of the data collection can be found in Mimica et al. 2022 [5]. They also perform a behavioral clustering on the data, providing an opportunity to compare results.

Seven rats are recorded at different times and in different conditions, providing 57 recordings with a total of 8745586 distinct time points. Of these 410560 time points include NaN-values in one or more of the features, leaving 8335026 time points remaining eligible for analysis. A capture frame rate of 120Hz gives an average time series length of approximately 20 minutes. Each recording contains motion data seven postural parameters, running speed, neck elevation, back angles and head direction.

## 5.2 Parameters

The detrending is done with cubic spline regression with knots spaced at 0.5Hz. We then apply the continuous wavelet transform with a Morlet mother wavelet at 18 frequencies between 0.5Hz and 20Hz, chosen as in equation (3.1). This created 133 features, which was reduced to 83, using principal component analysis keeping the components explaining at least 95% of the total variance. The data is downsampled at 0.5Hz, creating a subset of 34730 time points which is used in the t-SNE embedding. The perplexity parameter is set as 30, based on visual inspection of the final segmentation, shown in figure 5.1. Finally, the bandwidth matrix in the Gaussian kernel density estimation is chosen using Scott's rule (2.33).

## 5.3 Clustering

The final heat map showing the watershed segmentation is shown in figure 5.2. It shows the estimated kernel density of the t-SNE embedding, divided into 96 regions corresponding to a distinct behavior. This is over twice as many detected in Mimica et. al [5]. One reason for this

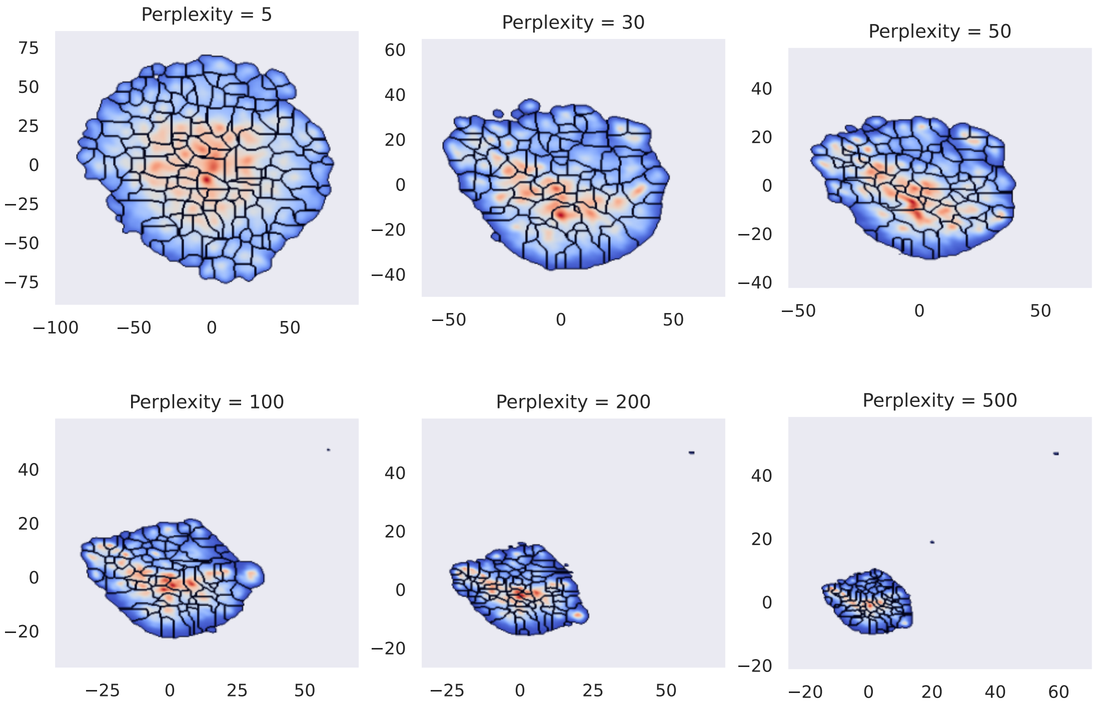


Figure 5.1: Tuning the perplexity parameter in t-Stochastic neighbor embedding on the complete data set.

could be the use of pre-embedding before the density estimation, which they do not do. There is also a large difference in the number of principal components needed to explain over 95% of the variance after the time frequency analysis. They alter the wavelet transform output to a larger degree before applying PCA, which I assume lead to this discrepancy.

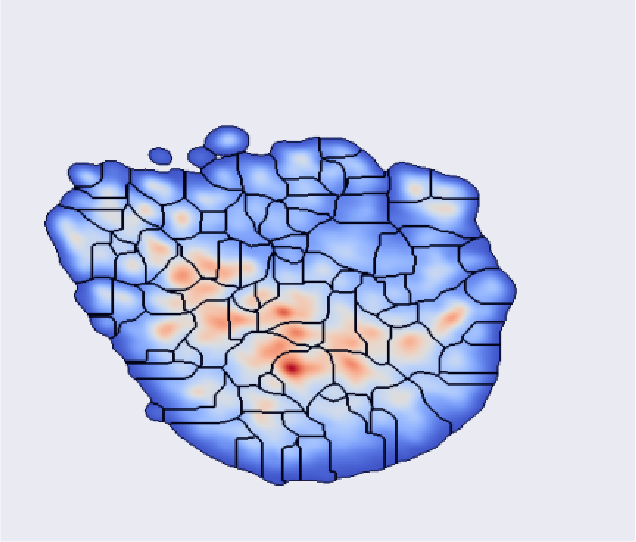


Figure 5.2: Final behavioral segmentation on the complete data set. The t-SNE plane is divided into 96 regions each corresponding to a behavior. The underlying heat map shows the estimated probability density of the embeddings.

# Chapter 6

# Conclusions

Using time frequency analysis and dimensionality reduction techniques, we are able to group animal motion recording segments into distinct behaviors. On real motion recordings of seven rats, the algorithm finds 96 such behaviors. As we do not have access to the video recordings themselves, it is impossible to say what these behaviors represents. There are many choices throughout the methodology which could be further investigated, especially how the data is reduced to two dimensions. An analysis of the found behaviors should also be performed: How long do the animals stay in the different behaviors? Does there exists patterns in the changes in behavior? How is the distribution of time spend in each behavior? How does the sample size affect the number of detected behaviors? What is the effect of adding new motion features? Luckily I have a master thesis to write in which I can explore these questions.

# Appendix A

# Code

The main clustering pipeline is included as below. All code for producing figures and simulations in this project can be found at this github repository.

```
import numpy as np
import pickle as pkl

from helper_functions import (
        pickle_relevant_features, spline_regression,
        plot_scaleogram, estimate_pdf,
        get_watershed_labels, assign_labels,
)

import pycwt as wavelet
from pycwt.helpers import find

from sklearn.decomposition import PCA, IncrementalPCA
from sklearn.manifold import TSNE

from scipy.spatial.distance import cdist


class BehavioralClustering():
    """
    Wrapper class for clustering distinct behaviours
    in rats from recorded postural dynamics.

    Attributes:
        train_file_names (List(str)): Name of raw data
            to be used in the clustering
        original_file_path (str): Location of the training
            files.
```

```
extracted_file_path (str): Location of the pickled
    files containing only relevant features.
raw_features (List(np.ndarray)): The features relevant to
    the analysis, straight from the input files.
knot_frequency (float): Interior spacing of the knots
    used in the spline regression
spline_dim (int): Dimension of the regression spline
    used in detrending the data
trend (list(np.ndarray)): The trend found in the
    time series by spline regression
data_detrended (list(np.ndarray)): The detrended
    time series.
feature_names (list(str)): Names of the features
    used
n_features (int): Number of different recorded
    features, i.e., number of movements recorded
    per animal.
capture_framerate (int): Frequency (Hz) of the
    recorded postural time series.
dt (float): Time interval in the time series
used_indices (list(np.ndarray)): Indices of non-NaN values
num_freq (int): Number of dyadically spaced frequency
    scales used in the CWT
min_freq (float): Minumum frequency scale used in
    the CWT
max_freq (float): Maximum frequency scale used in
    the CWT
dj (float): The number of sub-octaves per octave,
    logarithmic spacing for the scales used
    in the CWT
mother (str): Mother wavelet used in the CWT
power (list(np.ndarray)): The computed power
    spectrum from the cwt
features (list(np.ndarray): Features extracted
    from the time frequency analysis
scales (np.ndarray): Stores the cwt scales
freqs (np.ndarray): Stores the cwt frequencies
var_pca (np.ndarray): Percentage ratios of the
    explained variance in the principal component
    analysis
n_pca (int): Number of principal components
    explaining more than 95% of the data
fit_pca (np.ndarray): The transformed (reduced)
    features using principal component analysis
ds_rate (int): Downsampling frequency in [Hz],
    used as t-SNE memory complexity is O(n^2)
perp (float): Perplexity parameter used in t-SNE
embedded_train (np.ndarray): Two dimensional embedding
    obtained by t-SNE on the downsampled data
```

```
        embedded (np.ndarray): The low dimensional embedding
            of all points
        tsne_ind (np.ndarray): Indices of time points used
            when finding the t-SNE embedding
        kde (np.ndarray): Kernel density estimation of
            the t-SNE embedding
        border (int): Border around the t-SNE embedding
            for improved watershed segmentation, and
            visualization
        grid (list(np.ndarray)): Grid on which the
            kernel density estimation is applied
        bw (float/str): Bandwidth method used in the kernel
            density estimation
        ws_labels (np.ndarray): Assigned clusters
            found by watershed segmentation
        beh_labels (np.ndarry): Final classification of
            the non-nan time points
    """

    def __init__(self):
        self.train_file_names = []
        self.original_file_path = None
        self.extracted_file_path = None
        self.raw_features = []
        self.knot_frequency = 0.5
        self.spline_dim = 3
        self.trend = []
        self.data_detrended = []
        self.feature_names = ["exp0", "exp1", "exp2",
                              "speed2", "BackPitch",
                              "BackAzimuth", "NeckElevation"]
        self.n_features = len(self.feature_names)
        self.capture_framerate = 120
        self.dt = 1/self.capture_framerate
        self.used_indices = []
        self.data = []
        self.num_freq = 18
        self.min_freq = 0.5
        self.max_freq = 20
        self.dj = 1/(self.num_freq - 1)*np.log2(
                self.max_freq/self.min_freq)
        self.mother = "morlet"
        self.power = []
        self.features = []
        self.scales = np.zeros(self.num_freq)
        self.freqs = np.zeros(self.num_freq)
        self.var_pca = None
        self.n_pca = None
        self.fit_pca = None
```

```
        self.ds_rate = 2
        self.perp = 30
        self.embedded_train = None
        self.embedded = None
        self.kde = None
        self.border = None
        self.grid = None
        self.bw = ”scott”
        self.ws_labels = None
        self.beh_labels = None


    def remove_nan(self):
        ”””
        Removes each time point where one or more value
        in the time series contain a NaN value.
        Stores the new time series data in
        an attribute, together with the used row indices.
        ”””

        # Iterate over animals
        for i in range(len(self.raw_features)):
            # Find the indices of usable rows
            self.used_indices.append(~np.isnan(
                self.raw_features[i]).any(axis =1))
            # Filter the data and add to an attribute
            self.data.append(self.raw_features[i][self.used_indices[i]])



    def detrend(self):
        ”””
        Detrends the time series individually
        using spline regression, and stores the
        trend and detrended data as attributes
        ”””

       # Iterate over animals
        for d in range(len(self.data)):
            # Perform spline regression on each time series
            trend = [spline_regression(y, self.spline_dim,
                        self.capture_framerate,
                        self.knot_frequency) for y in self.data[d].T]
            self.trend.append(np.array(trend).T)
            self.data_detrended.append(self.data[d] - self.trend[d])
            # Normalize data
            std = np.std(self.data_detrended[-1], axis = 0)
            self.data_detrended[-1] = self.data_detrended[-1] / std
```

```
def time_frequency_analysis(self):
    """
    Perform time frequency analysis
    using the continuous wavelet transform on the
    detrended time series data. The power spectrum
    is computed using the pycwt python package,
    and divided by the scales. Taking the square root,
    centering and rescaling by the trend standard
    deviation, the extended time series are concatenated
    with the normalized trend data creating new
    features. We store the power matrix, scales and
    frequencies to be used for plotting scaleograms.
    """

    # Iterate over animals
    for d in range(len(self.data)):
        # Create storage for new feature vector
        x_d = np.zeros(shape = (len(self.data[d]),
                                self.n_features*
                                (self.num_freq + 1)))

        # Iterate over raw features
        for i in range(self.n_features):
            # Apply cwt
            wave, scales, freqs, coi, fft, fftfreqs = wavelet.cwt(
                    self.data[d][:,i], self.dt, self.dj, 1/self.max_freq,
                    self.num_freq - 1, self.mother)
            # Store the frequencies and scales
            if scales.all() != self.scales.all():
                self.scales = scales
                self.freqs = freqs
            # Compute wavelet power spectrum
            power_i = np.abs(wave)**2
            # Normalize over scales (Liu et. al. 07)
            power_i /= scales[:, None]
            # Take the square root to ...
            power_i = np.sqrt(power_i)

            # Center and rescale trend
            trend = self.trend[d][:,i]
            trend_std = np.std(trend)
            trend = (trend - np.mean(trend)) / trend_std

            # Center and rescale power spectrum
            power_i = (power_i - np.mean(power_i)) / trend_std
            # Store new features
            x_d[:,i] = trend
            x_d[:,self.n_features + i*self.num_freq:self.n_features +(i + 1)*
```

```
                    self.num_freq] = power_i.T

                self.features.append(x_d)


    def standardize_features(self):
        """
        Standardizes the features extracted via CWT such that they can
        be used in PCA, i.e., with mean zero and variance one.
        """

        # Iterate over animals
        for i in range(len(self.data)):
            self.features[i] = ((self.features[i] - np.mean(self.features[i],
                                                             axis = 0)) /
                                np.std(self.features[i], axis = 0))


    def pca(self):
        """
        Compute the principal components explaining 95% of
        the variance in features extracted via CWT.
        Stores the new reduced features as an attribute.
        """

        # Concatenate features
        features = np.concatenate(self.features, axis = 0)

        # Find principal components
        pca = PCA()
        pca.fit(features)

        # Find number of features explaning 95% of the variance
        self.var_pca = pca.explained_variance_ratio_
        self.n_pca = np.argmax(np.cumsum(self.var_pca) > 0.95) + 1

        # Apply the transformation using the sufficient
        # number of principal components
        pca = PCA(n_components = self.n_pca)
        self.fit_pca = pca.fit_transform(features)


    def tsne(self):
        """
        Embed the principal component scores onto a
        two dimensional manifold using t-SNE.
        Previous to the dimensionality reduction
        we downsample to reduce memory complexity in
        the algorithm.
```

```
        """

        # Downsample the principal component scores
        self.tsne_ind = np.arange(0, len(self.fit_pca),
                        int(self.capture_framerate / self.ds_rate))
        train = self.fit_pca[self.tsne_ind,:]

        # Perform t-SNE
        self.embedded_train = TSNE(n_components = 2,
                                   perplexity = self.perp,
                                   init = "pca").fit_transform(train)


    def pre_embedding(self):
        """
        Embeds all data points into the t-SNE plane
        by choosing the components of the nearest
        neighbor (in the training set) by euclidean
        distance in the PCA space.
        """

        # Principal component scores for
        # points used to find t-SNE embedding.
        pca_train = self.fit_pca[self.tsne_ind,:]

        # Create storage for embeddings
        self.embedded = np.zeros(
                shape = (len(self.fit_pca), 2))

        # Iterate over all time points
        for i in range(len(self.fit_pca)):
            # Find closest time point in PCA space
            dist = cdist(self.fit_pca[i,:][np.newaxis,:],
                         pca_train)

            # Choose embedding corresponding to this
            # time point
            self.embedded[i] = self.embedded_train[dist.argmin()]


    def kernel_density_estimation(self, pixels):
        """
        Perform kernel density estimation on the t-SNE
        embedding, estimating the pdf using Gaussian kernels.

        Arguments:
            pixels (int): Pixelation aloach each axis which
                the pdf is computed on
        """
```

```
        self.border = np.max(np.abs(self.embedded))/5

        # Outer border set for better visualizations
        self.kde, self.grid = estimate_pdf(
                self.embedded, self.bw, self.border, pixels)


    def watershed_segmentation(self):
        """
        Perform watershed segmentation on the
        kernel density estimation pdf.
        """

        self.ws_labels = get_watershed_labels(self.kde)


    def classify(self):
        """
        Assigns a behavioral action label to
        all (non-nan) time points in the data set.
        """

        # Classify the embedded points
        self.beh_labels = assign_labels(self.embedded,
                                        self.ws_labels,
                                        self.grid)


    def set_original_file_path(self, original_file_path):
        self.original_file_path = original_file_path


    def set_extracted_file_path(self, extracted_file_path):
        self.extracted_file_path = extracted_file_path


    def pickle_relevant_features(self, file_names):
        """
        Extract the relevant raw features from the original
        data, pickle it, and dump it in the location
        specified in the extraced_file_path attribute

        Parameters:
            file_names (List(str)): File names of original data
        """
        for name in file_names:
            pickle_relevant_features(self.original_file_path + name,
                                     self.extracted_file_path +
```

```
                                name.split(".")[0] +
                                "_extracted.pkl")


def load_relevant_features(self, train_files):
    """
    Loads the pickled raw relevant features into the
    instance attribute to be used for training.
    Stores number of features as attribute.

    Parameters:
        train_files (List(str)): Names of training files
    """

    self.train_file_names = train_files
    for filename in train_files:
        with open(self.extracted_file_path +
                  filename.split(".")[0] + "_extracted.pkl",
                  "rb") as file:
            self.raw_features.append(pkl.load(file))

    # Store the number of features
    self.n_features = self.raw_features[0].shape[1]
```

# Bibliography

[1] J. Kain, C. Stokes, Q. Gaudry, *et al.*, "Leg-tracking and automated behavioural classification in drosophila," *Nature Communications*, vol. 4, no. 1, 2013. DOI: `10.1038/ncomms2908`.

[2] G. J. Berman, D. M. Choi, W. Bialek, and J. W. Shaevitz, "Mapping the stereotyped behaviour of freely moving fruit flies," *Journal of The Royal Society Interface*, vol. 11, no. 99, p. 20 140 672, 2014. DOI: `10.1098/rsif.2014.0672`.

[3] A. Wiltschko, M. Johnson, G. Iurilli, *et al.*, "Mapping sub-second structure in mouse behavior," *Neuron*, vol. 88, no. 6, pp. 1121–1135, 2015. DOI: `10.1016/j.neuron.2015.11.031`.

[4] J. D. Marshall, T. Li, J. H. Wu, and T. W. Dunn, "Leaving flatland: Advances in 3d behavioral measurement," *Current Opinion in Neurobiology*, vol. 73, p. 102 522, 2022. DOI: `10.1016/j.conb.2022.02.002`.

[5] "Behavioral decomposition reveals rich encoding structure employed across neocortex," *bioRxiv*, 2022. DOI: `10.1101/2022.02.08.479515`. eprint: `https://www.biorxiv.org/content/early/2022/02/10/2022.02.08.479515.full.pdf`. [Online]. Available: `https://www.biorxiv.org/content/early/2022/02/10/2022.02.08.479515`.

[6] W. W. Wei, *Time series analysis univariate and multivariate methods*, 2nd ed. Pearson, 2006.

[7] R. H. Shumway and D. S. Stoffer, *Time series analysis and its applications: With R examples*, 4th ed. Springer International Publishing, 2017.

[8] L. Cohen, "Time-frequency distributions-a review," *Proceedings of the IEEE*, vol. 77, no. 7, pp. 941–981, 1989. DOI: `10.1109/5.30749`.

[9] G. Kaiser, *A friendly guide to wavelets*. Birkhäuser, 1994.

[10] C. Torrence and G. P. Compo, "A practical guide to wavelet analysis," *Bulletin of the American Meteorological Society*, vol. 79, no. 1, pp. 61–78, 1998. DOI: `10.1175/1520-0477(1998)079<0061:APGTWA>2.0.CO;2`.

[11] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Springer, 2010.

[12] T. Hastie, J. Friedman, and R. Tisbshirani, *The elements of Statistical Learning: Data Mining, Inference, and prediction*, 2nd ed. Springer, 2017.

[13] I. T. Jolliffe, *Principal component analysis. 2nd ed.* Springer-Verlag, 2002.

[14] L. van der Maaten and G. E. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[15] J. S. Simonoff, *Smoothing methods in statistics*. Springer, 1998.

[16] D. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*. Mar. 2015, ISBN: 9781118575536.

[17] L. Najman and M. Schmitt, "Geodesic saliency of watershed contours and hierarchical segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 12, pp. 1163–1173, 1996. DOI: `10.1109/34.546254`.

[18] Y. Liu, X. San Liang, and R. H. Weisberg, "Rectification of the bias in the wavelet power spectrum," *Journal of Atmospheric and Oceanic Technology*, vol. 24, no. 12, pp. 2093–2102, 2007. DOI: `10.1175/2007jtecho511.1`.

[19] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000. DOI: `10.1126/science.290.5500.2319`.