Unsupervised methods to discover events from spatio-temporal data

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Xi Chen

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Prof. Vipin Kumar, Advisor

May, 2017

Acknowledgements

First of all, I would like to thank my advisor, Professor Vipin Kumar, for his unconditional help, advice, and support. He is the person that walked me into the data mining world. I learned tons of skills and knowledge from him, which build the technique foundation for my future career. In addition, he also taught me how to learn a new thing, how to solve a new problem, how to present a work, and how to collaborate with people from different backgrounds. These knowledge and experience helped me finish my PhD study very fruitfully. And even more important, I believe all the things that I learned from him are treasures that will be of great values in my entire career.

I also would like to thank two brilliant researchers, Dr. Shyam Boriah, and Professor James Faghmous. I worked with them in different stages of my PhD study. To me, they are not only very respectful collaborators, but more like my mentors that helped me finally grow up as an independent researcher. Next, I would like to thank the committee members, Professor Shashi Shekhar, Professor Arindam Banerjee, and Professor Snigdhansu Chatterjee. I could not finish my dissertation without their help. Furthermore, deep thanks also go to the undergraduate students that worked with me very closely in many research projects, especially Yuanshun Yao, Sichao Shi, and Robert Warmka. In addition, many thanks go to my labmates in University of Minnesota for the colorful and rewarding time that we spent together. The chatting, discussions, and arguing related to data mining questions that we have in the lab and the Espresso cafe is one of my most cherished memory.

Last but the most important, I want to thank my family, especially my parents, Guisen Chen and Yumin Cai, and my husband Chao Guo. Nothing can be achieved in my life without the support and unconditional love they provide. You were and will always be my heroes.

Dedication

To my father, Chen Guisen, and my mother, Cai Yumin.

Abstract

Unsupervised event detection in spatio-temporal data aims to autonomously identify when and/or where events occurred with little or no human supervision. It is an active field of research with notable applications in social, Earth, and medical sciences. While event detection has enjoyed tremendous success in many domains, it is still a challenging problem due to the vastness of data points, presence of noise and missing values, the heterogeneous nature of spatio-temporal signals, and the large variety of event types.

Unsupervised event detection is a broad and yet open research area. Instead of exploring every aspect in this area, this dissertation focuses on four novel algorithms that covers two types of important events in spatio-temporal data: change-points and moving regions.

The first algorithm in this dissertation is the Persistence-Consistency (PC) framework. It is a general framework that can increase the robustness of change-point detection algorithms to noise and outliers. The major advantage of the PC framework is that it can work with most modeling-based change-point detection algorithms and improve their performance without modifying the selected change-point detection algorithm. We use two real-world applications, forest fire detection using a satellite dataset and activity segmentation from a mobile health dataset, to test the effectiveness of this framework.

The second and third algorithms in this dissertation are proposed to detect a novel type of change point, which is named as contextual change points. While most existing change points more or less indicate that the time series is different from what it was before, a contextual change point typically suggests an event that causes the relationship of several time series changes. Each of these two algorithms introduces one type of contextual change point and also presents an algorithm to detect the corresponding type of change point. We demonstrate the unique capabilities of these approaches with two applications: event detection in stock market data and forest fire detection using remote sensing data.

The final algorithm in this dissertation is a clustering method that discovers a particular type of moving regions (or dynamic spatio-temporal patterns) in noisy, incomplete, and heterogeneous data. This task faces two major challenges: First, the regions (or clusters) are dynamic and may change in size, shape, and statistical properties over time. Second, numerous spatio-temporal data are incomplete, noisy, heterogeneous, and highly variable (over space and time). Our proposed approach fully utilizes the spatial contiguity and temporal similarity in the spatio-temporal data and, hence, can address the above two challenges. We demonstrate the performance of the proposed method on a real-world application of monitoring in-land water bodies on a global scale.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Spatio-temporal data are rapidly becoming ubiquitous thanks to affordable sensors and storage. These information-rich data have the potential to revolutionize diverse fields such as social, Earth, and medical sciences where there is a need to extract and understand complex phenomena and their dynamics. Additionally, data in such scientific domains tend to be unlabelled since collecting data labels is often a time-consuming and labor-intensive operation. This highlights the importance of unsupervised methods in analysing spatio-temporal data.

Unsupervised event detection is an important problem that has enjoyed tremendous interest in the data mining community [3, 5, 18, 21, 45, 52, 54, 69, 71]. Yet, it is still a challenging problem in many applications due to the vastness of data points, presence of noise and missing values, the heterogeneous nature of spatio-temporal signals, and the large variety of event types. Designing event detection methods that can overcome all the above challenges is an ultimate and yet ambitious task. This dissertation is one step in this direction. It includes four algorithms, each of which solves a combination of these challenges under certain assumptions. I hope that introducing these methods to the data mining community will allow us to explore the real-world event detection problem in more depth and design a host of methods to analyze spatio-temporal data more efficiently and accurately.

1.1 Events in spatio-temporal data

Literally, an event is an occurrence of something that is important. Although event detection is considered as one of the major research tasks, there is no formal definition of events in the data mining community. In this dissertation, I consider events as either rarely occurring patterns (*e.g.*, outliers or anomalous) or patterns that include changes (*e.g.*, change-points in a time-series or moving regions).

Spatio-temporal data typically contain three types of attributes: behavior attributes, temporal attributes, and spatial attributes. Theoretically, events can be defined by their behavior attributes only. However, most interesting events in spatio-temporal data are defined in the context of spatial and/or temporal information. I summarize events in spatio-temporal data into three categories: spatial events, temporal events, and spatio-temporal events. Figure 1.1 shows the taxonomy of events in spatio-temporal data. Figure 1.1 also provides several example events in each category (in the blue area). All the examples are commonly studied events and have been applied in real applications to solve critical problems. Among all types of events, this dissertation focuses only on modeling-based time-series change-points (Chapter 2), contextual time-series change-points (Chapter 3 and Chapter 4), and moving regions (Chapter 5). These three types of events are marked by grey blocks in Figure 1.1.

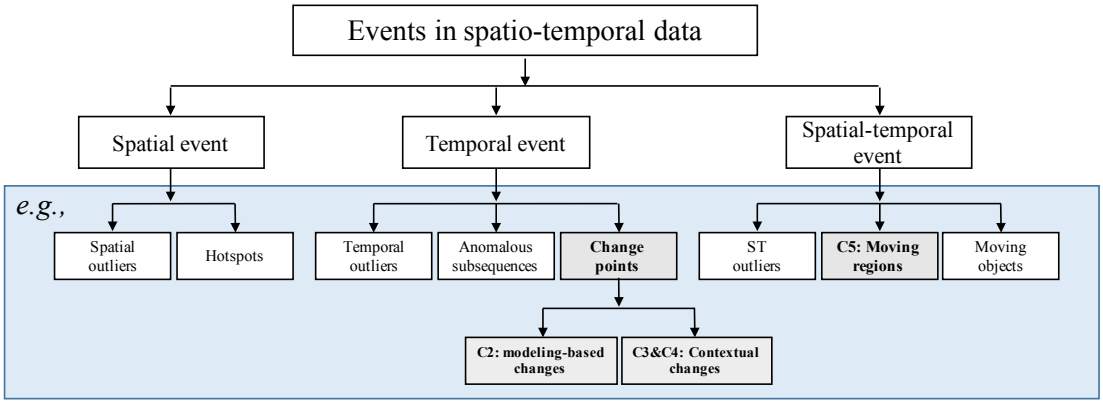


Figure 1.1: The taxonomy of events in spatio-temporal data.

1.1.1 Spatial events

Spatial events are data points with abnormal behavior attributes in the context of spatial attributes. Spatial outliers and hot-spots are two commonly seen spatial events.

Spatial outliers [18] are data points with abnormal behavior attributes compared with their spatial neighbors (*i.e.*, data points that are spatially closed to the target data points). Figure 1.2 shows data points in a dataset. The x-axis and y-axis in Figure 1.2 are the two spatial attributes (*i.e.*, the latitude and longitude) of the data points. The color of any point indicates the value of its behavior attribute. In other words, points with the same color have the same behavior attributes and points with different colors have different behavior attributes. The shapes of points are for illustration purposes and they are not related to data attributes. In Figure 1.2, the two rectangle points (one in red and the other in green) are spatial outliers. The red rectangle point is surrounded by blue points and the green rectangle point is surrounded by red points, which means that the two points have different behavior attributes compared with their spatial neighbors. Thus, the two rectangle points are spatial outliers.

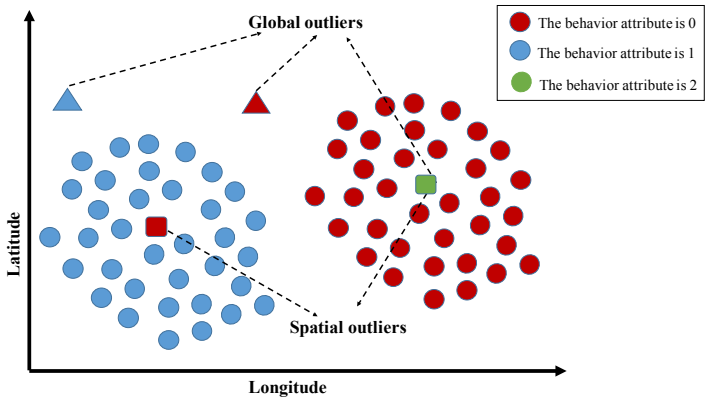


Figure 1.2: The x-axis and y-axis of the plot are the spatial attributes (*i.e.*, the latitude and longitude). The color of a point indicates its behavior attribute. The two rectangle points are spatial outliers because they are surrounded by points with very different behavior attributes. The green rectangle point is also a global outlier with respect to the behavior attribute because it is the only green point in the dataset. The two triangle point are global outliers with respect to the spatial attributes since their spatial attributes are very different from all the other data points.

Global outliers are data points with certain attributes (*e.g.*, spatial attributes or

behavior attributes) that do not conform to the majority data. Figure 1.2 shows two types of global outliers: global outliers with respect to spatial attributes and global outliers with respect to behavior attributes. In detail, the blue triangle point and the red triangle point are global outliers with respect to spatial attributes since they are located very far away from the other points. In other words, their spatial attributes are very different from the majority of the points. The green rectangle point is a global outlier with respect to behavior attributes because its behavior attribute is different from all the other points (*i.e.*, the green rectangle point is the only data point with green color). In contrast, in addition to the red rectangle point, there are many other red points in the plot. Hence, the red rectangle point is not a global outlier with respect to behavior attributes.

While global outliers indicate global extreme events (*e.g.*, extremely low rainfall in the whole earth), spatial outliers are sensitive to locations (*e.g.*, extremely low rainfall in Minnesota), and hence are critical in detecting regional events, such as local extreme meteorological events (*e.g.*, tornadoes and hurricanes) and abnormal highway traffic patterns [20].

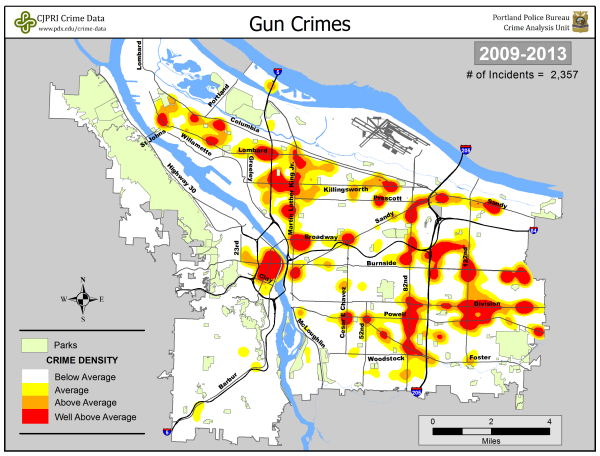


Figure 1.3: Hotspots (red regions) in the gun crime incidents data in Portland between 2009 to 2013. This example is from [2].

Hot-spots [11, 32, 68] are areas where the density of data (with a certain behavior attributes) is significantly higher than the other areas. Figure 1.3 shows the density map of gun crime incidents in Portland between 2009 and 2013 [2]. We can observe that

hotspots were dispersed throughout downtown and Northwest, Northeast, Southeast, and East Portland. Hot-spots have been widely used in analyzing social activities (*e.g.*, crimes [15, 16, 73]) and biodiversity (*e.g.*, the distribution of a certain species) [43, 74].

1.1.2 Temporal events

Temporal events are defined by behavior attributes and temporal attributes. Temporal outliers, anomalous sub-sequences, and change-points are three commonly studied temporal events.

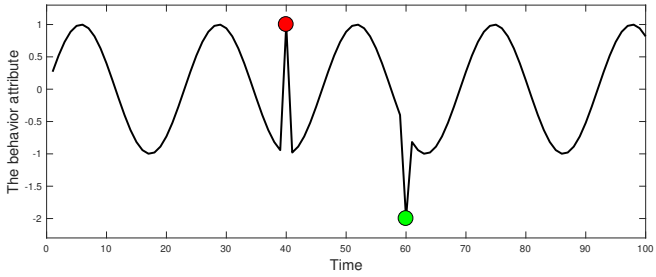


Figure 1.4: Two temporal outliers (the red point and the green point) in a time-series data. The red point is not a global outlier, while the green point is a global outlier.

Temporal outliers are time-points that do not follow the general temporal trend of a time-series. Similar to spatial outliers, temporal outliers may not be global outliers. Figure 1.4 shows two temporal outliers (a red point and a green point) in a time-series data. The x-axis in the plot is the temporal attribute and the y-axis is the behavior attribute. We can observe that both the green point and the red point do not match the general time-series trend. Thus, they are two temporal outliers. Additionally, the behavior attribute of the green point is different from all the other time points. Hence, the green point is also a global outlier (with respect to the behavior attributes). The red point, in contrast, is not a global outlier since its behavior attribute is similar to the behavior attributes of many other time points. Temporal outliers are important in detecting events that are sensitive to time. For example, 0 ^{o}F is normal and not very harmful during winter in Minnesota. But 0 ^{o}F in late spring or early summer in the same location can significantly reduce the yield of crops. Similar applications can be found in many other domains such as traffic control [57] and fraud detection [18].

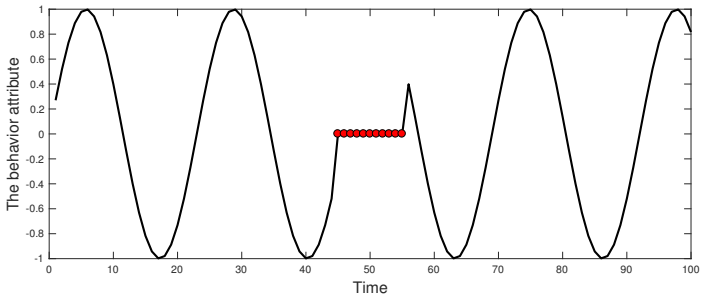
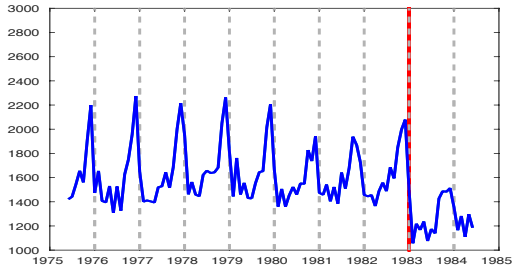


Figure 1.5: An example of anomalous sub-sequence (red points).

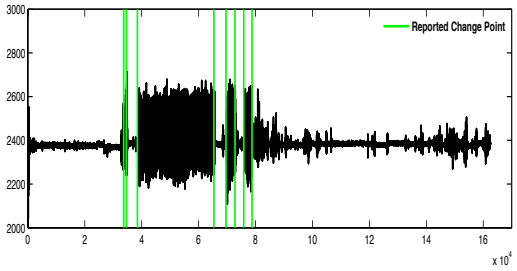
An anomalous sub-sequence, which is also known as a collective anomaly [18, 19], is several consecutive time-points that form an abnormal temporal pattern. Figure 1.5 shows an anomalous sub-sequence (marked as red points) in a periodic time-series. Anomalous sub-sequences have been applied to clinic data as key patterns of certain diseases. For example, an anomalous sub-sequence in electrocardiography data may indicate a heart problem [62]. In addition, anomalous sub-sequences are also used in detecting Web-based attacks [18] and monitoring air transportation systems [12].

Change points are the times when the behavior of one or more time series changes. Many types of change points have been defined in literature. Modeling-based change points, or structural change points, is one of the most commonly used types of change points [5, 46, 47, 71, 72, 78, 83, 86, 88]. Roughly speaking, a modeling-based time-series change point is the time when a time-series starts to significantly deviate from its own historical data.

Figure 1.6 (a) and (b) show two examples of modeling-based change points. The time series in Figure 1.6 (a) represents the monthly deaths and serious injuries on UK roads between 1975 and 1985. This time series contains one change point, which is marked by a red vertical dot line. This change point matches the time when the seat-belt law was introduced. Figure 1.6 (b) shows the chest-mounted accelerometer data (the y-acceleration field) that was recorded from a user when he was asked to do different activities. Vertical green lines in the plot indicate change points in the time series. All the change points are the times when the user changed his activity.



(a) Monthly deaths and serious injuries in UK roads between 1975 to 1985. The change point (the red vertical dot line) matches the time when the seat-belt law was introduced.



(b) The y-acceleration field in a chest-mounted accelerometer time-series data. The change points (green vertical lines) are the times when the user changed his activity.

Figure 1.6: Examples of modeling-based time-series change points.

1.1.3 Spatio-temporal events

Spatio-temporal events are defined in the context of both spatial and temporal information. Spatial outliers, moving objects, and moving regions are three examples of spatio-temporal events.

Similar to the definitions of spatial outliers and temporal outliers, spatio-temporal outliers are data points with abnormal behavior attributes compared with their spatio-temporal neighbors [26, 53]. Identifying spatio-temporal outliers can lead to discovery of many interesting events such as local instability or deformation. For example, Jun et al. [50] detects faulty sensors as spatio-temporal outliers in the sensor network.

Most studies on moving objects aim to discover the trajectory patterns of objects whose spatial attributes (*e.g., GPS signals*) change with time [51, 58–60, 77]. Examples of studied objects include animals, human beings, and cars. Moving object clusters [77] is one major type of event in this research area. Typically, moving object clusters are groups of objects that travel together for an extended period of time [51, 58, 59]. Discovery of such clusters is critical in understanding animal behaviors, detecting climate events (*e.g.*, hurricane tracks), and helping in vehicle controls.

Typically, moving regions are defined in raster spatio-temporal data (*e.g.*, earth-orbiting satellites, fMRI recordings, and surveillance videos). A raster spatio-temporal dataset is a three dimensional gridded data cube as shown in Figure 1.7. The three dimensions consist of two dimensional spatial attributes (*e.g.*, latitude and longitude) and

a one dimensional temporal attribute (*i.e.*, date). Each data cell in the data cube contains one or multiple behavior attributes. For example, in surveillance videos, each data cell is a pixel in a certain time frame and it usually contains three behavior attributes (*i.e.*, the R, G, and B values).

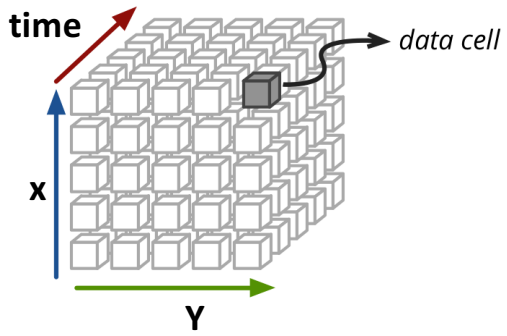


Figure 1.7: A raster spatio-temporal dataset typically is a three dimensional data cube.

At any time-step, a region in a raster spatio-temporal dataset includes multiple spatially connected data cells that share similar behavior attributes. There is a moving region in the dataset when the position of the same region shifts or moves over time. Figure 1.8 shows an example of moving regions. In this example, red pixels in each time framework have similar behavior attributes in the SSH satellite dataset and hence each red area is a region. The regions in different time frames are located in different positions, but they indicate the same physical phenomenon (*i.e.*, an ocean eddy [38]). Therefore, the red regions form a moving region.

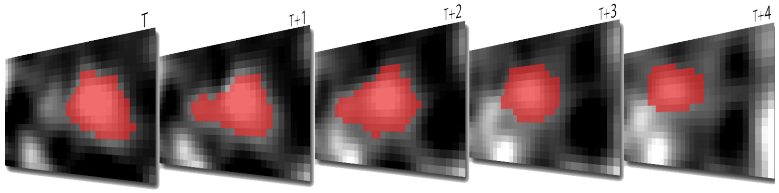


Figure 1.8: An example of a moving region. The moving region (marked by red color pixels) is a moving eddy at five successive time-steps of SSH satellite data.

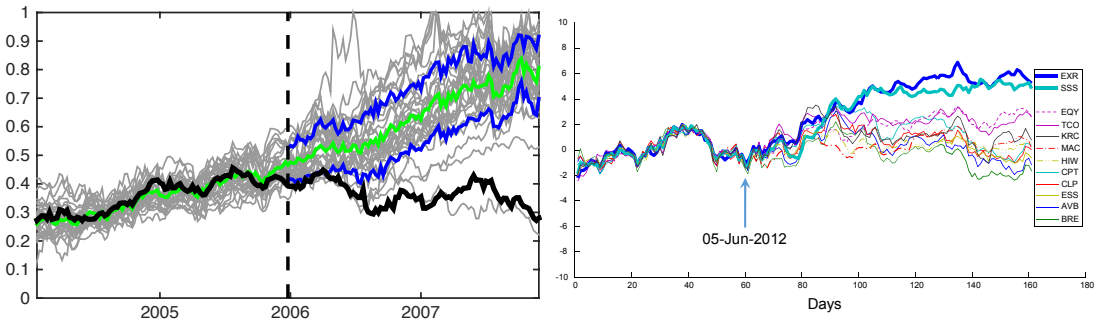
1.2 Summary of contributions

Discovery of events in spatio-temporal data is a wide open area. Instead of exploring every aspect in this domain, this dissertation includes four novel algorithms that cover two major types of events in spatio-temporal data: change-points and moving regions.

The first proposed algorithm is a Persistence Consistency (PC) framework. The PC framework is a solution to detect modeling-based change points in noisy time-series data. While modeling-based change-point detection methods have achieved tremendous success in many applications, they perform poorly when data are noisy and have outliers. The PC framework enhances the performance of existing modeling-based methods when these data challenges exist. There are three contributions in this work. First, this work summarizes the common structure of most modeling-based change-point detection methods. Based on this common structure, it also introduces a four step approach to design a modeling-based approach for a given application. Second, this work explains and demonstrates the negative impact of noise and outliers on a modeling-based approach. Finally, this work presents the PC framework. This PC framework can be combined with most modeling-based approaches to produce more accurate detection results when data are noisy and contain outliers.

The second and third algorithms in this dissertation are designed to detect contextual change points from time series. Contextual change points are a novel type of change point. While a modeling-based change point more or less indicates that the time series is different from what it was before, a contextual change point typically suggests an event that causes the relationship of several time series to change. This dissertation introduces two types of contextual change points: Singleton Contextual Time-series Change points (S-CTCs) and Group level Contextual Time-series Change points (G-CTCs). Roughly speaking, a S-CTC is the time when one time series starts to deviate from its "peer group" and a G-CTC is the time when a new peer group forms or an old peer group disbands. Here, we define the peer group as a group of highly correlated time series.

Figure 1.9 shows one example of S-CTC and one example of G-CTC. In Figure 1.9 (a), the stock price of Sprint Corporation (the black curve) behaved similarly to stock prices of several other companies (the grey curves) from year 2004 to year 2006. Hence, the grey time series formed a peer group of the black time series. The black time series



(a) An example of a S-CTC.

(b) An example of a G-CTC.

Figure 1.9: Examples of contextual time-series change points.

began to deviate from its peer group in the beginning of year 2006. Therefore, there is a S-CTC occurred in Jan. 2006. We use blue lines for the 16 & 84 percentiles of all the grey time series after the change point, and a green line for the mean of the grey time series. In Figure 1.9 (b), several REITs stock price data had highly correlated behaviors before Jun. 5th, 2012, and hence they formed a peer group. The two self-service storage companies (one in dark blue and the other in light blue) began to fork out from the general REIT context, which led the whole group to split. Thus, Jun. 5th, 2012 is a G-CTC in the REITS stock price data.

As shown in Figure 1.9, contextual change points are useful patterns to discover events from non-stationary data, where the modeling-based change points are not suitable. There are two major contributions in the work of detecting contextual changes. First, these two works introduce contextual time-series change points to the data mining community. Second, two algorithms are proposed. Each of them detects one particular type of contextual change point.

The last algorithm in this dissertation is to discover moving regions from a spatio-temporal dataset. This task faces two major challenges. First, the regions are dynamic and may change in size, shape, and statistical properties over time. Second, numerous spatio-temporal data are incomplete, noisy, heterogeneous, and highly variable (over space and time). The proposed algorithm is a new spatio-temporal data mining paradigm and can autonomously identify dynamic spatio-temporal clusters in the presence of the above data issues. The major contribution of this work is the proposed

paradigm. Compared with most existing methods, which either analyze data in space and then aggregate/associate over time, or analyze data over time and then smooth over space [30, 75], the proposed paradigm takes advantage of both spatial and temporal auto-correlation. In addition, it utilizes knowledge from physical domains. Hence, it can discover moving regions in noisy, incomplete, and heterogeneous spatio-temporal data.

1.3 Thesis Overview

The remainder of this dissertation is organized in the following way. Chapter 2 presents the PC framework. The PC framework is a general framework that can increase the robustness of most modeling-based change-point detection methods to noise and outliers. Chapter 3 and Chapter 4 describe the two types of contextual change points and present the corresponding detection algorithms. Chapter 5 introduces the clustering paradigm that discovers a type of moving region in noisy, incomplete, and heterogeneous spatio-temporal data. Finally, Chapter 6 summarizes the whole dissertation and discusses potential research directions and applications of event detection in spatio-temporal data.

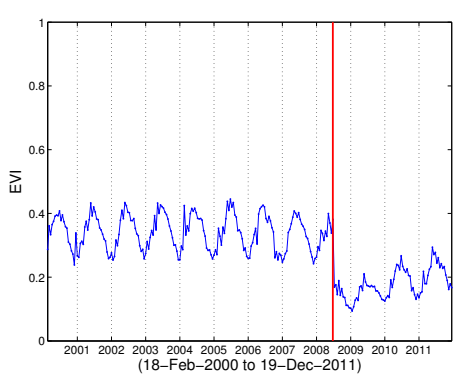
Chapter 2

The Persistence Consistency (PC) framework:

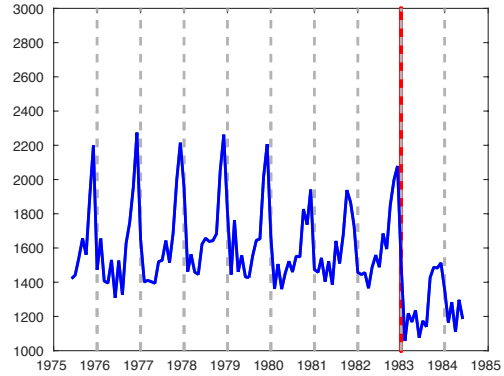
A solution to detect change-points in noisy time-series data using modeling-based methods

2.1 Motivation

Time-series change-point detection aims to autonomously identify time-steps where the behavior of a time-series significantly deviates from a predefined model. It is an active field of research with notable applications in finance [4], health [42], advertising [10], network security [41], and a host of other domains where data is presented as time-series. Figure 2.1 shows two examples of change-points (the red vertical lines) in a remote sensing time-series (left) and an automotive accident time-series (right). As the number of time-series continues to grow, there is an increasing need for autonomous change point detection and reporting methods. One common class of change-point detection algorithms relies on time-series models to define change-points. These modeling-based methods tend to be ad hoc, where each method is specifically tailored to the target application and deep domain expertise is required. While these methods are specialized, they share the common characteristic that they perform poorly when the data are noisy



(a) The "greenness" index of a forest area in Northern California. A forest fire occurred in this location in 2008.



(b) Monthly deaths and serious injuries on UK roads between 1975 to 1985. A seat-belt law was introduced in 1983.

Figure 2.1: Examples of change points (red vertical lines) that can be detected by modeling-based methods.

or have outliers. We propose a general framework to make existing modeling-based change-point detection algorithms robust to noise and outliers, regardless of the underlying change point detection algorithms used by the researcher. This general framework immediately improves existing methods without the need to change the existing algorithm.

2.2 Background

As the name suggests, modeling-based change-point detection methods detect change-points by modeling the given time series. In detail, these methods assume data in a time-series are generated from a time-series model. We call this model *the underlying model* of the given time-series. Typically, the underlying model is a stochastic function. A time-series, in most cases, follows one single underlying model. However, sometimes the underlying model of a time-series may alter from one function to the other. The time when the underlying model changes is named *change-points*. Modeling-based change-point detection methods are concerned with detecting these change-points. Examples of modeling-based change-point detection methods include BFAST [88], the cumulative sum (CUSUM) chart [71, 72], and many other algorithms [5, 46, 47, 78, 83, 86].

2.2.1 The common structure of most modeling-based change-point detection methods

While the existing methods are numerous, most modeling-based change-point detection algorithms share a similar algorithm structure as shown in Algorithm 2.1[1] . In detail, these modeling-based methods can be defined by a time-series model function and a change-score function. Their final outputs are usually a change-score time-series.

Algorithm 2.1: *calAnomalyTS*

Input: $\boldsymbol{x}$: a time-series data;
$f(\cdot)$: a time-series model;
$g(\cdot)$: a change-score function;

Output: $\boldsymbol{a}$, a change-score time series;

1 **for** *any* $x_t \in \boldsymbol{x}$ **do**
2 $\quad \boldsymbol{x'} = \boldsymbol{x}(t-w, t-1))$;;
3 $\quad \hat{\boldsymbol{\theta}} = est(\boldsymbol{\theta}|\boldsymbol{x'}, f)$;;
4 $\quad a_t = g(\hat{\boldsymbol{\theta}}, x_t)$;;
5 **end**

The time-series model function is a mathematical function with several unknown parameters. Most algorithms assume the underlying models of all time-series data are the same function with different parameters. With the input time-series model function, a modeling-based detection method learns an underlying model for each time-series independently (Line 3). The change-score function is to calculate a change-score for each time-point in the time-series. The change-score of a time-step measures how much the underlying model changes around that time-step. Most change-point detection methods keep all change-scores of a given time-series also as a time-series (Line 4). We name it the *change-score time-series*.

After obtaining a change-score time-series for each input time-series, an algorithm can report the final change-points. There are two commonly used methods. First, we can label all time-steps as change-points if their change-scores exceed a user-defined

[1] Algorithm 2.1 is a pseudo-code that calculates the change-score time-series for a given time-series. This algorithm trains the time-series model using a sliding-window approach and hence is good when detecting multiple change-points from a single time-series. This pseudo-code has been commonly used in many modeling-based change-point detection methods [5, 47, 88].

threshold. Second, we can report k change-points totally from the dataset. The change-points are the k time-steps with the highest change-scores in a certain time duration.

Hundreds of modeling-based change-point detection methods exist in the literature mainly due to the special tailored time-series model and change-score function. In most cases, these two components are designed in an application-specific fashion to achieve the best performance. In other words, most modeling-based change-point detection methods have their own time-series model and change-score function and are only suitable for one or several applications.

2.2.2 The four steps in designing a modeling-based change-point detection method

The common structure of existing modeling-based change-point detection methods also inspires insights on how to design a new method for a particular detection task. Designing a new modeling-based method usually involves four steps: (i) choose an appropriate time-series model; (ii) design a change-score function that quantifies how "different" two time-series segments are (before and after a change-point)[2] ; (iii) compute the change-scores for each time-point based on the chosen time-series model and anomaly-score function; and (iv) select a mechanism to report change-points. Next, we illustrate the four steps using two simple examples.

Case I: Detecting abrupt changes for periodic time-series

In case I, we aim to detect abrupt change-points in periodic time-series data. The top panel in Figure 2.2 shows one time-series (the black curve) and its change-point (the red line). As shown in Figure 2.2, data after the change-point have much higher values than data before the change-point. The given change occurs suddenly and dramatically. We name this type of change *abrupt changes*. Here, we design a simple abrupt change-point detection method using the above mentioned four steps.

First, we choose an appropriate time-series model. Given the regular periodicity in the data, we choose the most simple seasonal ARIMA model [47]:

$$x_t = \alpha_t x_{t-1} + \beta_t x_{t-p} + e$$

[2] Sometimes, the change-score may integrate anomaly-scores from multiple time-steps to enhance change events from random noise and/or outliers.

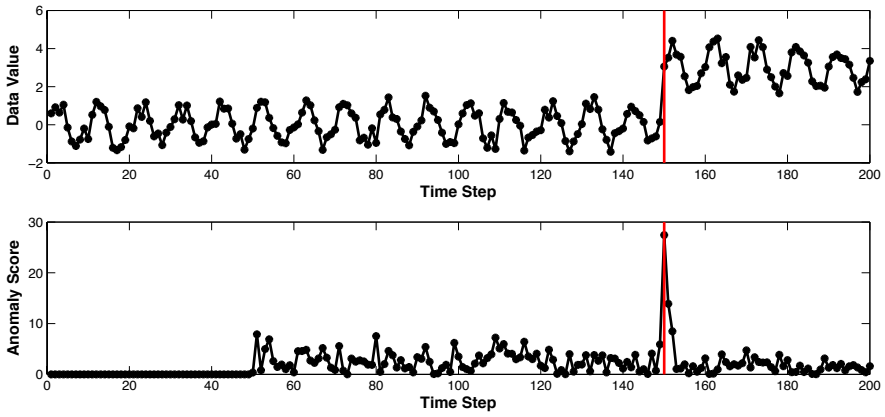


Figure 2.2: A periodic time series (on the top panel) that changes around the red line and its corresponding anomaly scores.

where, x_t is the observation of the given time-series $\mathbf{x}$ at time-step t; p is the periodicity of $\mathbf{x}$, which is 10 in our example; e is a random noise term; and α_t and β_t are the two unknown parameters.

Second, we choose an change-score function. Since the time-points after the change-point are dramatically greater than the time-points before the change-point (as shown in the top panel of Figure 2.2), we measure the degree of a change by the difference between the observed and the predicted values, which is:

$$a_t = |\hat{x}_t - x_t| = |\hat{\alpha}_t x_{t-1} + \hat{\beta}_t x_{t-p} - x_t|$$

where a_t is the change-score of x_t; $\hat{x}_t$ is the predicted observation; and $\hat{\alpha}_t$ and $\hat{\beta}_t$ are the two estimated parameters.

Next, we compute the change-score for each time-step. To obtain change-scores, we need to estimate $\hat{\alpha}_t$ and $\hat{\beta}_t$ for each time-step x_t. Here, we use a sliding window method as shown in Algorithm 2.1 and set w to 50. The bottom panel of Figure 2.2 is the change-score time-series of the given example.

Finally, we report the change-points. Here, we use a threshold (*e.g.*, 15) to report the event. For a more complex problem, we can train the threshold using a training set.

Case II: Detecting changes in variance for a stationary time-series

In case II, we need to detect changes in variance for a stationary time-series. Figure 2.3 shows one time-series (the black curve) and its two change-points (red vertical lines). We design a modeling-based change-point detection method also using the four steps.

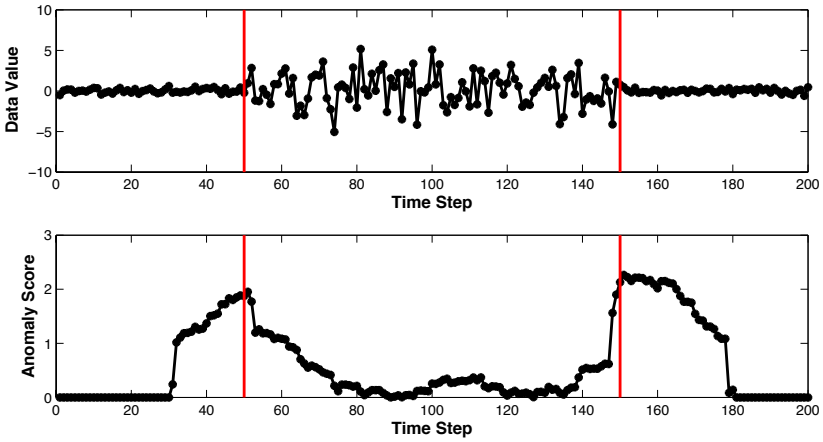


Figure 2.3: A stationary time series (top panel) that experiences a change in variance around the red line and its corresponding anomaly scores (bottom panel).

First, as shown in the top panel in Figure 2.3, the time-series is stationary (*i.e.*, its expectation and variance are almost constants) when no change occurs. Hence, we select the time-series model as:

$$x_t = \alpha + e_t$$

where, α is a constant value; and e_t is the noise term that follows a Normal distribution $N(0, \sigma_t^2)$. In this model, α and σ_t are the two unknown parameters.

Second, the top panel in Figure 2.3 also shows that only the variance of the time-series changes around the change-points. The expectation remains same. Therefore, we decide to score this event using the differences in the variance.

$$a_t = |log(\frac{\sigma_{tb}}{\sigma_{ta}})| = |log(\sigma_{tb}) - log(\sigma_{ta})|$$

where, σ_{tb} is the time-series variance before time-step x_t and σ_{ta} is the variance after x_t.

Next, we compute the change-score for each time-step. We use 30 values to estimate a variance value. In other words, σ_{tb} is the variance of $\{x_{t-30}, \cdots, x_{t-1}\}$ and σ_{ta} is the variance of $\{x_{t+1}, \cdots, x_{t+30}\}$. The change-score time-series of the given example is shown in the bottom panel of Figure 2.3.

Finally, we report the change-points. For the given example, we can report time-points with the highest two change-scores as the final change-points.

2.2.3 Challenges

The majority of work in this area has focused on applied problems. General solutions have been less studied. Thus, there are two major opportunities within this line of work. First, there is an opportunity to create autonomous (application-agnostic) change-point detection algorithms that take any type of time-series and return change-points. Second, there is an opportunity to develop general solutions to overcome common data quality problems such as noise and outliers. The first opportunity is our grand vision but will require significant innovations. Instead, we focus on the second opportunity of creating a framework to allow existing model-based change-point detection algorithms to overcome outliers and noise. We next highlight how these challenges affect existing methods.

Outliers

Outliers are rare and anomalous observations that are different from the overall population. They are not change-points. Typically, the underlying time-series model does not change from one function to the other when an outlier presents. Figure 2.4 shows a time-series with two outliers (the red points).

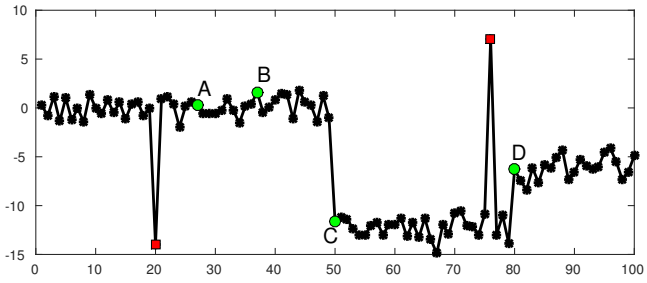


Figure 2.4: The impact of outliers on modeling-based change-point detection methods.

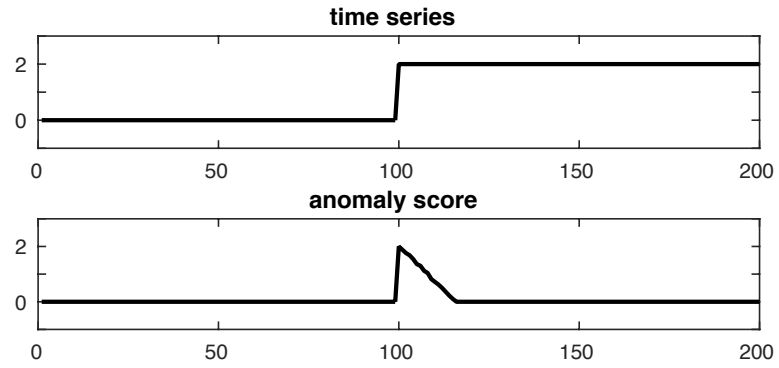


Figure 2.5: The decay phenomenon in a change-score time series.

Outliers negatively affect modeling-based change-point detection algorithms in two aspects: (i) they can be detected as change-points due to their anomalous behaviors; and (ii) they bias parameter estimation of the time-series model, especially when estimating the parameters using least square methods. Figure 2.4 demonstrates these two impacts of outliers. The figure shows a time-series that contains both change-points and outliers. The change score of a time step is defined as the absolute difference between its observation and the mean of its previous 20 time-steps. The four points highlighted with green color dots have change-scores 0.32(A), 1.95(B), 2.88(C) and 0.58(D). If we delete outliers manually and re-compute the change-scores, they are 0.15(A), 1.95(B), 2.88(C) and 2.54(D). The scores of A and D when outliers are removed are significantly different from the ones with outliers. This shows that outliers impact change-point detection, which in turn (i) causes non-change points to have artificially high change-scores; and (ii) decreases the change-scores of real change-points.

Noise

Noise is a random signal provided by a meaningless or irrelevant resource(s). Typically, noise is mixed with "real" signals and causes the observed data to randomly deviate from their true value. It can be challenging to detect change-points from a noisy time-series. Common solutions include (i) preprocessing the data; and (ii) aggregating change-scores of several consecutive time-steps to assess the significance of a change event. However, both of these solutions have limited applicability. First, preprocessing methods (*e.g.*,

smoothing), have aggregation issues and may introduce extra false negatives. Second, change-score aggregation methods tend to use the area-under-the-curve as a measure of the significance of a change event. This can be misleading since change-scores tend to be high and then decay over time as shown in Figure 2.5. This fact makes the area-under-the-curve an unreliable metric.

The top panel of Figure 2.6 shows a noisy time-series that changes around the green line. Its anomaly scores are shown in the bottom panel. The change-scores are calculated in the same way as the example shown in Figure 2.4. Note that the area under the curve of the real change-point (the green line) is smaller than the one caused by random noise (the red line).

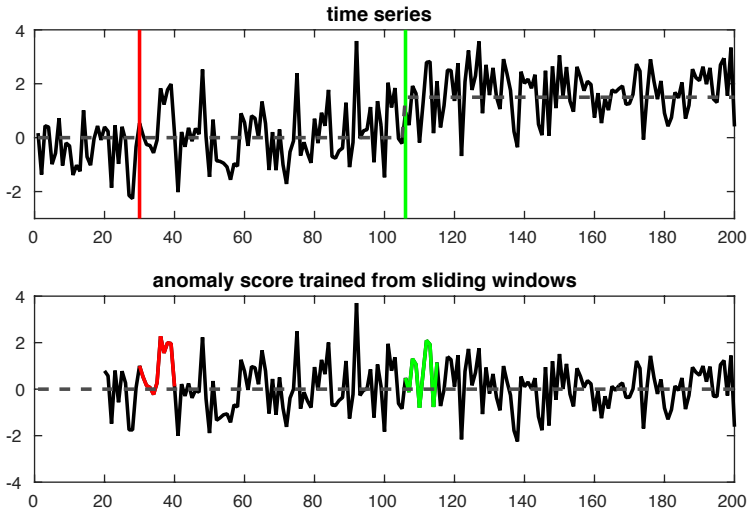


Figure 2.6: The impact of noise on modeling-based change-point detection methods.

2.3 Proposed method

To improve the robustness of most existing modeling-based change-point detection methods to noise and outliers, we propose to: (i) utilize the central method to estimate time-series models and calculate anomaly-scores and (ii) incorporate both the persistence and consistency properties to score the change events. Although the two proposed technologies can be used independently, we construct a framework that can use them together to achieve the best performance. In the following section, we first

introduce the central method and the concepts of persistence and consistency. Then, we present the proposed framework, which has been named the Persistence-Consistency (PC) framework.

2.3.1 The central method for anomaly-score estimation

As we discussed in Section 2.2, outliers, if they exist in the training data, can dramatically reduce the accuracy of a time-series model estimation method and therefore negatively impact the final performance of the change-point detection algorithm. To address this challenge, we propose a central method to calculate anomaly-scores.

The central method is a random sampling technique. We show its pseudo-code as Algorithm 2.2. For comparison purposes, we also show Algorithm 2.1 from Section 2.2 on the right side. For the sake of simplicity, we use TS for time-series in all of the pseudo-codes.

Algorithm 2.2: *calAnomalyCentral*

Input: $f(\cdot)$, a TS model
$g(\cdot)$, an anomaly score function
$\boldsymbol{x}$, a TS data
Output: $\boldsymbol{a}$, An anomaly score TS

1 **for** *any* $x_t \in \boldsymbol{x}$ **do**
2 **for** r *from* 1 *to* β **do**
3 $\boldsymbol{x'} = sample(\boldsymbol{x}(t-w, t-1))$;
4 $\hat{\boldsymbol{\theta}} = est(\boldsymbol{\theta}|\boldsymbol{x'}, f)$;
5 $temp_r = g(\hat{\boldsymbol{\theta}}, x_i)$;
6 **end**
7 $a_t = med(temp_1, \cdots, temp_\beta)$;
8 **end**

Algorithm 2.1: $calAnomalyTS(\cdot)$

Input: $f(\cdot)$, a TS model
$g(\cdot)$, an anomaly score function
$\boldsymbol{x}$, a TS data
Output: $\boldsymbol{a}$, An anomaly score TS

1 **for** *any* $x_t \in \boldsymbol{x}$ **do**
2 $\boldsymbol{x'} = \boldsymbol{x}(t-w, t-1))$;
3 $\hat{\boldsymbol{\theta}} = est(\boldsymbol{\theta}|\boldsymbol{x'}, f)$;
4 $a_t = g(\hat{\boldsymbol{\theta}}, x_t)$;
5 **end**

As introduced in Section 2.2, Algorithm 2.1 is one of the most commonly used methods to generate an anomaly-score for each time-step. As shown in Line 2-3, Algorithm 2.1 uses all available data in the training window to estimate the time series model. When outliers occur in the training data, the estimated time-series model may dramatically differ from the true model. The proposed central method, on the other hand, does

not use all training data. Instead, it estimates one time-series model using a subset of the training data (Line 3 - 4 in Algorithm 2.2) and then calculates one anomaly score from the obtained model (Line 5 in Algorithm 2.2). We repeat this procedure multiple times (Line 2 - 6 in Algorithm 2.2). The final anomaly score is the median of all the obtained scores (Line 7 in Algorithm 2.2). The intuition behind the central method is to reduce the chance of including outliers in the training data.

The following lemma demonstrates the conditions under which anomaly scores calculated by the central method are not impacted by outliers at all. In many cases, even when the condition may not be fully satisfied, the central method can still largely increase the robustness of a change point detection method to outliers.

Lemma 2.1. *Assume that the sampling rate is b, the outlier rate is o, and the times of the randomly sampling we do is n, then the probability that no outliers were used to estimate the parameters $\boldsymbol{\theta}$ has the lower bound as below.*

$$p(\hat{\theta} = \Theta | b, o) > (\frac{1-b-o}{1-o})^{on}$$

where Θ is the set of $\boldsymbol{\theta}$ where no outliers are used in the estimation.

Proof. Let $k = on$ and $m = bn$. The probability that no outliers were used to estimate θ is

$$p(\hat{\theta} = \Theta | b, o) = \frac{\frac{(n-k)!}{m!(n-m-k)!}}{\frac{n!}{m!(n-m)!}} = \frac{(n-k)!(n-m)!}{(n-m-k)!n!}$$

When outliers are rare, we have $k < m < n$. Then,

$$\begin{aligned} & p(\theta = \Theta | n, m, k) \\ =& \frac{(n-k)!(n-m)!}{(n-m-k)!n!} \\ =& \frac{(n-m)(n-m-1)\cdots(n-m-k+1)}{n(n-1)\cdots(n-k+1)} \\ =& \prod_{i=0}^{k-1} \frac{n-m-i}{n-i} \end{aligned}$$

Also since $0 < k < m$ and $0 \leq i \leq k-1$, we have

$$\begin{aligned} \frac{n-m-i}{n-i} > \frac{n-m-k}{n-k} &= \frac{n-bn-on}{n-on} \\ &= \frac{1-b-o}{1-o} \end{aligned}$$

Hence,

$$p(\theta = \Theta | n, m, k) > (\frac{1-b-o}{1-o})^{on}$$

□

2.3.2 Persistence and Consistency

Persistence and consistency are two properties of change-points that we can use in a detection method to improve its detection accuracy when noise and outliers present in a dataset. More precisely, we can use the persistence property to distinguish change-points from outliers and the consistency property to enhance the performance of a detection method against noise.

Persistence is a characteristic that we use to avoid labeling outliers as change-points. We define an anomalous time-point (x_t) to be *persistent* if many time-steps after x_t persistently differ from time-steps before x_t. As discussed in Section 2.2, change-points and outliers are two types of anomalous observations in a time series. The (underlying) time-series model is expected to change around a change-point but not around an outlier. Hence, we consider change-points as persistent anomalies since typically multiple time-steps after a change-point are significantly different from time-steps before the change point. In contrast, we define outliers as non-persistent anomalies since time-steps after an outlier typically follow the same time-series model as time-steps before the outlier.

Consistency is used to reduce false alerts due to random noise. We say an anomaly is *consistent* when the anomaly can be detected using multiple subsets of historical data. Most predictive modeling-based change-point detection methods use consecutive historical data to train the model (as shown in Line 2 in Algorithm 2.1). When data is very noisy, normal data may be assigned high anomaly scores occasionally. When change-points are very rare compared with normal points, a small fraction of false alerts can lead to a pool accuracy in the detection results. We use consistency to solve this problem. In detail, we assign multiple anomaly-scores to each time-step using different training data. We say an anomaly is consistent if all or most of its anomaly scores are high. Otherwise, the anomaly is inconsistent. Both change points and outliers are considered to be consistent. A normal data point, on the other hand, may have several high anomaly scores. But the probability that anomaly scores of a normal time-step are consistently high across different training sets is low.

2.3.3 The Persistence Consistency (PC) framework

The proposed Persistence-Consistency (PC) framework takes advantage of both the persistence and consistency properties. In addition, it is also very convenient to add the central method into the framework. Hence, the PC framework can dramatically increase the robustness of a given change point detection method to both noise and outliers. The PC framework can be applied to most modeling-based detection methods.

Algorithm 2.3 shows the pseudo-code of the PC framework. This proposed framework has three types of inputs: (i) a time series data, which is denoted by a vector $\mathbf{x}$; (ii) a change point detection method, which is characterized by a time-series model $f(\cdot)$ and an anomaly-score function $g(\cdot)$; and (iii) a set of user-defined parameters Θ. Here, we assume that the selected time-series change-point detection method (*i.e.*, $f(\cdot)$ and $g(\cdot)$) fits the application very well (*i.e.*, the detection accuracy is very high) if no noise and outliers exist in the time-series data. The PC framework outputs a change-score for each time-step. We keep all the change-scores in a time-series (which is denoted by $\mathbf{y}$) and name the time-series as the change-score time-series.

Algorithm 2.3: The pseudo-code of the PC framework.

Input: $\boldsymbol{x}$: a TS data;
$f(\cdot)$: a TS model;
$g(\cdot)$: an anomaly-score method;
Θ: user defined parameters;

Output: $\boldsymbol{y}$: a change score time series.

1 $\boldsymbol{M} = calAnomalyMatrix(\mathbf{x}, f(\cdot), g(\cdot), \Theta)$;
2 $\boldsymbol{y} = calChangeScoreTS(\mathbf{M}, \Theta)$;

There are two steps in the PC framework. The first step (Line 1 in Algorithm 2.3) is to calculate an anomaly-score matrix $\mathbf{M}$ for the given time-series data $\boldsymbol{x}$. We name the first step *anomaly-score matrix construction* or function $calAnomalyMatrix(\cdot)$. The second step (Line 2 in Algorithm 2.3) is to generate the final change-score time-series from the obtained anomaly-score matrix. We call the second step *change-score time-series generation* or function $calChangeScoreTS(\cdot)$.

Anomaly-score matrix construction: $calAnomalyMatrix(\cdot)$

Anomaly-score matrix is an internal data structure that we use to keep all the anomaly scores of the input time-series data. This novel data structure is the core component in the proposed PC framework. Algorithm 2.4 illustrates the construction process of the anomaly-score matrix for a given time-series $\boldsymbol{x}$. Note that in Algorithm 2.4, we DO NOT include the central method for anomaly-score calculation. The most comprehensive algorithm (*i.e.,* the PC framework with the central method) is provided later in Algorithm 2.5. Again, we show Algorithm 2.1 on the right side of Algorithm 2.4 to highlight the differences between the construction of an anomaly-score matrix and an anomaly-score time-series.

Algorithm 2.4: $calAnomalyMatrix$

Input: $\boldsymbol{x}$: a TS data;
$f(\cdot)$: a TS model;
$g(\cdot)$: an anomaly score function;
Θ: user defined parameters;
Output: $\boldsymbol{A}$: an anomaly score Matrix;

1 **for** *any* $x_t \in \boldsymbol{x}$ **do**
2 $\hat{\boldsymbol{\theta}} = est(\boldsymbol{\theta}|\boldsymbol{x}(t-w, t-1), f)$;
3 **for** *i from t to the end* **do**
4 $A_{t,i} = g(\hat{\boldsymbol{\theta}}, x_i)$;
5 **end**
6 **end**

Algorithm 2.1: $calAnomalyTS$

Input: $\boldsymbol{x}$: a TS data;
$f(\cdot)$: a TS model;
$g(\cdot)$: an anomaly score function;
Θ: user defined parameters;
Output: $\boldsymbol{a}$: an anomaly score TS;

1 **for** *any* $x_t \in \boldsymbol{x}$ **do**
2 $\hat{\boldsymbol{\theta}} = est(\boldsymbol{\theta}|\boldsymbol{x}(t-w, t-1), f)$;
3 $a_t = g(\hat{\boldsymbol{\theta}}, x_t)$;
4 **end**

The PC framework calculates anomaly-scores for multiple time-steps from a single estimated time-series model. In detail, for any qualified time step x_t, the PC framework trains a time-series model using w data before it (*i.e.*, time steps between x_{t-w} and x_{t-1}) (Line 2 in Algorithm 2.4). This step is exactly the same as the selected change point detection method (Line 2 in Algorithm 2.1). Then, besides using the obtained model to estimate an anomaly score for time-step x_t, the PC framework also estimates anomaly scores for all time-steps after x_t and keep all the obtained values in the corresponding positions in the t^{th} row of the anomaly-score matrix (Line 3 - 5 in Algorithm 2.4).

We repeat this procedure through a sliding window approach for all other qualified time-steps.

Figure 2.7 illustrates this process using an image to further help in understanding the construction process. The target time series is on the top of the image and its corresponding anomaly-score matrix is in the bottom right. Values in each row of the anomaly-score matrix are calculated using the same time series model. We mark the training window by a horizontal line on the left of that row and record the anomaly scores in the corresponding columns. For illustration purposes, instead of writing down the real anomaly score values, we use H (red), M (blue) and L (green) to indicate high anomaly scores, median anomaly scores, and low anomaly scores respectively in Figure 2.7.

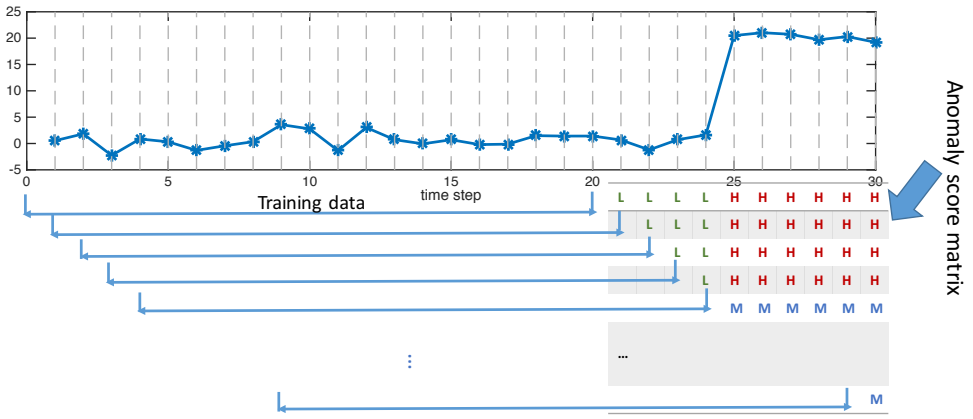


Figure 2.7: The construction of the anomaly score matrix for a time series. The anomaly scores, in general, are continuous values. For the illustration purposes, we separate them into three categories and use H (red), M (blue) and L (green) to indicate high anomaly scores, median anomaly scores, and low anomaly scores respectively.

In anomaly-score matrix, as we discussed above, each time-series model generates multiple anomaly-scores for different time-steps. Simultaneously, each time-step gets multiple anomaly-scores. When we generate the anomaly-score matrix, we keep all the anomaly-scores for a single time-step in the same column (as shown in Figure 2.7). One interesting fact is that all the anomaly scores of a single time-step are obtained using different training sets. Such a novel data structure has several advantages.

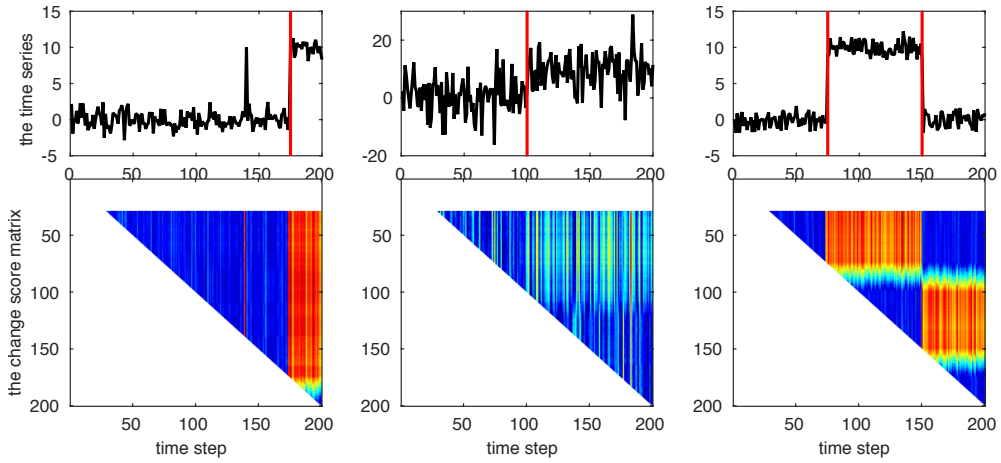


Figure 2.8: Examples of the anomaly score matrix for different time series.

First, outliers and change-points have easily discernible patterns in the anomaly-score matrix. The left panel in Figure 2.8 shows a time series (the top panel) and its corresponding anomaly-score matrix (the bottom panel). There are one outlier and one change-point in the time series. In the anomaly-score matrix, we use red for high anomaly-score values and blue for low values. Change-points have the persistence property and hence there is a patch of high anomaly scores (orange color) after the change point. In contrast, outliers are non-persistent anomalies and hence an outlier is shown in an anomaly-score matrix as one column of high values.

Second, the anomaly-score matrix also enables us to robustly detect change-points in a noisy time-series. The middle panel in Figure 2.8 shows a noisy time-series (top panel) with its anomaly-score matrix (bottom panel). Although the time-series is so noisy that any single anomaly-score (each cell in the matrix) is not strong enough to make a decision, we can still observe a patch with relatively high anomaly scores right after the change-point. This increases our ability to identify change-points in highly variable data.

Finally, the anomaly-score matrix allows us to easily identify multiple change-points in a single time-series. The right panel in Figure 2.8 shows a time-series with two change-points (top panel) and its anomaly-score matrix (bottom panel). We can clearly identify the two change-points as the time-steps preceding the high anomaly-score blocks in the bottom panel.

While Algorithm 2.4 has many advantages as we discussed above, the anomaly-score matrix itself still cannot reduce the negative impact of outliers on model estimation. To overcome this challenge, we need to incorporate the central method in the algorithm. Algorithm 2.5 is a more comprehensive algorithm that calculates the anomaly-score matrix using the central method.

Algorithm 2.5: *calAnomalyMatrixCentral*

Input: $f(\cdot)$, a TS model
$g(\cdot)$, an anomaly score function
$\boldsymbol{x}$, a TS data
Θ: User defined parameters;

Output: $\boldsymbol{A}$, An anomaly score Matrix

```
for any x_t ∈ x do
    for r from 1 to β do
        x' = sample(x(t − w, t − 1));
        θ̂ = est(θ|x', f);
        for i from t to the end do
            B_{r,i} = g(θ̂, x_i) ;
        end
    end
    for i from t to the end do
        A_{t,i} = med(B_{1,i}, ··· , B_{β,i});
    end
end
```

Change-score time-series calculation: $calChangeScoreTS(\cdot)$

A change-point can be observed from an anomaly-score matrix as the boundary of a low anomaly score triangle block on the left and a high anomaly-score rectangle block on the right. When only one single change occurs in a time-series, as shown in the first two plots in Figure 2.8, the change-point is the column where the matrix is partitioned into the two most contrasting "sub-matrices". However, when there are multiple change-points, as shown in the right panel in Figure 2.8, scoring change-points becomes a non-trivial

problem.

We use Figure 2.9 to illustrate the method we use to calculate change-scores. The detailed psedo-code is shown in Algorithm 2.6. The left panel in Figure 2.9 shows a time-series and its anomaly-score matrix. To account for multiple change events, we use a sliding window approach. We adopt Matlab notation for matrices. Let $\boldsymbol{M}(i_1 : i_2, j_1 : j_2)$ be the sub-block in a matrix $\boldsymbol{M}$ that starts at the i_1^{th} row and ends with the i_2^{th} row, and starts at the j_1^{th} column and ends with the j_2^{th} column. Using this notation, we define $\boldsymbol{B}_t$ (the reference area of time step t) as the upper triangular part of $\boldsymbol{M}(t-c:t-1, t-c:t-1)$ and $\boldsymbol{A}_t$ (the test area of time step t) as the sub-matrix $\boldsymbol{M}(t-c:t-1, t+1:t+p)$, where p and c are the parameters that control the size of the reference area and the test area.

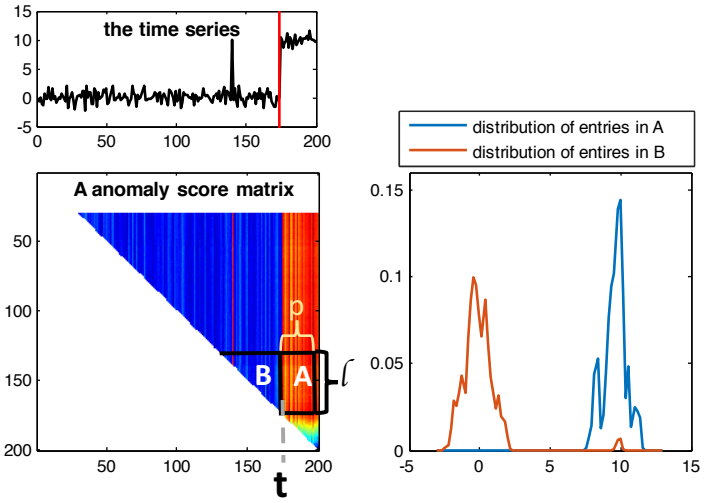


Figure 2.9: Illustration of the change-score calculation method. The left: an illustrative example of how we report change events from an anomaly score matrix. The right: the distribution of entries in sub-matrix A and sub-matrix B. The change points can be detected as the column that maximizes the difference between the two distributions.

This proposed scoring method allows us to easily combine consistency and persistence properties. First, the length of both the reference area and the test area indicates the *consistency* of an anomaly event – that is, an anomaly, whether an outlier or a real change, has high anomaly-scores based on multiple training sets. We use parameter c to control it. Second, the width of the test area indicates the *persistence* of a change

event – that is, how long does the time-series model remain different from a training period. We use parameter p to control it.

We can detect the change points as the time-steps such that the difference between $\mathbf{A}_t$ and $\mathbf{B}_t$ are maximized. One way to quantify the difference between $\mathbf{A}_t$ and $\mathbf{B}_t$ is by the distribution of their entries (anomaly scores). The right panel in Figure 2.9 is the distribution of the anomaly scores within $\mathbf{A}_t$ and $\mathbf{B}_t$ in the left panel. Here, we opt for a two-sample student t-test because of its modest computational costs. The final change score for time step t can be obtained by function $scoring(.)$.

$$scoring(x_t) = \frac{\|mean(\boldsymbol{A}_t) - mean(\boldsymbol{B}_t)\|}{\sqrt{var(\boldsymbol{A}_t)/count(\boldsymbol{A}_t) + var(\boldsymbol{B}_t)/count(\boldsymbol{B}_t)}}$$

where, mean(.), var(.), and count(.) are the mean, variance, and number of entries in the sub-matrix, respectively.

Algorithm 2.6: *calChangeScoreTS*

Input: $\boldsymbol{A}$: an anomaly-score Matrix;
Θ: user defined parameters;

Output: $\boldsymbol{y}$: an anomaly-score TS;

```
for any valid t do
    A_t = M(t − c : t − 1, t + 1 : t + p) ;
    B_t = M(t − c : t − 1, t − c : t − 1) ;
    y_t = scoring(A_t, B_t);
end
```

Parameter selection: Θ

The PC framework has a set of user defined parameters, which is denoted as Θ in all the above pseudo-codes. Parameters in Θ belong to three categories. The first category is parameters that are introduced by the selected time-series method. One example is the size of the sliding window used to train the time series model (*i.e.*, the parameter w in Algorithm 2.1 - Algorithm 2.5).

The second categories include two values (*i.e.*, p and c in Algorithm 2.6) that control the size of the reference area and the test area. Since the performance of the two-sample student t-test is better when the sample size is larger, we should make p and c a large

number. However, if there is more than one change point in a time series, one might include information from previous change periods when p and c are too large. Therefore, in general, p and c should be a large number but smaller than the shortest time duration between two change points.

The final category is the parameter used in the central method. The central method contains one parameter β, which is the times of random sampling we do in the anomaly score calculation. β is bounded by the requirement of algorithm accuracy (See Lemma 2.1) and the requirement of the computation cost. When β decreases, the accuracy of the central method drops. When β increases, the computational cost increases. Hence, for β, we can select any number within the acceptable region. The lower bound of the acceptable region is the smallest value that matches the error rate requirement. The upper bound is the largest value that satisfies the computational cost requirement.

2.4 Evaluation

We illustrate the performance of the proposed PC framework through two applications: (i) autonomous forest fire detection from satellite images and (ii) Activity segmentation using chest-mounted accelerometer data.

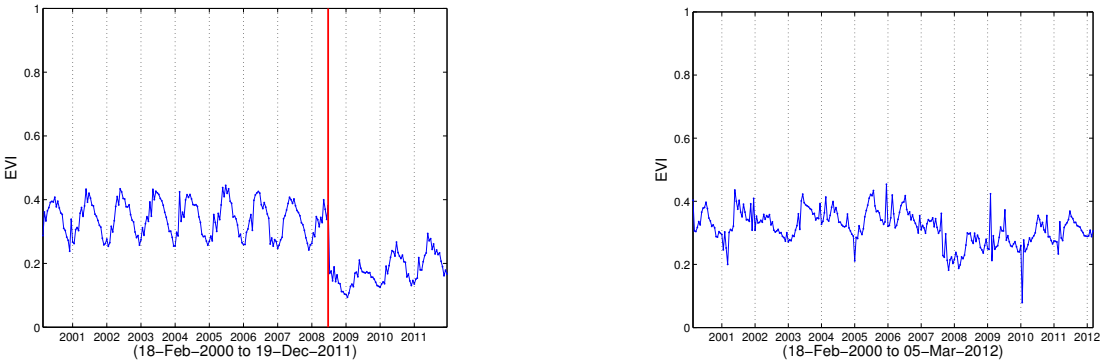
2.4.1 Autonomous forest fire detection from satellite images

We first evaluate the performance of the PC framework by its ability to detect forest fires from satellite data.

Experimental setup

The dataset that we use is the Enhanced Vegetation Index (EVI) data [87]. EVI data measure the surface "greenness" as a proxy for the amount of vegetated biomass at a particular location. Forest fires cause an abrupt change on the land surface due to vegetation loss, which can be characterized as abrupt changes in the EVI time series [24]. Figure 2.10 shows two EVI time-series for two burned locations from different geographic regions. Applying most existing time-series change detection methods to identify forest fires from EVI data has three challenges: (i) a significant number of time-series are very noisy and contain outliers, (ii) the time-series are periodic, and (iii)

the data are very large ($\sim$ 1 billion time-series). In this experiment, we aim to show the robustness of the proposed framework to outliers and noise.



(a) Burned in 2008.

(b) Burned in 2007.

Figure 2.10: Example EVI time-series from two locations where forest fires occurred. The left time-series is from Northern California while the right one is from Southern California.

To demonstrate the generalization ability of the PC framework, we choose two ecologically varied regions in the state of California (US). Each experimental region contains roughly two-thousand time series. The first region is near San Diego (the region between $32.85^oN \sim 32.53^oN$ and $116.4^oW \sim 117^oW$) and the second region is near Salinas (the region between $36.4^oN \sim 35.9^oN$ and $121.2^oW \sim 122^oW$). These two geographic areas represent diverse regions with different variability, land cover types, geography, and noise characteristics.

We select four baseline methods that can detect changes in periodic time-series. The baseline methods are (i) seasonal ARIMA [47]; (ii) adaptive CUSUM (CUSUM method after removing the periodicity of the time series); (iii) Gaussian process method with periodic model (GP) [17] and (iv) Variance Independent Drop (VID) [67] – an algorithm designed to specifically detect sudden drops in remote sensing time series. We Local Instant Drop (LID) [24] as the input change detection method in PC framework. LID is very sensitive to outliers and noise and has poor performance if directly used in this experiment.

We use manually curated data to quantitatively evaluate and compare the performance of the proposed PC framework and the four baseline algorithm. These fire

validation data are from government agencies responsible for monitoring and managing forests and wildfires. According to these validate data, we assign a label for each time series: 0 for unburned, 1 for burned, and 2 for partially burned[3] . We consider a time-series to be *positive* if it is totally burned (label is 1) and negative if it is totally unburned (label is 0). The remaining pixels (partially burned) are discarded from our evaluation to avoid ambiguity.

When reporting events, each detection method provides one change-score to each time-series. The reported change-score of a time-series is the maximum value in the corresponding change-score time-series. Given change-scores of all time-series, there are two ways to mark a time-series as positive. We can either report all the time-series whose change-scores are larger than a specific threshold as positives or label the time-series with the top-k change-scores as positives. For this experiment, we choose the second method.

		Predicted	
		Fire	No Fire
Validation Data	Fire	TP	FN
	No Fire	FP	TN

Table 2.1: The confusion matrix for the fire detection experiment.

Forest fires are rare events. We use *Precision* and *Recall* as evaluation metrics to quantitatively compare the various methods [84]. Given a number of k, each algorithm returns a set of positive and negative events. Comparing the detected results against the fire validation data, we obtain the number of true positives (TP_k), false positives (FP_k), false negatives (FN_k) and true negatives (TN_k) for each algorithm (as shown in the confusion matrix in table 2.1). Precision (p_k) and recall (r_k) are then computed as:

$$p_k = \frac{TP_k}{TP_k + FP_k}$$

[3] Although government agencies make their best effort in documenting historical fires, fire perimeter data are neither complete nor without error due to the finite resources available to any agency. However, inaccuracies and incompleteness are represented only in a small portion of the validation data, and these data are still useful for quantitatively comparing methods across large spatial regions.

$$r_k = \frac{TP_k}{TP_k + FN_k}$$

We illustrate the performance of different algorithms using precision curves and recall curves. A precision (or recall) curve is a line that shows how the precision (or recall) values change with k, *i.e.* the number of detected positive points.

Experimental results

Figure 2.11 shows the precision (in the top panel) and recall (in the bottom panel) curves for each method in both Northern California (in the left panel) and Southern California (in the right panel). The PC framework has better precision and recall values in both regions, especially when k is not too large. Thus, the proposed PC framework provides more reliable results. Since all methods can handle periodicity well, the major improvement of the PC framework is because of its robustness to noise and outliers. Another observation from the results is that most methods perform reasonably well in Northern California but our method is much better than the others in Southern California. This is because the data in Southern California are more heterogeneous and noisy.

2.4.2 Activity segmentation through chest-mounted accelerometer data

The second experiment illustrates the performance of the proposed PC framework by its ability in activity segmentation through chest-mounted accelerometer data. There has been work in time-series segmentation using this dataset (*e.g.* [14]). However, a change-point detection approach may be more suitable especially for real-time monitoring. This application is challenging since the data are highly variable with changing periodicity (*e.g.* different behavior when walking and standing).

Experimental setup

The chest-mounted accelerometer data are from the UCI Machine Learning Repository [14, 61]. This dataset records the uncalibrated accelerometer signals from fifteen users when they are asked to do seven activities: (i) work at a computer, (ii) stand up, walk and go up/down stairs, (iii) stand, (iv) walk, (v) go up/down stairs, (vi) walk and talk

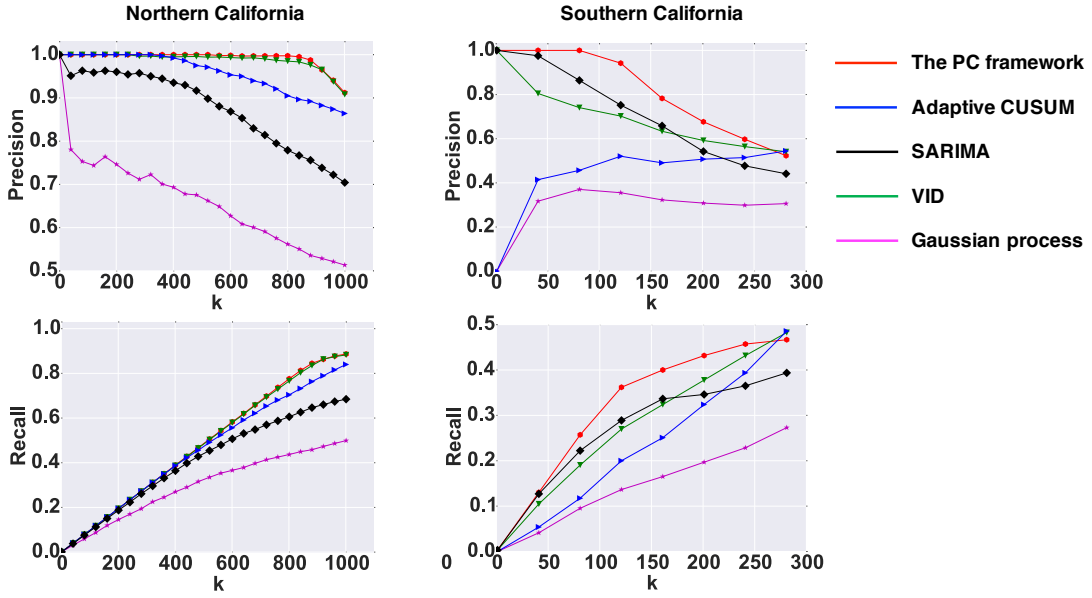


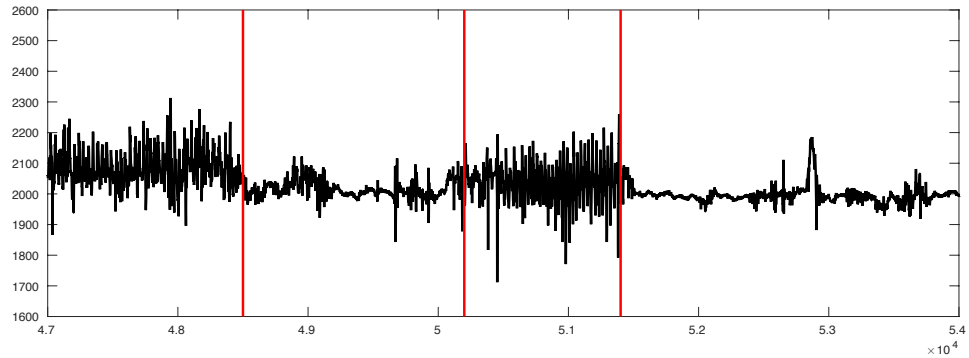
Figure 2.11: The precision and recall curves as a function of k of the proposed PC framework against the four baseline methods (best seeing in color).

with someone and (vii) talk while standing. The dataset contains fifteen time-series (one per user). Each time-step in any of the fifteen time-series is a three-dimensional variable. Besides, each time-step also has its activity label, which we use as the validation data. All the fifteen time-series have the same sample rate (*i.e.*, 52HZ) but their lengths vary from sixty thousand to a hundred and sixty thousand time-steps.

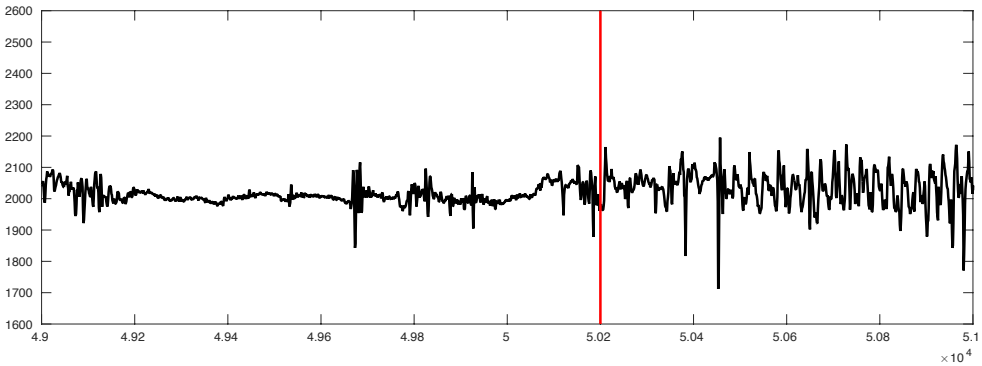
In this experiment, we detect change-points (the time when a user changes his/her activity type) using the y-acceleration field. Figure 2.12(a) shows a portion of y-accelerometer data (the black curve) and its corresponding change-points[4] (marked by the red lines). To visualize the property of the time-series in detail, we also zoom into the sub-sequence of the time series (shown in Figure 2.12(a)) around the second change-point. The sub-sequence is shown in Figure 2.12(b).

We use the activity labels provided in the dataset to generate change-points as the validation data. We name this validation data "reported change-points". We find that reported change-points are consistently prior to the actual change-points. To illustrate

[4] Here, we use the adjusted change-points. The detailed information of adjusted change-points is provided in the following paragraph.



(a) A portion of the y-acceleration field in a chest-mounted accelerometer time-series data.



(b) The zoomed-in sub-sequence of the above time-series around the second change point.

Figure 2.12: Y-accelerometer data with the reported change-points in the chest-mounted accelerometer data.

this fact, we plot a segment from a time-series with its reported change-points (the green lines) in Figure 2.13. In our evaluation, we manually shift all change-points by a constant value and use the "adjusted change-points" as the final evaluation data. The adjusted change points of the given time-series in Figure 2.13 are marked by red dot-lines.

We detect the beginning of a segment as a time-point when the variance of the given data changes. We compare our algorithm with two baseline algorithms: VarCUSUM and Binary Segmentation (BinSeg). VarCUSUM first estimates the variance of each time-point using data around it, and then applies CUSUM method to discover change-points. The second baseline, BinSeg, has been used to monitor changes in the variance

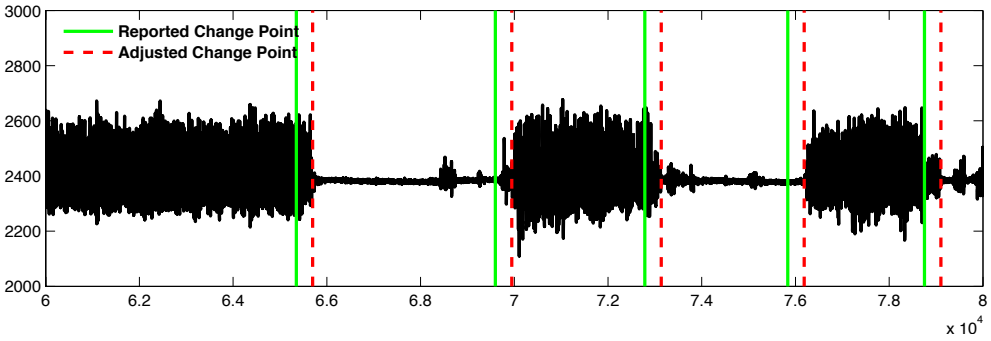


Figure 2.13: A segment of a chest-mounted accelerometer time series with the reported change points (the green lines) and the adjusted change points (the red dot-lines).

of a time-series [52, 79]. The change detection method that we use in our framework calculates the variance of two consecutive time windows and the anomaly score is the ratio of the two.

Similar to the previous experiment, we use the precision curve and the recall curve as the metrics to compare across methods. Since there are multiple change-points in one time-series, we follow the evaluation mechanism used in [63]. In detail, we define

- TP_s: Detected change-points that are located within s seconds of true change-points.
- FP_s: Detected change-points that are NOT located within s seconds of true change-points.
- FN_s: True change-points within s seconds of which there is no detected change-points.

The precision and recall values are calculated for each time-series independently. Therefore, each algorithm has fifteen precision values and fifteen recall values for a given k and s. In our experimental results, we report both the mean and the standard deviation of the fifteen precision and recall values.

Experimental results

Figure 2.14 shows the comparison between the PC framework and the baseline methods. In detail, we shows the performance of the proposed PC framework (red) and the two baseline methods (green for VarCUSUM and pink for BinSeg) when $s = 10$, $s = 5$ and

$s = 2$. As the evaluation method we use in Section 2.4.1, each plot demonstrates how the precision (or recall) values change with the selection of k.

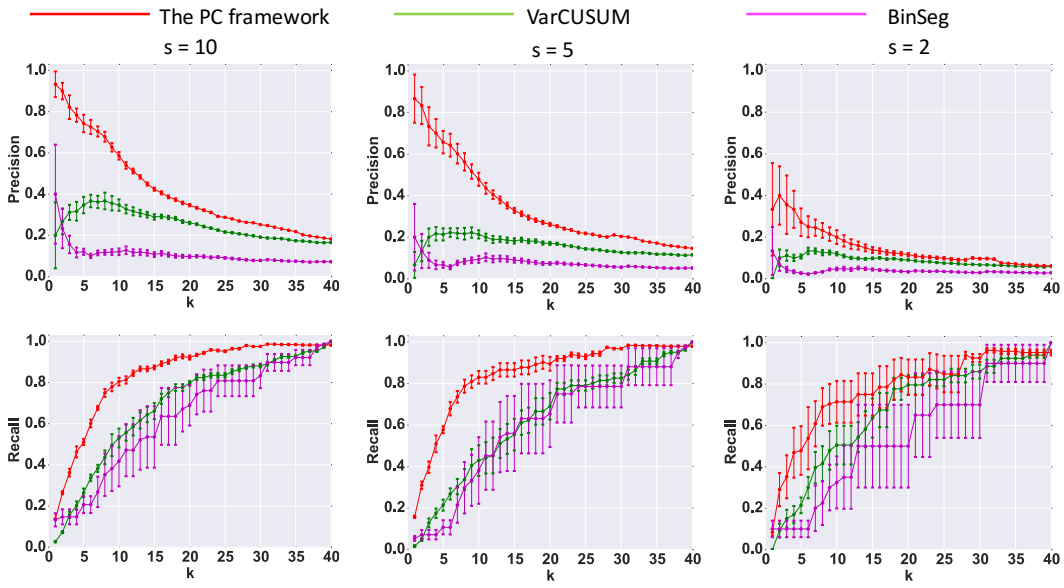


Figure 2.14: The performance of the PC framework against the two baselines on the activity segmentation through chest-mounted accelerometer data. Mean precision and recall curves of the top-k change points for 15 time series (better seen in color). The error bars show the variance of precision/recall values across time series.

As shown in Figure 2.14, the performance of the proposed PC framework is better than the other two methods for all the s values. Our experimental setup provides three information. First, the data are very noisy (as shown in Figure 2.12. Second, the time-series model and the anomaly-score function used in the PC framework is very similar to the one used in the VarCUSUM. Third, each time series contains multiple change-points. Therefore, we conclude that (i) the PC framework can increase the robustness of a change-point detection method to noise and outliers and (ii) the PC framework can be used to detect multiple change-points in one time series.

Figure 2.14 also shows that the performance (both the precision and recall values) of all the three methods decrease with s values. This fact may indicate that the variance-based methods have difficulties in detecting the exact time when a user changed his/her activity. Since we consider this as the limitation of the variance-based method, we do

not provide detailed analysis on it in this chapter.

2.5 Conclusion

We present a framework that makes existing modeling-based change-point detection algorithms robust to noise and outliers. Our work makes two major contributions to the "modeling-based" time-series change-point detection literature. First, we introduce the central method for anomaly-score estimation. The central method is a sampling-based method that enables us to be more resilient to outliers. Second, we propose a Persistence-consistency framework that allows us to intuitively identify change-points and estimate their statistical significance. The proposed framework shows promising results on two real-world applications, and even outperforms application-specific algorithms (*e.g.* VID).

Our algorithm has several built-in parameters, such as the number of time points used to train the time-series. These parameters need to be assigned by the user, which can be a nontrivial task. Thus, a method that autonomously learns these parameters is a valuable future research question. Additionally, we use the student t-test to measure the significance of a change event. Such test assumes the samples follow a normal distribution. In the applications where this assumption is violated, the proposed method may not work well. Thus, investigating other significance tests is another potential topic in the future research. Finally, the proposed method, although improving the robustness of an algorithm to outliers and noise, increases the computational cost. Thus, designing the parallel version of this framework is critical for applying this framework to time-sensitive applications.

Chapter 3

S-CTC detection:

Singleton contextual time-series change-point detection

3.1 Introduction

Time-series change-point detection methods aim to discover change points from time-series data. Most existing detection methods (*e.g.*, BFAST [88], CUSUM [71, 72], and many other algorithms [5, 46, 47, 78, 83, 86]) define change points with respect to values in the previous portion of the same time series. By contrast, a *Singleton Contextual Time-series Change point* (or S-CTC) refers to the time when the behavior of *one* time series (referred to as the target time series) starts to deviate from its own *context*. The context, in this work, consists of several time series that exhibit similar behaviors to the target time series before the change point.

Figure 3.1 demonstrates the differences between S-CTCs and traditional change points. In Figure 3.1, we show the target time series using a black line. The context of each target time series in this example is 20 time series. We use gray-color lines to indicate these 20 time series. Each column contains a pair of plots. In each pair, the target time series are identical but their contexts are different. As a result, in any pair, one time series contains a S-CTC and the other one does not. More precisely, all

three target time series in the top row behave similarly to their contexts and hence do not contain any S-CTCs. In contrast, all target time series in the bottom row behave differently from their own contexts after the blue vertical line. Therefore, these three time series change contextually. Note that the target time series in the left column do not have any traditional change point, while each target time series in the middle and right columns does have one traditional change point.

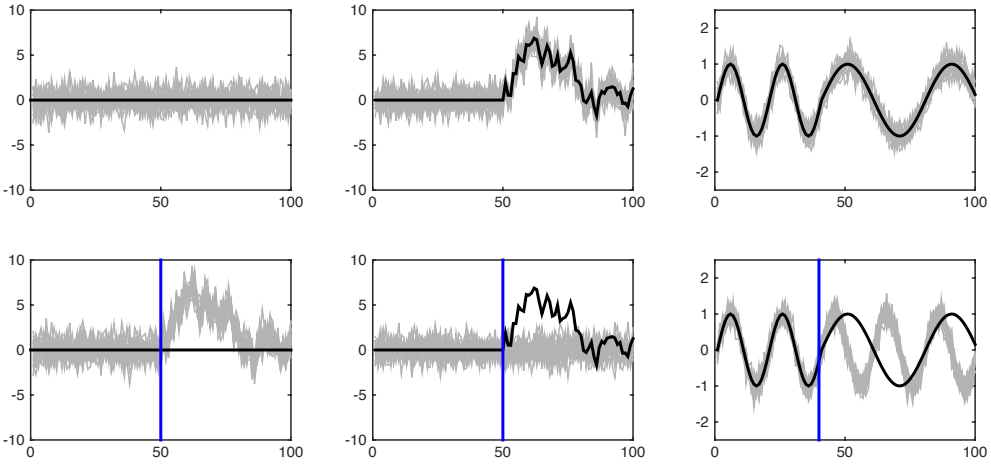


Figure 3.1: The comparison between S-CTCs and traditional change points. In each column, the two target time series (black lines) are identical. None of the target time series in the top row change contextually, while the target time series in the bottom panel do change contextually. The S-CTCs are marked by blue vertical lines.

Discovery of S-CTCs from time-series data is useful in many real-world settings. For example, consider a collection of sensors that measure the temperature in a factory. It is very likely that the sensors exhibit a diurnal pattern, with temperatures increasing during the morning hours and cooling in the evenings. Thus, a change in the sensor readings does not necessarily constitute an event. However, if a single sensor *does* change differently from the others, it may indicate a sensor failure or an unusual condition in the sensor environment. Similar situations arise in a wide range of application settings.

In this chapter, we first formally define Singleton Contextual Time-series Change points (S-CTCs). Then, we propose an approach to detect S-CTCs from a time-series dataset. Finally, we perform qualitative and quantitative evaluation on the proposed method in datasets from two different real-world domains: historical stock market data

and remote sensing data. The experimental results show that S-CTCs indicate a new type of events. These events typically cannot be discovered by traditional change-point detection methods.

We have four innovations in this work: (i) We introduce a novel type of change point, S-CTCs, to the data mining community; (ii) We propose a new similarity function called k^{th} *order statistic distance* to increase the robustness of our proposed method to outliers; (iii) We use a new metric called *time-series area depth* to measure the deviation of a target time series from its own context; and (iv) We introduce a heuristic algorithm, Multimode Remove, to solve the multimodal problem in the proposed method.

3.2 Notations and problem formulation

In this section, we first introduce the assumption of this work and the mathematical notations that are used in this chapter. Then, we formally define Singleton Contextual Time-series Change points (S-CTCs).

Assumption 3.1 (The properties of the datasets)**.** *We detect S-CTCs from a time-series dataset that has the following properties:*

1. *All input data is real-valued.*
2. *Observations at the same time step in all time series are collected simultaneously. We do not require that time steps in a time series are equally spaced in time.*
3. *Recent historical data is accessible. Considering that the data volume may be large, we do not require that all the historical data are stored offline. But, we assume that we can access a certain amount of historical data. The detailed amount of data varies with applications.*
4. *We DO NOT assume that the time series follows stationary, periodic stationary or other known time-series models [76].*

The second condition in Assumption 3.1 is difficult to satisfy completely. In many cases, we can relax this condition and assume that data in the same time step are collected within a short time period (*e.g.*, in the same hour or the same day).

Notations. *We follow the MATLAB style and use the mathematical notations below.*

- *A boldface upper case letter (e.g., $\boldsymbol{X}$ and $\boldsymbol{Y}$) is a time-series matrix and stands for a time-series dataset or a group of time series. Each row in the matrix is a time series. $\boldsymbol{X}(i,:)$ denotes the i^{th} time series in $\boldsymbol{X}$. $\boldsymbol{X}(i,t)$ denotes the observation at the t^{th} time step of the i^{th} time series in $\boldsymbol{X}$.*

- *A boldface lower case letter (e.g., $\boldsymbol{x}$ and $\boldsymbol{y}$) stands for a time-series data. $\boldsymbol{x}(t)$ denotes the observation at time step t of $\boldsymbol{x}$. $\boldsymbol{x}(i:j)$ denotes observations of time series $\boldsymbol{x}$ from time step i to time step j.*

- *A non-boldface lower case letter stands for a time step. For example, x_t is the observation at time step t of time series $\boldsymbol{x}$.*

A time series changes contextually when its observations begin to consistently deviate from its context. Although various sorts of contexts (*e.g.*, predefined time-series patterns) can be used, we define the context of a time series as a group of other time series that are similar to the target time series within a certain time duration. We name this type of context the *Dynamic Peer Group (or DPG)* of the target time series. More precisely, we define the DPG of any time series $\boldsymbol{x}$ as below.

Definition 3.1 (Dynamic Peer Group (DPG))**.** *A group of time series is the DGP of time series $\boldsymbol{x}$ at time step t (which is referred to as $\boldsymbol{G}_t$) if and only if any time series $\boldsymbol{y} \in \boldsymbol{G}_t$ satisfies the following condition:*

$$d\left(\boldsymbol{x}(t-w:t-1), \boldsymbol{y}(t-w:t-1)\right) < \delta$$

where, $d(.,.)$ is an arbitrary distance function that measures the difference between two sub-sequences. w and δ are two user-defined parameters. w is the length of the DPG construction window and δ is the threshold used to judge whether or not two sub-sequences are similar.

A S-CTC is the time when a time series starts to deviate from its DGP. Formally, S-CTC is defined as below.

Definition 3.2 (Singleton Contextual Time-series change points (S-CTCs))**.** *x_t is a S-CTC of time series $\boldsymbol{x}$ if and only if*

1. *$\boldsymbol{G}_t$, the DPG of $\boldsymbol{x}$ at time step t, exists.*

2. $\boldsymbol{x}$ *behaves differently from* $\boldsymbol{G}_t$ *after time step* t*, i.e.,*

$$d_G\left(\boldsymbol{x}(:, t:t+l), \boldsymbol{G}_t(:, t:t+l)\right) > \beta$$

where, $d_G(.,.)$ *is a distance function that measures the difference between a sub-sequence and a group of sub-sequences.* l *and* β *are two user-defined parameters.* l *controls the length of the scoring window and* β *controls the smallest deviation between the target time series and its DPG when a S-CTC occurs.*

3.3 Related work

Most existing change-point detection methods focus on discovering the time when a time series begins to be significantly different from its own historical data [5, 9, 21, 45–47, 54, 78, 83, 86]. These methods typically detect change points of each time series independently. Many well-known change-point detection methods (*e.g.* BFAST [88], CUSUM chart [72], ARIMA-based methods [47], and the Periodic Gaussian Process method [17]) are examples of these "traditional" change-point detection methods.

A S-CTC, as defined in Definition 3.2, is the time when the relationship among more than one time series is altered. Such contextual changes in time series are a relatively novel concept and until now little research has been done on this topic. However, there is prior research in analyzing the behavior of a target time series with respect to a group of related time series in the field of fraud and/or outlier detection [8, 57, 89].

Peer Group Analysis (PGA) is an unsupervised fraud detection method [8, 89]. PGA labels time steps as fraud points if those values depart from the observations at the same time step of the time series in the corresponding peer group. A peer group of a time series in PGA is the k nearest neighbors of the target time series that are constructed using only the first few time steps. There are two major differences between our approach and PGA. First, the peer group of a time series is unchangeable throughout the analysis process. In contrast, our scheme re-constructs a peer group for each time step. This allows our scheme to handle cases where the peer group of a time series changes with time. Second, because PGA focuses on fraud detection, it is not as effective in detecting change points that are more reliably observed over multiple consecutive time steps, especially in noisy data.

Temporal Outlier Detection (TOD) [57] also focuses on detecting outliers with respect to other time series in the dataset. Instead of constructing a peer group for each time series (as is done in our approach and PCA), TOD maintains a series of Temporal Neighborhood Vectors (TNVs) for each time series. The TNVs of a time series contain the similarity values between the target time series and all other time series in the dataset for all time steps. The outlier score of the target time series at any time step is the L_1 distance between the TNV at the current time step and its TNV at the previous time step. Temporal outliers and S-CTCs are entirely different types of patterns. Figure 3.2 demonstrates the differences. As shown in Figure 3.2, all the time series in the same color have similar behaviors during the entire time-period. Therefore, all time series in the figure do not change contextually. However, compared with the gray time series, the black time series have higher observations before time step 100 and lower observations after. Such changes can be detected by TOD as temporal outliers.

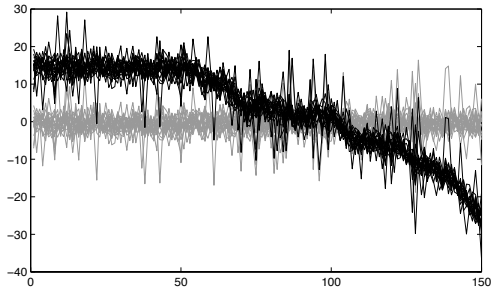


Figure 3.2: Temporal outliers and S-CTCs are different patterns. The black time series and gray time series have similar behaviors over time within each group. None of the time series change contextually. However, due to the changes in the relationship between the black group and the gray group, TOD may report many temporal outliers.

3.4 Proposed method

We propose an algorithm to discover S-CTCs from a time-series dataset. Algorithm 3.1 is the pseudo-code. The proposed method has two types of inputs: a time-series dataset $\boldsymbol{X}$, and a set of user-defined parameters $\boldsymbol{\Theta}$. The proposed method outputs a time-series matrix $\boldsymbol{Y}$. Each row of $\boldsymbol{Y}$ is the *change-score time series* of the corresponding time series in $\boldsymbol{X}$. The proposed method (as shown in Algorithm 3.1) is a two-step approach. For any time step t in any time series $\boldsymbol{x}$, we first construct the DGP of $\boldsymbol{x}$ at time step

t using the function $DPGConstruction(.)$ (Line 4). Then, we calculate a change score using the scoring function $TAD(.)$ (Line 5).

Algorithm 3.1: The proposed S-CTC detection method

Input: $\boldsymbol{X}$: a time-series dataset that contains n time series;
$\boldsymbol{\Theta}$: user defined parameters;

Output: $\boldsymbol{Y}$: change-score time series;

```
1 for i from 1 to n do
2     x = X(i, :);
3     for any valid t do
4         G_t = DPGConstruction(x, X, t, Θ) ;
5         Y(i, t) = TAD(x, G_t, t, Θ);
6     end
7 end
```

3.4.1 DPG construction: *DPGConstruction(.)*

As the name suggests, the DPG construction function is to construct the DPG for a target time series at a given time step. Algorithm 3.2 is the pseudo-code of function $DPGConstruction(.)$.

Algorithm 3.2: *DPGConstruction*

Input: $\boldsymbol{x}$: a time-series data;
$\boldsymbol{X}$: a time-series dataset;
t: a time step;
$\{\delta, w, k\}$: user-defined parameters;

Output: $\boldsymbol{G}_t$: the DPG of $\boldsymbol{x}$ at time step t;

```
1 for any time series y in X do
2     α = d_k(y(t − w : t − 1), x(t − w : t − 1));
3     if α < δ then
4         Put y into G_t
5     end
6 end
```

Definition 3.3. *(δ-NEIGHBOR OF A TIME SERIES) We say time series $\boldsymbol{y}$ is a δ-neighbor of time series $\boldsymbol{x}$ if and only if the distance between $\boldsymbol{y}$ and $\boldsymbol{x}$ is smaller than δ.*

Roughly speaking, the DPG construction function is to find all the δ-neighbors of a target time series (Line 2 - 5). The distance metric that is used to find the δ-neighbors is the key component in $DPGConstruction(.)$. The selected distance function may vary with applications. In this work, we use a modified Minkowski distance, $d_k(.)$ or k^{th} order statistic distance.

While Minkowski distances have been commonly used in time-series analysis [57], they perform poorly when time-series data contains outliers. Previous studies [22] have shown that smoothing methods (*e.g.*, moving average or Savitzky-Golay filter) cannot fully address the problem of outliers. One of the major reasons is as below. Smoothing methods typically assign a new value to any time step using a weighted (or unweighted) average of several consecutive time steps. Thus, they can reduce the anomalous level of an outlier. However, such an averaging process also increases the anomalous level of multiple normal data points near the outlier. When calculating a Minkowski distance, it is highly possible that at least a fraction of anomalous information is used. To address the above challenge, the proposed method uses a new time-series distance metric k^{th} *order statistic distance.*

Definition 3.4. *(k^{th} ORDER STATISTIC DISTANCE $d_k(.)$) The k^{th} order statistic distance of two subsequences $\boldsymbol{x}(i:j)$ and $\boldsymbol{y}(i:j)$ is calculated using the following steps. Note that the proposed distance function is based on the Minkowski distance of order p.*

1. *Calculate $\boldsymbol{z}(i:j)$, the point-wise distance time series, from $\boldsymbol{x}(i:j)$ and $\boldsymbol{y}(i:j)$, where each time step t in $\boldsymbol{z}$ is obtained as below:*

$$z_t = |x_t - y_t|$$

2. *Rank $\boldsymbol{z}(i:j)$ in ascending order and obtain a new subsequence $\boldsymbol{\alpha}(i:j)$.*
3. *The k^{th} order statistic distance between $\boldsymbol{x}(i:j)$ and $\boldsymbol{y}(i:j)$ is*

$$d_k(\boldsymbol{x}(i:j), \boldsymbol{y}(i:j)) = \left(\sum_{t=i}^{j-k} \alpha_t^p \right)^{1/p}$$

k^{th} order statistic distance introduces a user-defined parameter k to the proposed method. To a certain extent, the performance of the proposed method depends on the choice of k. Roughly speaking, if k is too small, outliers may still be included in the distance calculation. Hence, some time series may be mistakenly excluded from the DPG due to outliers. If k is too large, the distance value may be calculated without considering the most different portion between two substances. As a result, some time series that have very different behaviors from the target time series during some period of time may also be included in the DPG.

3.4.2 The scoring mechanism: *TAD(.)*

The scoring step is to assign a change score to each time step. We use *Time-series Area Depth* (TAD) function to quantitatively measure the degree of contextual changes. The proposed TAD is a new distance function that is derived from statistical depth [90].

Definition 3.5. *(*Time-Series Area Depth*, TAD) Given a time series $\boldsymbol{x}$ and its DPG at time step t, $\boldsymbol{G}_t$, the TAD of $\boldsymbol{x}$ at time step t with respect to $\boldsymbol{G}_t$ is as follows:*

$$TAD\left(\boldsymbol{x}, \boldsymbol{G}_t, t\right) = \sum_{i=t+1}^{t+l} \frac{|2x_i - c_i^1 - c_i^2|}{|c_i^1 - c_i^2|}$$

where, l is an user-defined parameter that controls the length of the scoring period. c_i^1 and c_i^2 are the m_1 and m_2 percentiles of $\boldsymbol{G}_t$ at time step i, and m_1, and m_2 are two user-defined parameters.

In this work, we choose $m_1 = 16$ and $m_2 = 84$ because the area within them corresponds to one standard deviation region when data follows the normal distribution. l is a parameter that depends on the application. Typically, we choose a larger value for l if change events have long-term impacts on the time series and a smaller value for l if the target time series may rejoin its DPG shortly after the change point.

TAD is a good measure for S-CTC due to the following five reasons:

1. TAD indicates the change of $p(\boldsymbol{x} \in \boldsymbol{G}_t)$ (*i.e.*, the probability that time series $\boldsymbol{x}$ belongs to a group of time series $\boldsymbol{G}_t$) before and after time step t.
2. TAD is a non-parametric method and does not assume the data distribution. Thus, we can apply it to many real-world datasets.

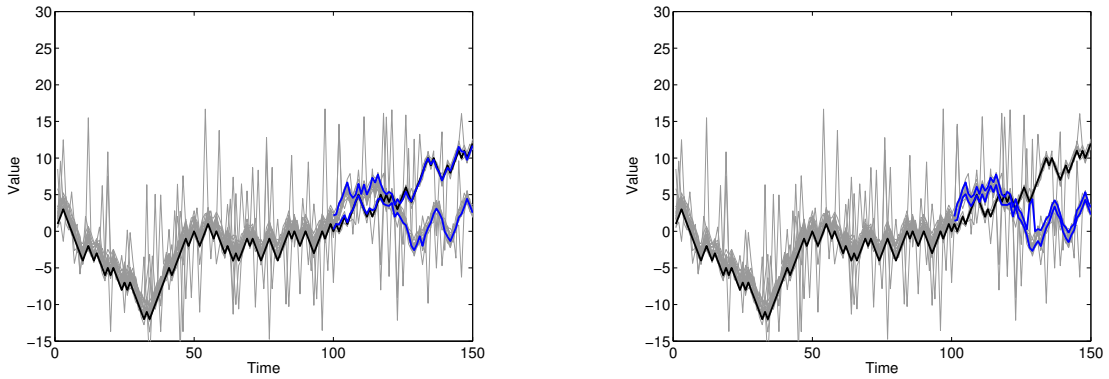
3. It is easy to extend TAD to multivariate time-series analysis. Although we focus on univariate time series in this work, the ability to extend the proposed approach to multivariate analysis is still under consideration.

4. TAD helps address the unequal variance issues of the observations at different time steps.

5. Experiments suggest the TAD is parsimonious. In other words, it performs quite well even if the number of time series in the DPG is smaller than the dimension of the data (i.e., the number of time steps in the scoring period).

3.4.3 Dealing with multiple modes

Some time series in the DPG may change simultaneously with the target time series and exhibit similar behaviors as the target time series after the change point. In this situation, DPG typically shows multimodal behavior, and TAD may underestimate the degree of changes. We name this problem the multimodal problem. Figure 3.3 (a) illustrates this problem. In the given example, 5 out 20 time series in the DPG (gray) are similar to the target time series (black) after the change point. The target time series is still located within the 16 - 84 percentiles of its DGP (the region between the two blue lines) during the scoring period, and hence its TAD score is low.



(a) Without the Multimode Remover algorithm.

(b) With the Multimode Remover algorithm.

Figure 3.3: Illustration of the multimodal problem. The two plots show the same target time series (black) together with its DPG (gray). The blue lines are the 16 and 84 percentiles of the DPG during the scoring period.

We propose a *Multimode Remover* (MR) algorithm to solve the multimodel problem. The MR algorithm removes the small modes in the DPG and keeps only the major one. Essentially, the MR algorithms removes $r\%$ of time series that are farthest from the mean of the DPG each time. This procedure is iteratively performed until the mean of the DPG stabilizes. Figure 3.3 (b) shows the percentiles of the major mode (blue lines) generated by the proposed MR algorithm. Algorithm 3.3 is its pseudo-code. This heuristic algorithm assumes that (i) most time series in the DPG still behave similarly (at least) for a short duration after the DPG construction time; (ii) When a S-CTC occurs, only a small portion of time series in the DPG change similarly to the target time series.

Algorithm 3.3: Multimodal Remover

Input: $\boldsymbol{G}_t$: a DPG constructed at time step t;
$\{l, r, m_1, m_2\}$: user-defined parameters;

Output: $\boldsymbol{c}^1$ and $\boldsymbol{c}^2$: the top and bottom percentiles of the major mode in $\boldsymbol{G}_t$.

```
for i from t+1 to t+l do
    g = G_t(:, i);
    μ = ∞;
    while |mean(g) − μ| > ε do
        μ = mean(g);
        g' = |g − μ|;
        Rank g according to g' by the ascending order;
        Remove the last r% data from g;
    end
    c_i^1 = m_1 percentile of g;
    c_i^2 = m_2 percentile of g;
end
```

3.5 Experimental results

In this section, we demonstrate the capabilities of the proposed S-CTC detection algorithm on two real-world datasets: (i) stock market data and (ii) remote sensing data.

3.5.1 Event detection based on historical stock market data

In this experiment, we detect change points from stock market time series using both the proposed method and the CUSUM chart method [72]. This experiment shows that the proposed S-CTC detection method discovers novel events that cannot be discovered by traditional methods. In detail, S-CTCs detected by the proposed method from the stock market data typically relate to internal events at the affected companies (*e.g.*, release of lower forecasts or changes in management). In contrast, the traditional method (*i.e.*, CUSUM), in many cases, detects changes in the overall financial market (*e.g.*, global recessions).

The input dataset in this experiment is the weekly closing stock prices of S&P 500 companies from January 2000 to December 2009. This data is publicly available at the Yahoo! Finance website. Since individual stock prices differ in scale, we normalize all raw time series into a range [0,1]. In detail, we generate the normalized time-series data (*i.e.*,$\boldsymbol{x}$) from the original stock price data (*i.e.*,$\boldsymbol{o}$) using the following method:

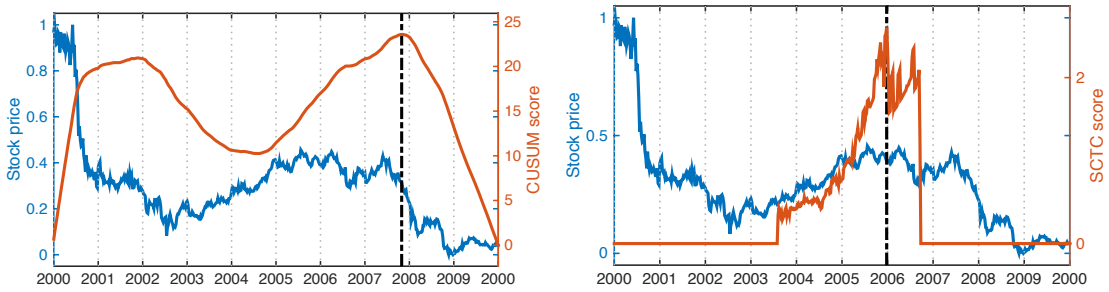
$$x_t = \frac{o_t - min(\boldsymbol{o})}{max(\boldsymbol{o}) - min(\boldsymbol{o})}$$

where $max(\boldsymbol{o})$ and $min(\boldsymbol{o})$ are the maximum and minimum values in time series $\boldsymbol{o}$.

This experiment contains two parts. First, we discuss the differences between S-CTCs and traditional change points using a case study of Sprint Corporation. Then, we briefly demonstrate the performance of the proposed S-CTC detection method on the other historical stock market data. The parameters used in this experiment are: $w = 100$, $k = 1$, and $l = 100$.

A case study of Sprint Corporation

Figure 3.4 shows the normalized weekly closing stock prices of Sprint Corporation (Symbol: S) together with its CUSUM change scores (in Figure 3.4 (a)) and its S-CTC scores (in Figure 3.4 (b)). We use blue lines for the stock price time series and red lines for the score time series. CUSUM detects the time when the mean of a time series shifts. For this specific example, as shown in Figure 3.4 (a), the time step with the largest CUSUM change score is at the end of 2007 (marked by the black vertical dotted line). The end of 2007 is considered as the starting point of the global financial crisis (from



(a) Sprint stock prices vs. the CUSUM scores.

(b) Sprint stock prices vs. the SCTC scores.

Figure 3.4: Comparison between the CUSUM method and the proposed S-CTC detection method in the stock price data of Sprint Corporation.

2007 to 2009) [85]. In contrast, the proposed method detects change points based on the relationship between the target time series and other "similar" time series. As shown in Figure 3.4 (b), the major S-CTC occurs at the beginning of 2006 (marked by the black dotted line) when Sprint Corporation and Nextel Communications reached their merger agreement. Figure 3.5 shows the normalized time series (black) with its DPG (gray). In the plot, we use a black vertical dotted line to indicate the detected S-CTC. Note that this S-CTC is the same one shown in Figure 3.4 (b). We observe from Figure 3.5 that after the change point the Sprint stock leveled off and suffered some decline, while most time series in its DPG continued to rise.

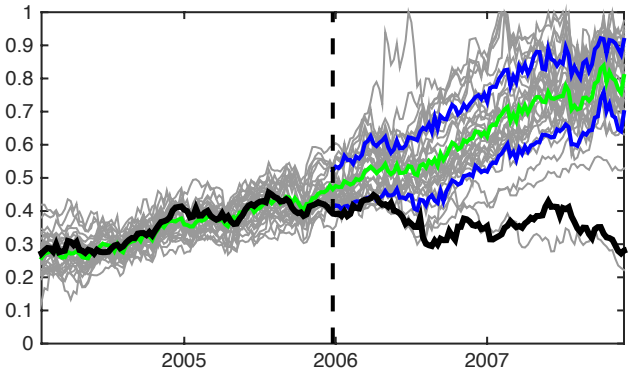


Figure 3.5: The stock price data of Sprint Corporation (a black curve) and its DPG (gray curves) around the time when a S-CTC occurs (a black dotted line). We use blue lines for the 16 & 84 percentiles of the DPG during the scoring period, and a green line for the mean of the DPG.

S-CTCs in the S&P 500 dataset

We discover multiple S-CTCs from the S&P 500 dataset. Since the ground truth is not available in this experiment, we visually examine the results. Figure 3.6 shows the top 9 events (*i.e.*, the 9 time points with the highest change scores). Note that most constructed DPGs contain stocks from multiple industry sectors and thus are good indicators of the overall economic trends.

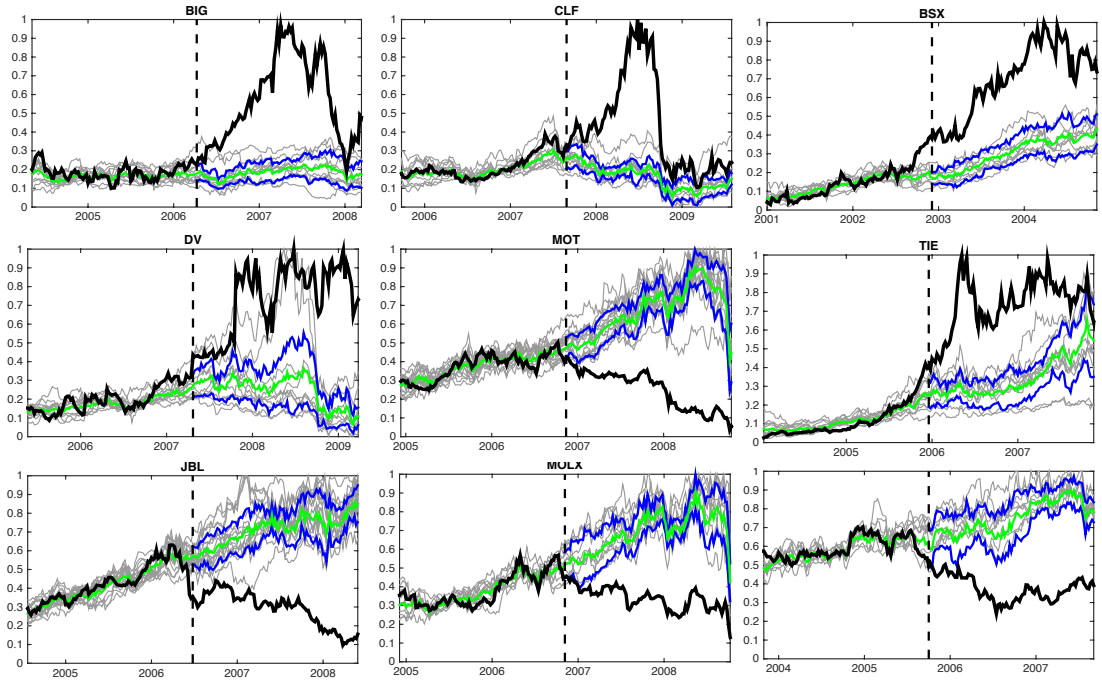


Figure 3.6: The top 9 S-CTCs detected from the S&P 500 dataset. In each plot, we use a black curve for the target time series, gray curves for time series in the DPG, a black dotted line for the detected change point, blue lines for the 16 & 84 percentile of the DPG during the scoring period, and a green line for the mean of the DGP.

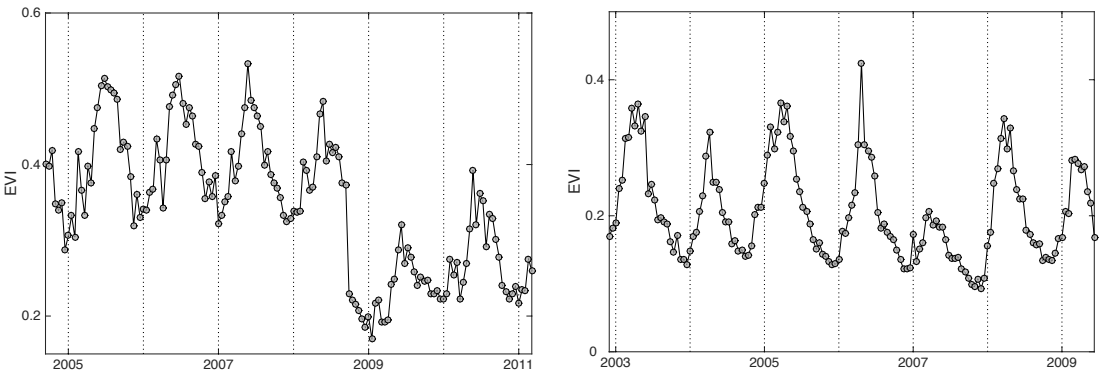
3.5.2 Fire detection using remote sensing data

In this experiment, we compare the proposed method and the V2Delta method in their performance on forest fire detection. The V2Delta method is a traditional change-point detection method and it is designed to detect forest fires [67]. This experiment shows that the proposed S-CTC detection method is capable of identifying sub-area events

(*i.e.*, events that only impact a relatively small area, such as forest fires), while the V2Delta method detects all types of changes (which includes large scale events such as droughts). Hence, the proposed method can help to distinguish fires from droughts.

Fire detection from remote sensing data : V2Delta vs. S-CTC detection

We detect wild fires as change points in the Enhanced Vegetation Index (EVI) time series. A EVI time series is an indicator of "greenness" reflected from the Earth's surface. In most cases, wild fires can lead to abrupt changes in EVI time series. Figure 3.7 (a) shows the EVI time series of a location that burned in 2008. The V2Delta algorithm is designed to capture drops in the annual mean from a periodic time series and hence works well in fire detection. However, droughts can also cause decreases in EVI signals. Figure 3.7 (b) shows a EVI time series from a drought area. We can observe a significant decrease in the annual mean of the EVI time series in 2007 when the drought occurred. Therefore, the V2Delta algorithm does not perform well in forest fire detection when droughts may also occur. Many attempts have been made to increase the accuracy of fire detection in drought areas using traditional change-point detection methods. However, to the extent of our knowledge, there is no definitive set of EVI features that has been discovered to distinguish fires from droughts.



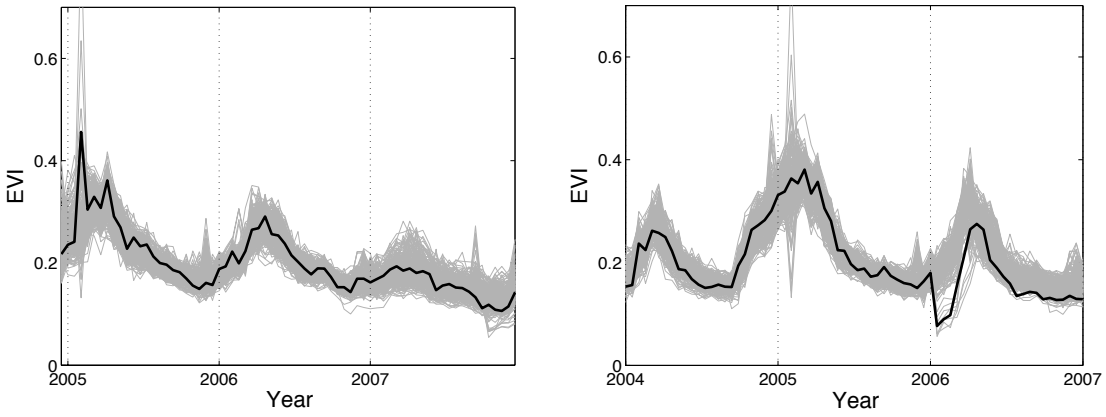
(a) The area was burned in 2008.

(b) A drought occurred in 2007.

Figure 3.7: Two EVI time series. The left one is from a burned area, and the right one is from a drought area.

The proposed S-CTC detection method provides a solution to this problem. In

general, when a drought occurs, many locations from the same land cover type exhibit the same level of drops in their greenness. Hence, there are no S-CTCs in the group of EVI time series. Figure 3.8 (a) shows a time series from a drought area. The drought occurred in 2007. We observe that the target time series (black) behaves similarly to its DPG (gray) in the drought year. In contrast, fires typically affect limited regions and hence can be detected as S-CTCs in EVI time series. Figure 3.8 (b) shows a time series from a burned location. The fire happened in 2006. We observe that the target time series (black) began to deviate from its DPG (gray) when the location burned.



(a) Drought in 2007

(b) Fire in 2006

Figure 3.8: Prototypical EVI signals for drought and forest fire events.

Experimental setup

To demonstrate the capability of the proposed method for fire detection in the context of droughts, we focus on an area that suffered a level D3 drought in 2007 [49]. Almost all vegetation in that region lost a certain degree of greenness in that year. The experimental region is in the state of California and bounded by 33.9°N, 33.45°N, 117.75°W and 117.05°W.

The validation data is the fire perimeter polygons [1] from government agencies that are responsible for monitoring and managing forests and wildfires. According to these validation data, we assign a label for each time series: 0 for unburned, 1 for burned, and 2 for partially burned. We consider a time series to be *positive* if it is totally

burned (label is 1) and negative if it is totally unburned (label is 0). The remaining pixels (partially burned) are discarded from our evaluation to avoid ambiguity. Since wild fires are rare events, we use precision and recall [84] as evaluation metrics in this experiment. The parameters selected in this experiment are: $w = 46$, $k = 1$, and $l = 6$.

Experimental results

Figure 3.9 (best viewed in color) shows the precision-recall curves of the proposed S-CTC detection algorithm (red lines) and the V2Delta method (blue lines). From this result, we observe a clear improvement in both precision and recall values when using the proposed method in fire detection.

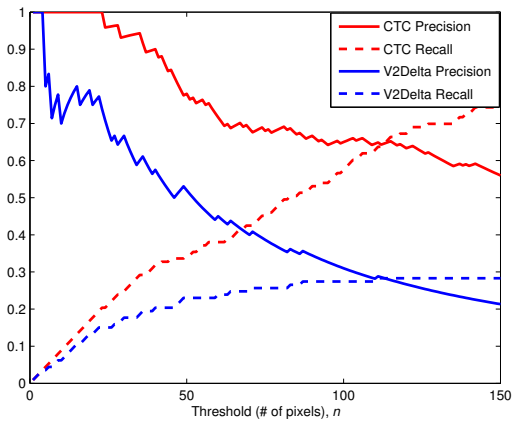
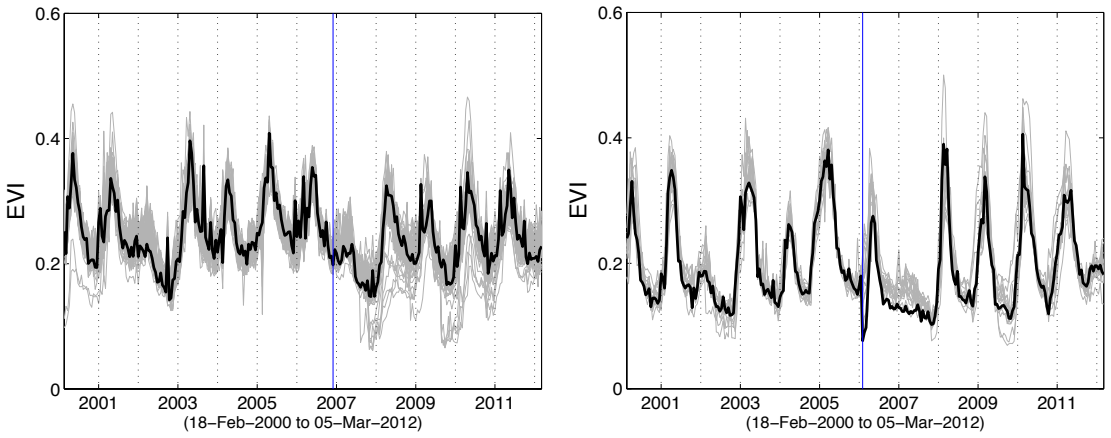


Figure 3.9: Precision and recall curves of the proposed S-CTC detection algorithm and the V2Delta method in forest fire detection using remote sensing data.

We use two examples in Figure 3.10 to demonstrate the advantages of the proposed method. Figure 3.10 (a) shows a time series (black) from an unburned location. This location suffered a drought in 2007. We refer to the black time series as the target time series in this example. The target time series is mistakenly detected by the V2Delta method as a burned location but is correctly reported as a normal location by the proposed method. We can observe that time steps after the detected V2Delta change point (the blue vertical line) have lower values than the time steps before the change point. Therefore, the detected V2Delta change point is a traditional change point. However, all time series in the DPG of the target time series (gray lines) share a similar

drop as the target time series in 2007. Hence, the target time series does not contain a S-CTC. This example shows that S-CTCs can help to distinguish fires from droughts, while traditional methods detect both fires and droughts as change points.

Figure 3.10 (b) shows a burned location that is not detected by V2Delta. The proposed method successfully labels it as a positive. We can observe a significant yearly variance from the target time series (the black curve). For example, the time series has low observations in 2002 and 2004 but has high observations in 2001 and 2003. As a result, the magnitude of the drop when fire occurred (in 2006) is not large enough to be detected as a change point by V2Delta. In contrast, the target time series always behaves similarly to its DPG (gray curves) before the blue vertical line even in 2002 and 2004. After the blue vertical line, the target time series suddenly starts to deviate from its DPG. The proposed method detects a S-CTC at the blue vertical line. In fact, the detected S-CTC matches with the time when this location burned.



(a) A FP of V2Delta but a TN of the proposed method. The blue line is the change point detected by V2Delta.

(b) A FN of V2Delta but a TP of the proposed method. The blue line is the change point detected by the proposed method.

Figure 3.10: Two examples that demonstrate the advantages of the proposed S-CTC detection method for fires detection in the context of droughts.

Limitations of the proposed method

There are two major types of errors when using the proposed method in fire detection. First, the TAD score (Definition 3.5) uses the distance between the m_1 and m_2 percentiles as the denominator. When most time series in the DPG are very similar to each other, the denominator can be very small. As a result, the TAD score may be high even if the target time series is slightly deviate from its DPG. Figure 3.11 shows such an example.

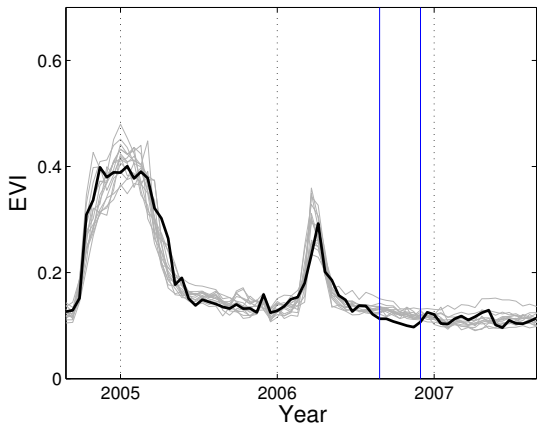
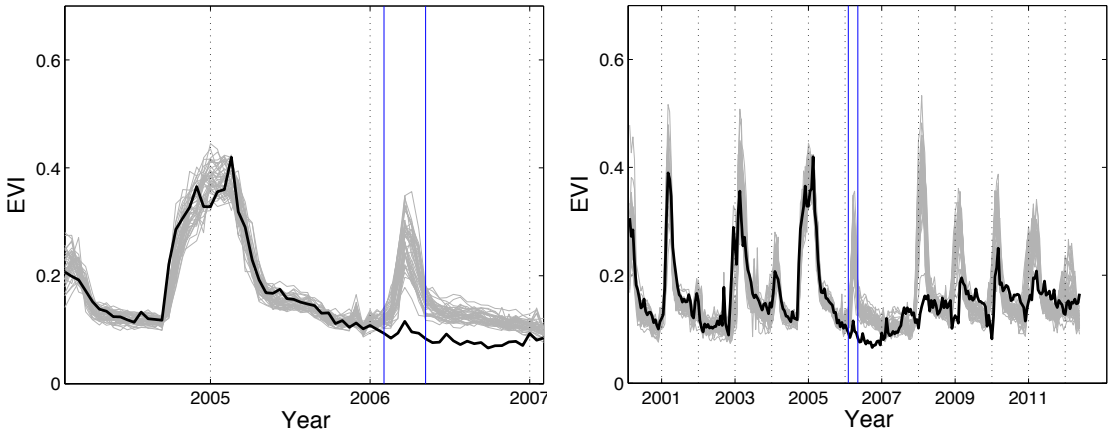


Figure 3.11: A false positive of the S-CTC detection method in detecting forest fires due to the small denominator in the TAD function. The scoring period is the area between the two blue vertical lines.

Second, S-CTCs help in distinguishing fires from droughts. However, using S-CTCs cannot exclude other types of events (*e.g.*, conversions in the land cover type). Figure 3.12 (a) shows a detected S-CTC (the left blue line). The target time series (black) is not from a burned location. We observe a clear separation between the target time series and its DPG (gray) after the change point. Figure 3.12 (b) is the entire ten years data of the same target time series with its DPG. We observe that the time series pattern before and after 2006 are very different and conclude that there is a land cover type conversion that occurred in this location around the detected S-CTC.



(a) (b)

Figure 3.12: The proposed S-CTC detection method may also detect other types of events in addition to forest fires. The given time series contains a S-CTC that indicates a land cover type conversion instead of a forest fire. The scoring period is the area between the two blue vertical lines.

3.6 Conclusion

In this chapter, we present a novel type of change points, Singleton Contextual Time-series change points (S-CTCs), and also propose a method to detect S-CTCs from a time-series dataset. This work has four innovations. First, we formally define S-CTCs. Second, we use a new distance function, k^{th} order statistic distance, to measure the difference between two sub-sequences. This function is very robust to outliers. Third, we introduce Time-series Area Depth (TAD) function to score S-CTCs. Finally, the proposed scoring mechanism assumes that only a very small number of time series in the DPG change contextually. We provide a heuristic algorithm (*i.e.*, *Multimode Remover*) to detect S-CTCs when this assumption is not met (*e.g.*, 30% of time series in the DPG changes contextually). The experimental results show that the proposed method finds new types of events that cannot be discovered by traditional change-point detection methods (*e.g.*, CUSUM and V2Delta).

There are several open challenges that remain in detecting contextual change points. First, we define the DPG of a target time series using k^{th} order statistic distance. This distance function is designed from Minkowski distances. In some applications, other

time-series distance functions (*e.g.*, Pearson's correlation and dynamic time warping [6]) may be more suitable. Hence, there is a need to accommodate other measures into the S-CTC detection algorithm. Second, the proposed method contains several user-defined parameters. The performance of the proposed method depends on the selected parameters. Thus, we need to work on developing an automatic algorithm to discover the best parameters. Finally, S-CTCs are one type of contextual change points. There are many other ways to define the context of time series and the type of change points. One of the most interesting research directions from our opinion is to define other useful contextual change points and design new algorithms to discover them automatically.

Chapter 4

G-CTC detection:

Group level contextual time-series change-point detection

4.1 Introduction

A Contextual Time-series Change-point (CTC) detection method aims to detect the time when the relationship among several time series changes. In our earlier work (Chapter 3), we propose a method to detect one type of CTCs, Singleton Contextual Time-series Change points (S-CTCs). S-CTCs are the times when *one* target time series starts to deviate from its own context (*i.e.*, a group of time series that behaves similarly to the target time series). In this chapter, we present another type of CTCs, *Group level Contextual Time-series Change points* (G-CTCs). As opposed to S-CTCs that are defined by the behavior of a single target time series with respect to a group of related time series, G-CTCs are the times when a whole group of time series begins to behave similarly or starts to behave differently.

More precisely, G-CTCs include two types of events: group formation and group disbanding. Group formation is the formation of time-series clusters. Here, we refer to a group of time series that behaves similarly during a certain time period as a time-series cluster in that period. The bottom row of Figure 4.1 shows two examples of

group formation events. In the bottom right plot, the time-series cluster is formed from several totally unrelated time series. In the bottom left plot, two small groups of time series merge into one group. Group disbanding events, the other type of G-CTCs, are the dissolution of a time-series cluster. The top row of Figure 4.1 shows two group disbanding events. As shown in the plots, when a disbanding event occurs, time series in the old cluster may disband into two or more smaller clusters (the top left plot) or dissipate into totally uncorrelated individual components (the top right plot).

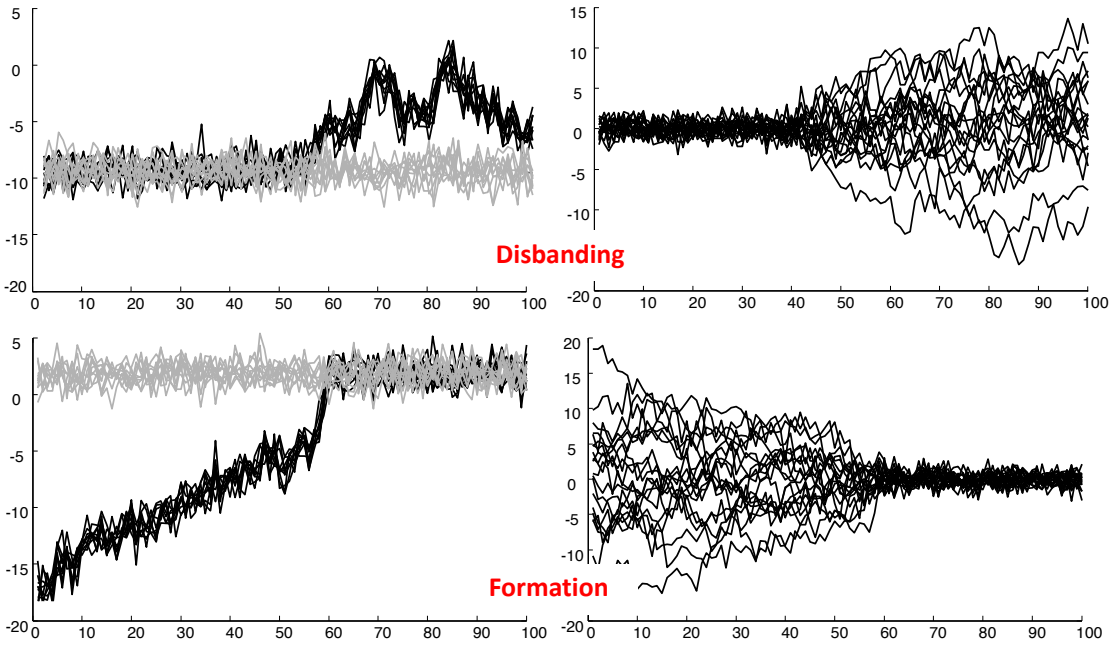
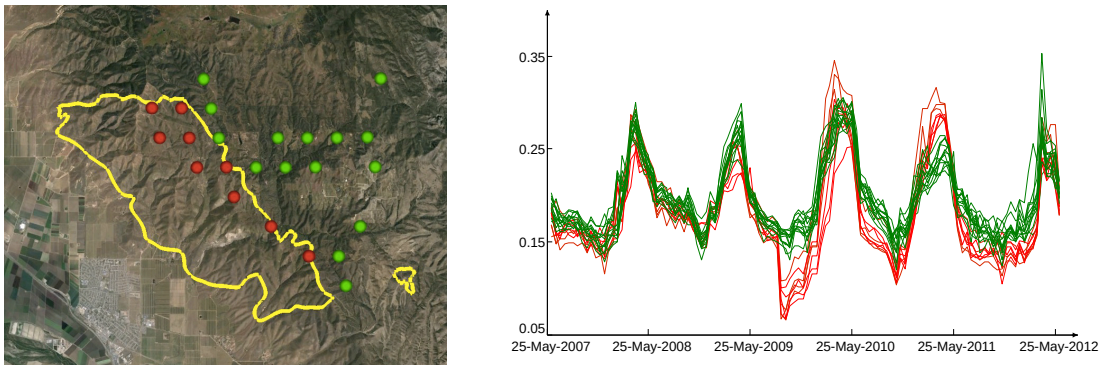


Figure 4.1: G-CTCs include two types of events: group disbanding (in the top row) and group formation (in the bottom row).

G-CTCs are intuitive and reliable patterns of many important events in real-world applications. For example, we are able to infer when (the time) and where (the geographical locations) forest fires occurred by discovering G-CTCs in "greenness" time series. Below we use an example to demonstrate this idea. The greenness of a location is determined by many factors, such as the type of vegetation, temperature, and soil-moisture. The time series of greenness (*i.e.*, Enhanced Vegetation Index or EVI) at geographically proximal locations show highly coherent behavior if they are covered by the same type of vegetation. Figure 4.2 shows a group of such locations. Figure 4.2 (a)

shows their positions in a satellite image and Figure 4.2 (b) shows the corresponding EVI time series. We can observe from Figure 4.2 (b) that all EVI time series are very similar to each other during normal years (year 2007 and year 2008). We know that locations marked by red dots were burned because of a natural forest fire in Aug. 2009, while the locations marked by green dots were not burned. Since not all locations in this area were burned, we can observe a group disbanding event at the time when the forest fire occurred.



(a) (b)

Figure 4.2: A set of EVI time series which disbands in August 2009 because of a forest fire. The yellow curve in (a) is the boundary of the burned area. Detecting such G-CTCs helps in discovering forest fires from a remote sensing dataset. Red and green curves in (b) are the time series of points inside (marked as the red dots) and outside (marked as the green dots) the fire-affected region, respectively.

In this work, we propose an efficient algorithm to detect G-CTCs. The proposed method contains three innovations. First, we design a new time-series clustering method, AutoDBSCAN, to discover time-series clusters with different densities. Second, we introduce the indexing technique and the iterative technique to reduce the computational cost of the DBSCAN method, which is the most time consuming component in the AutoDBSCAN algorithm. Finally, we assign a change score for each discovered time-series cluster using the similarity-aware entropy function. This novel function measures the within-cluster distance of a time-series group. The experimental results show that G-CTCs indicate novel types of events, and the scalable clustering algorithm is more efficient than the original algorithm.

4.2 Notations and problem formulation

In this section, we first introduce the mathematical notations and then define G-CTCs.

Notations. *We follow the MATLAB style and use the mathematical notations below.*

- *Boldface upper case letters (e.g., $\boldsymbol{X}$ and $\boldsymbol{Y}$) stand for time-series datasets or groups of time series. A time-series dataset or a group of time series is a matrix. Each row in a matrix is a time series. $\boldsymbol{X}(i,:)$ denotes the i^{th} time series in dataset/time-series group $\boldsymbol{X}$. Each column in a matrix includes all the observations at a single time step. $\boldsymbol{X}(:,t)$ denotes the values at the t^{th} time step of all time series in $\boldsymbol{X}$. $\boldsymbol{X}(:,i:j)$ denotes the values from the i^{th} time step to the j^{th} time step of all time series in $\boldsymbol{X}$.*

- *Boldface lower case letters (e.g., $\boldsymbol{x}$ and $\boldsymbol{y}$) stand for time series. A time series is a vector. $\boldsymbol{x}(t)$ denotes the observation at time step t of time series $\boldsymbol{x}$. $\boldsymbol{x}(i:j)$ denotes observations of time series $\boldsymbol{x}$ from time step i to time step j.*

- *Non-boldface lower case letters (e.g., x_t and y_t) stand for values at a single time step. x_t is the observation at time step t of time series $\boldsymbol{x}$.*

G-CTCs are the time when a new time-series cluster forms or an old time-series cluster dissolves. Definition 4.1 is the formal definition of G-CTCs.

Definition 4.1. (Group Level Contextual Time-series Change points, G-CTCs) *Let $\boldsymbol{X}$ be a group of time series. Time step t is a G-CTC of $\boldsymbol{X}$ if and and if both of the following statements are true.*

- *$\boldsymbol{X}(:,t-w:t-1)$ forms a time-series cluster or $\boldsymbol{X}(:,t+1:t+w)$ forms a time-series cluster.*

- *The within-group distance of $\boldsymbol{X}$ changes dramatically, i.e.,*

$$\left| log\big(d_w\left(\boldsymbol{X}(:,t-w:t-1)\right)\big) - log\big(d_w\left(\boldsymbol{X}(:,t+1:t+w)\right)\big) \right| > \alpha$$

  where w is a user-defined parameter that controls the window length for time-series clustering and G-CTC scoring.

We name the event group disbanding *when $\boldsymbol{X}(:,t-w:t-1)$ forms a cluster. We call the event* group formation *when $\boldsymbol{X}(:,t+1:t+w)$ forms a cluster.*

4.3 Related work

Most existing change-point detection methods detect the time when a time series begins to be significantly different from its own historical data [5, 9, 21, 45–47, 54, 78, 83, 86]. Typically, these methods monitor some predefined time-series features (*e.g.*, mean, variance and other statistics of the given time series) and mark time points as change points when the selected time-series features change dramatically. For example, CUSUM chart [72] considers the time when the mean of a time series shifts as a change point. BIFAST fits a stochastic model for each time-series data and reports a change point when the parameters of the fitted model change a lot.

Recently, Chen et al. [25] proposed a new type of change points named (Singleton) Contextual Time-series Change points (S-CTCs). As opposed to the above mentioned traditional change points that are defined by the auto-correlation information of a time series, S-CTCs refer to the times when time series begin to deviate from their own contexts. In [25], the context of a time series is a group of time series that behaves similarly to the target time series. Other types of contexts such as event log [44] and geo-location may be used as well.

In this work, we aim to detect Group level Contextual Time-series Change points (G-CTCs). A G-CTC (as shown in Definition 4.1) is the time when a new time-series cluster forms or an old time-series cluster dissolves. Since G-CTCs are a novel type of change points, there is no literature on detecting G-CTCs. There is prior research on discovering time-series clusters (*e.g.*, [13, 59, 76]). However, most of these methods focus on discovering static time-series clusters and pay no attention to the time when the cluster structure of a time-series dataset changes.

4.4 Proposed method

We propose an efficient method to detect G-CTCs. Algorithm 4.1 shows its pseudo-code. This algorithm inputs a time-series dataset $\boldsymbol{X}$ and several user-defined parameters $\boldsymbol{\Theta}$. It outputs an event set for every time step. An event set $\mathcal{E}_t$ contains the time-series clusters that change contextually at time step t.

The proposed method is a two-step approach. For any time step, the method first groups time series into several clusters using the AutoDBSCAN algorithm (Line 2),

and then provides a G-CTC score for each time-series cluster using the similarity-aware entropy scoring function (Line 4). A G-CTC is reported when the similarity-aware entropy score of a time-series cluster is above a user-defined threshold α (Line 5 - Line 7). Next, we present the two major components in the proposed method, the AutoDBSCAN algorithm and the similarity-aware entropy scoring function, in detail.

Algorithm 4.1: The proposed G-CTC detection framework

Input: $\boldsymbol{X}$: a time-series dataset. Each time series contains T time steps;
$\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \alpha\}$: user defined parameters;
Output: $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_T\}$: the event sets;

```
1 for t from 1 to T do
2     {C_1, C_2, ..., C_m} = AutoDBSCAN (X, t, Θ_1) ;
3     for i from 1 to m do
4         s_i = simEntropy(C_i, t, Θ_2);
5         if s_i > α then
6             Put C_i into E_t (i.e., the event set of time-step t);
7         end
8     end
9 end
```

4.4.1 Grouping time series into clusters: *AutoDBSCAN(.)*

We propose two time-series clustering methods, MultiDBSCAN and AutoDBSCAN. These two methods provide almost identical clustering results. MultiDBSCAN is the most intuitive method to find time-series clusters with different densities. AutoDBSCAN is a scalable version of MultiDBSCAN. In our proposed method, we use AutoDBSCAN because its computational cost is less than MultiDBSCAN.

The DBSCAN function is the most time-consuming component in AutoDBSCAN. To further speed up the proposed method, we present two techniques (*i.e.*, the iterative technique and the indexing technique) to reduce the computational cost of the DBSCAN algorithm. The proposed techniques can work with a variety of time-series similarity functions (*e.g.*, Pearson's correlation and Euclidean distance).

Here, we first introduce the MultiDBSCAN algorithm and also provide the reasons

for using MultiDBSCAN in G-CTC detection. Then, we present the AutoDBCAN algorithm and the two techniques that can speed up the DBSCAN algorithm.

MultiDBSCAN

The G-CTC detection algorithm has two requirements for the time-series clustering method. First, the clustering algorithm can form an "unclustered" set. The unclustered set includes time series that are not similar to any other time series. Such an unclustered set is important to many real-world applications where not all time series belong to a cluster. Second, the algorithm can discover time-series clusters with different densities. G-CTCs are the times when time-series clusters form or dissolve. In many cases, it is very hard to define the density of an interesting time-series cluster. Furthermore, G-CTCs may occur simultaneously in multiple time-series groups with different densities. As a result, constructing time-series clusters in different density levels is an important property of the selected clustering method.

Algorithm 4.2 is the pseudocode of the proposed MultiDBSCAN. Simply put, MultiDBSCAN finds time-series clusters with different densities by running the DBSCAN algorithm multiple times with different neighborhood size parameters (*i.e.*, ϵ). In detail, the selected ϵ are all values in the set $\{\epsilon_1, \epsilon_1 + \delta, \epsilon_1 + 2\delta, \cdots, \epsilon_2\}$.

Algorithm 4.2: *MultiDBSCAN*

Input: $\boldsymbol{X}$: a time-series dataset;
t: the current time step;
$\boldsymbol{\Theta}_1 = \{\epsilon_1, \epsilon_2, minPts, \delta\}$: user-defined parameters;

Output: $\mathcal{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m\}$: time-series clusters;

```
ε = ε_1;
for any ε ∈ {ε_1, ε_1 + δ, ε_1 + 2δ, ···, ε_2} do
    {D_1, ···, D_k} = DBSCAN(X(:, t − w : t − 1), ε, minPts);
    C = C ⋃ {D_1, ···, D_k} ;          // put {D_1, ···, D_k} into C
end
```

The proposed MultiDBSCAN algorithm is a naive approach that meets the two requirements of a G-CTC detection method. First, MultiDBSCAN is constructed from DBSCAN [34] and hence time series that do not belong to any cluster are labeled as

noise. Second, the DBSCAN algorithm discovers clusters under a certain density setting. The cluster density is controlled by two parameters: the neighborhood size (ϵ), and the minimum number of points within the $\epsilon-$neighborhood ($minPts$). The MultiDBSCAN method fixs $minPts$ and runs the DBSCAN algorithm multiple times with different ϵ values. Therefore, it can find time-series clusters with different densities.

AutoDBSCAN

AutoDBSCAN provides almost equivalent clustering results as MultiDBSCAN, but AutoDBSCAN is computationally more efficient. Algorithm 4.3 is the pseudocode of AutoDBSCAN. This algorithm has three types of inputs: (i) a time-series dataset $\boldsymbol{X}$, (ii) the current time step t, and (iii) several user-defined parameters, where ϵ_1, ϵ_2, and δ determine the selected neighborhood sizes, $minPts$ is the minimum number of points within the $\epsilon-$neighborhood, and w controls the length of the clustering window. AutoDBSCAN outputs several time-series groups. Each of them is a time-series cluster within time period $[t-w, t-1]$.

Algorithm 4.3: *AutoDBSCAN*

Input: $\boldsymbol{X}$: a time-series dataset;
t: the current time-step;
$\{\epsilon_1, \epsilon_2, minPts, \delta, w\}$: user-defined parameters;
Output: $\mathcal{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m\}$: time-series clusters;

```
1 {D_1, ..., D_k} = DBSCAN(X(:, t-w : t-1), ε_2, minPts);
2 put {D_1, ..., D_k} into C;
3 for i from 1 to k do
4 |   C' = AutoDBSCAN(D_i, t, ε_1, ε_2 - δ, minPts, δ);
5 |   if number of clusters in C' = 0 then
6 |   |   break;
7 |   end
8 |   put all time-series clusters in C' into C;
9 end
```

AutoDBSCAN is a recursive algorithm. It starts by detecting clusters using DBSCAN with ϵ_2, the maximum value of all the selected neighborhood sizes (Line 2).

Then, the algorithm recursively discovers higher density sub-clusters within each detected cluster. In each iteration, this recursive algorithm reduces the neighborhood size by δ (Line 4). The recursive process breaks when no more higher density clusters are detected (Line 5 - Line 7).

AutoDBSCAN has identical clustering results to MultiDBSCAN, but its computational cost is much lower. The containment property (Lemma 4.1) demonstrates the equivalence of AutoDBSCAN and MultiDBSCAN.

Lemma 4.1. *(Containment property) Let* $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_m$ *and* $\boldsymbol{B}_1, \boldsymbol{B}_2, \cdots, \boldsymbol{B}_n$ *be clusters discovered by DBSCAN using neighborhood size* ϵ_a *and* ϵ_b*, respectively (with* $\epsilon_a < \epsilon_b$*). Then for any* $i \in [1, \cdots, m]$*, there is a* $j \in [1, \cdots, n]$ *such that* $\boldsymbol{A}_i \subset \boldsymbol{B}_j$*.*

Lemma 4.1 means that any cluster detected using a smaller ϵ is a subset of a cluster detected using a larger ϵ. Therefore, after finding all clusters with a lower density (*i.e.*, $\mathcal{B} = \{\boldsymbol{B}_1, \boldsymbol{B}_2, \cdots, \boldsymbol{B}_n\}$), clusters with a higher density (*i.e.*, $\mathcal{A} = \{\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_m\}$) can be obtained by searching within each cluster in $\mathcal{B}$. Hence, MultiDBSCAN and AutoDBSCAN provide the same outputs.

The computational cost of DBSCAN is $O(n^2 w)$ where n is the number of time series and w is the window size used to cluster time series. MultiDBSCAN executes DBSCAN for $J = \lfloor \frac{\epsilon_2 - \epsilon_1}{\delta} \rfloor$ times and its cost is $O(n^2 w J)$. AutoDBSCAN reduces the computational cost by discovering clusters within a smaller group of data. For example, the computational cost at the second iteration in AutoDBSCAN (*i.e.*, for $\epsilon_2 - \delta$) is $O(\sum_{i=1}^{k} n_i^2 \times w)$, where k is the number of clusters detected in the first iteration and n_i is the number of time series in the ith cluster. We know that $\sum_{i=1}^{k} n_i^2 < (\sum_{i=1}^{k} n_i)^2$. In some applications, $\sum_{i=1}^{k} n_i << n$ due to the large number of "unclustered" time series. Hence, the computational cost of AutoDBSCAN is much lower than MultiDBSCAN.

Scalable implementation of DBSCAN for sub-sequences clustering

A straightforward implementation of AutoDBSCAN in C++ language took more than 20 hours to process a patch of 5,000 EVI time series over a mere 200 time steps (at this geographical resolution, the entire earth generates over 150 million EVI time series). This highlights the importance of speeding up the AutoDBSCAN algorithm.

In AutoDBSCAN, DBSCAN is the most expensive function. Algorithm 4.4 is the

pseudocode of DBSCAN [34]. The computational cost of DBSCAN is dominated by the function $regionQuery(.)$, which returns the $\epsilon-$neighbors of each time series. We propose to reduce the computational cost of AutoDBSCAN by speeding up the region query function using (i) the indexing technique and (ii) the iterative technique.

Algorithm 4.4: *DBSCAN* [34]

Input: $\boldsymbol{X}$: the input dataset;
$\epsilon, MinPts$: user-defined parameters;

Output: $\mathcal{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m\}$: the sets of time-series clustres;

```
k = 0;
for each point x in X do
    if x is not visited then
        mark x as visited;
        NeighborPts = regionQuery(x, ε);
        if sizeof(NeighborPts) > minPts then
            k = k + 1;
            add x into cluster C_k;
            for each point g in NeighborPts do
                if g is not visited then
                    mark g as visited;
                    NeighborPts' = regionQuery(g, ε);
                    if sizeof(NeighborPts) ≥ minPts then
                        NeighborPts = NeighborPts joined with NeighborPts';
                    end
                    if g is not yet a member of any cluster then
                        add g in cluster C_k;
                    end
                end
            end
        end
    end
end
```

The indexing technique We name the region query function that utilizes the indexing technique *the indexing region query function*. The proposed indexing region query function first assigns an index to each data and reorders the dataset according to the index. And then, it returns the $\epsilon-$neighbors of each data point using the reordered dataset.

We use Algorithm 4.5 is to create the reordered dataset. In detail, we first randomly select one time series from the dataset (Line 1). We name this time series the reference time series. Then, we assign an index to each data in the dataset as its distance to the reference time series (Line 2 - Line 4). Finally, we rank all data points in the dataset in ascending order of the distance values (Line 5). We name the output of this step *the reordered dataset*. The computational cost of this reordering step is $O(nw)+O(n\log n)$.

Algorithm 4.5: The reordering step in the indexing region query function

Input: $\boldsymbol{X}$: the original dataset that contains n time series;

Output: $\boldsymbol{X'}$: the reordered dataset;

1 $\boldsymbol{r}$ = a random selected time series data from $\boldsymbol{X}$;
2 **for** *i from 1 to n* **do**
3 $\quad \beta_i = d\big(\boldsymbol{X}(i,:), \boldsymbol{r}\big)$;
4 **end**
5 $\boldsymbol{X'}$ = sort $\boldsymbol{X}$ in ascending order of $\boldsymbol{\beta}$;

The reordered dataset contains the following properties. Here, we use $\boldsymbol{r}$ for the reference time series, $\boldsymbol{X}$ for the input dataset, and $\boldsymbol{X'}$ for the reordered dataset.

1. For any i and j, if $i > j$, then
$$d\big(\boldsymbol{X'}(i,:), \boldsymbol{r}\big) > d\big(\boldsymbol{X'}(j,:), \boldsymbol{r}\big)$$
2. For any i, j, and k, if $i > j > k$ (or $i < j < k$) and
$$d\big(\boldsymbol{X'}(i,:), \boldsymbol{r}\big) - d\big(\boldsymbol{X'}(j,:), \boldsymbol{r}\big) > \epsilon$$
then,
$$d\big(\boldsymbol{X'}(i,:), \boldsymbol{X'}(k,:)\big) > \epsilon$$

The above two properties of the reordered data can help to reduce the computational cost of the region query function. Assume that we search for the $\epsilon-$neighbors of time

series $\boldsymbol{X'}(i,:)$. According to the properties, we can shrink our searching range from the entire dataset to a subset, $\boldsymbol{X'}(a:b,:)$, where

$$d\big(\boldsymbol{X'}(i,:),\boldsymbol{r}\big) - d\big(\boldsymbol{X'}(a,:),\boldsymbol{r}\big) < \epsilon \quad \text{and} \quad d\big(\boldsymbol{X'}(i,:),\boldsymbol{r}\big) - d\big(\boldsymbol{X'}(a-1,:),\boldsymbol{r}\big) > \epsilon$$

and

$$d\big(\boldsymbol{X'}(i,:),\boldsymbol{r}\big) - d\big(\boldsymbol{X'}(b,:),\boldsymbol{r}\big) < \epsilon \quad \text{and} \quad d\big(\boldsymbol{X'}(i,:),\boldsymbol{r}\big) - d\big(\boldsymbol{X'}(b+1,:),\boldsymbol{r}\big) > \epsilon$$

Algorithm 4.6: The querying step in the indexing region query function

Input: $\boldsymbol{X'}$: the input time-series data;
$\boldsymbol{X'}(i,:)$: the target time series;
$\boldsymbol{r}$: the reference time series;
ϵ: a user-defined parameter;

Output: $NeighborPts$

```
j = 1;
while d(X'(i,:), r) − d(X'(i − j,:), r) < ε do
    if d(X'(i,:), X'(i − j,:)) < ε then
        Add X'(i − j,:) into NeighborPts;
    end
    j = j + 1;
end
j = 1;
while d(X'(i,:), r) − d(X'(i + j,:), r) < ε do
    if d(X'(i,:), X'(i + j,:)) < ε then
        Add X'(i + j,:) into NeighborPts;
    end
    j = j+1;
end
```

Algorithm 4.6 is the pseudocode of this step. In detail, we find the $\epsilon-$neighbors of the i^{th} time series in the reordered dataset as below. We test against the neighbors of $\boldsymbol{X'}(i,:)$ in the reordered dataset with ranks $i-1, i-2, \ldots,$ until $d\big(\boldsymbol{X'}(i,:),\boldsymbol{r}\big) - d\big(\boldsymbol{X'}(i-j,:),\boldsymbol{r}\big) > \epsilon$ is satisfied. Similarly, we test against the neighbors of $\boldsymbol{X'}(i,:)$ with ranks $i+1, i+2, \ldots$ until $d\big(\boldsymbol{X'}(i,:),\boldsymbol{r}\big) - d\big(\boldsymbol{X'}(i+j,:),\boldsymbol{r}\big) > \epsilon$ is satisfied.

The iterative technique In most region query functions, including Algorithm 4.6, the computational cost of the distance calculation (between two data points) is $O(w)$, where w is the dimensionality of data points. In our case, w is the window length used in discovering the time-series clusters. The proposed iterative technique can reduce the computational cost of the distance calculation from $O(w)$ to $O(1)$. Below, we demonstrate this iterative technique in both Pearson's correlation and Euclidean distance.

Let us define the following notations.

$$\mu_x(t) = \sum_{i=t}^{t+w} x_i \qquad \mu_y(t) = \sum_{i=t}^{t+w} y_i \qquad S_x(t) = \sum_{i=t}^{t+w} x_i^2$$

$$S_y(t) = \sum_{i=t}^{t+w} y_i^2 \qquad P_{x,y}(t) = \sum_{i=t}^{t+w} x_i y_i$$

Then, the Pearson's correlation between two subsequences $\boldsymbol{x}(t : t+w)$ and $\boldsymbol{y}(t : t+w)$ can be calculated using the following formula.

$$corr(\boldsymbol{x}(t : t+w), \boldsymbol{y}(t : t+w)) = \frac{P_{x,y}(t) - \mu_x(t)\mu_y(t)/w}{\sqrt{S_x(t) - \mu_x(t)^2/w}\sqrt{S_y(t) - \mu_y(t)^2/w}}$$

Similarly, the Euclidean distance between $\boldsymbol{x}(t : t+w)$ and $\boldsymbol{y}(t : t+w)$ can be calculated as below.

$$d^2(\boldsymbol{x}(t : t+w), \boldsymbol{y}(t : t+w)) = S_x(t) + S_y(t) - 2P_{x,y}(t)$$

To compute the distance between $\boldsymbol{x}(t : t+w)$ and $\boldsymbol{y}(t : t+w)$ more efficiently, we first update $\mu_x(t)$, $\mu_y(t)$, $S_x(t)$, $S_y(t)$, and $P_{x,y}(t)$ using values $\mu_x(t-1)$, $\mu_y(t-1)$, $S_x(t-1)$, $S_y(t-1)$, and $P_{x,y}(t-1)$. Then, we calculate the similarity between two subsequences according to the selected distance function.

4.4.2 Scoring events using similarity-aware entropy: *SimEntropy(.)*

The scoring step is to assign an event score to each time-series cluster discovered by AutoDBSCAN. As shown in Definition 4.1, the event score of an cluster is the change in the within-group distances of the cluster before and after the change point. We design

the G-CTC scoring function as below (same as Definition 4.1) due to its equal ability in discovering group-disbanding events and group-formation events.

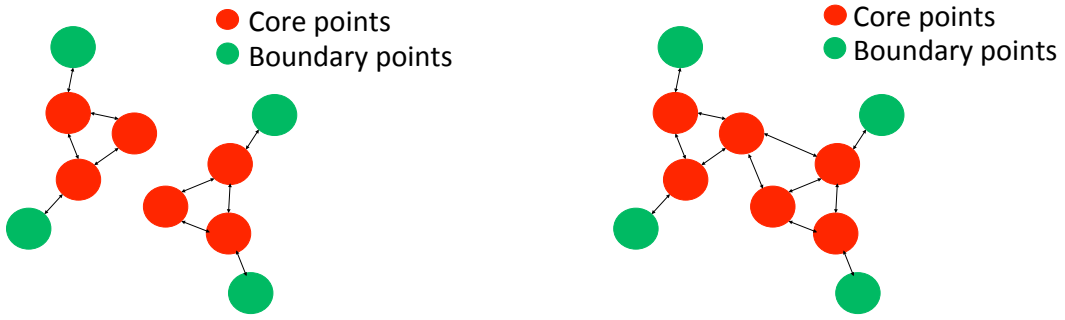
$$S(t) = \left|\log\big(d_w\left(\boldsymbol{X}(:, t-w:t-1)\right)\big) - \log\big(d_w\left(\boldsymbol{X}(:, t+1:t+w)\right)\big)\right|$$

We propose to use *the similarity-aware entropy function* to measure the within-group distance of a time-series group. The novel function is generalized from the traditional entropy function. Next, we first introduce the traditional entropy distance function and discuss its limitation. Then, we describe our proposed similarity-aware entropy function in detail.

Intuitively, the within-group distance of a time-series group should be low if those time series form a cluster, and the distance value should be high if time series in the group are different from each other (or do not cluster together). Entropy is a candidate of the within-group distance function. Let $\boldsymbol{X}$ be a time-series group that contains k time-series cluster, $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k$, during the time-period $[t, t+1, \cdots, t+w]$, n_i be the number of time series in $\boldsymbol{X}_i$, and and $n = \sum_{i=1}^{k} n_i$. Then, the entropy of $\boldsymbol{X}$ during $[t, t+1, \cdots, t+w]$ is as below.

$$H(\boldsymbol{X}|\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k) = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Although the above entropy function matches the intuition of our within-group distance function, there are two major limitations when using this entropy function in G-CTC detection. First of all, to calculate the entropy of a time-series group, we need to know the cluster membership (or cluster ID) of each time series during the given time period. Although we can estimate the cluster membership using a clustering algorithm, the estimated entropy will be sensitive to the selection of the clustering method and the corresponding parameters. We use an example to demonstrate this problem. Figure 4.3 shows two clustering results of DBSCAN on the same dataset with different parameters. In Figure 4.3 (a), we use a smaller ϵ value. DBSCAN discovers two equal sized clusters. In Figure 4.3 (b), we use a larger ϵ value. DBSCAN clusters all 10 points into the same group. As a result, the within-group distance of the given 10 points is 1 if we choose a smaller ϵ, and the within-group distance of the same 10 points is 0 if we choose a larger ϵ. Although we demonstrate this problem using DBSCAN, most clustering methods face the same problem as well.



(a) The entropy is 1 since the 10 points are clustered into 2 groups.

(b) The entropy is 0 since all 10 points are clustered into the same group.

Figure 4.3: The entropy value of a data group that is estimated by a clustering method is sensitive to the selection of the clustering method and the corresponding parameters.

Second, the entropy distance function is not aware of the distance among clusters. Figure 4.4 shows two sets of points. The distance between the two data clusters in Figure 4.4 (b) is much larger than the distance between the two clusters in Figure 4.4 (a). Assume that points in both figures form similar dense clusters before their current observations. Intuitively, the within-group distance of the 10 points in Figure 4.4 (b) should be larger than the within-group distance of the 10 points in Figure 4.4 (a). However, since data points in both examples are clustered into two groups, their entropy distances are same (*i.e.*, 1).

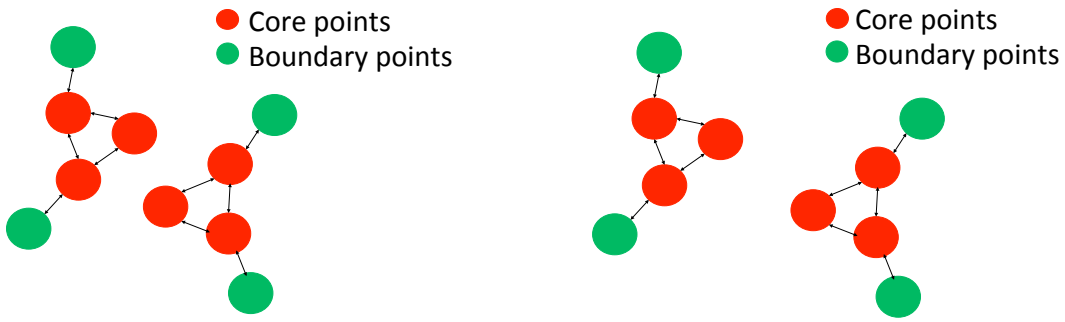
To solve the above mentioned limitations, we propose to use *the similarity-aware entropy function* to measure the within-cluster distance of any time-series cluster.

Definition 4.2. (SIMILARITY-AWARE ENTROPY FUNCTION) *Consider a time-series dataset* $\boldsymbol{X}$ *that contains* n *time series* $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$. *The similarity-aware entropy function of* $\boldsymbol{X}$ *during time period* $[t, t+1, \cdots, t+l]$ *is defined as below.*

$$S(\boldsymbol{X}(:, t:t+l)) = -\sum_{j=1}^{n} \frac{1}{n} \log \left(\sum_{i=1}^{n} \frac{1}{n} e^{\left(-d\left(\boldsymbol{x}_i(t:t+l), \boldsymbol{x}_j(t:t+l)\right)\right)} \right)$$

where $d\big(\mathbf{x}_i(t:t+l), \mathbf{x}_j(t:t+l)\big)$ *is the dissimilarity metric between time series* $\mathbf{x}_i$ *and time series* $\mathbf{x}_j$ *during time period* $[t, t+1, \cdots, t+l]$.

The proposed similarity-aware entropy function has the three properties that make it useful in G-CTC detection.



(a) (b)

Figure 4.4: The within-group distances of the two plots based on the entropy distance function are identical since each plot contains two clusters. However, compared with data in Figure (a), the two data clusters in Figure (b) are far away from each other, and hence the within-group distance of all data points in Figure (b) should be larger than the within-group distance of data points in Figure (a).

1. The similarity-aware entropy value is identical to the traditional entropy value when the distances among time-series clusters are large.
2. The similarity-aware entropy value is larger when time-series clusters are far away from each other and is small when time-series clusters are close to each other.
3. The similarity-aware entropy value is calculated directly from the given dataset. The cluster membership (or cluster ID) of each time series is not required.

The second and third properties are straightforward and address the two challenges when using traditional entropy function in G-CTC detection. Next, we use the two lemmas to show the correctness of the first property. More precisely, the following two lemmas show that the traditional entropy value of a time-series group is identical to the similarity-aware entropy value of the group when (i) the distances among time-series clusters in a the group is infinity, and (ii) time series in any cluster is similar to each other. Although it is hard to totally meet both of the two conditions in real-world applications, the proposed similarity-aware entropy still provides a good estimation of the within-group distance.

Lemma 4.2. *Let* $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k$ *be the time-series clusters in* $\boldsymbol{X}$ *such that for any*

$i, j \in [1, \cdots, k]$

$$d(\boldsymbol{X}_i, \boldsymbol{X}_j) = \begin{cases} \infty & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

Then, the traditional entropy of $\boldsymbol{X}$ *is identical to the similarity-aware entropy of* $\boldsymbol{X}$*, i.e.,*

$$E(\boldsymbol{X}|\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k) = H(\boldsymbol{X}|\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k)$$

Lemma 4.3. *Assume that (i) a group of time series* $\boldsymbol{X}$ *has* k *time-series clusters* $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k$*, and (ii) each cluster* $\boldsymbol{X}_i$ *is pure, i.e.,*

$$E(\boldsymbol{X}_i) = 0$$

Then, the similarity-aware entropy value of $\boldsymbol{X}$ *based on clusters* $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k$ *and the similarity-aware entropy value of* $\boldsymbol{X}$ *based on all time series* $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ *in* $\boldsymbol{X}$ *are same, i.e.,*

$$E(\boldsymbol{X}|\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k) = E(\boldsymbol{X}|\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n)$$

4.5 Experimental results

In this section, we first compare the running time of the original DBSCAN with both the indexing technique and the index + iterative distance computation. Then, we demonstrate the capability of the proposed G-CTC detection method using two case studies in finance and Earth science, respectively.

4.5.1 Scalability experiment

We implemented all the methods in C++ and run the codes on a desktop computer with an Intel Xeon 3.10GHz processor and 8GB of RAM. We compared the running time of the original DBSCAN implementation with (i) the indexing technique and (ii) the indexing + iterative distance computations. Figure 4.5 shows the running time of the three implementations. The result shows a speedup of up to 57× over the original implementation when using the indexing + iterative distance implementation.

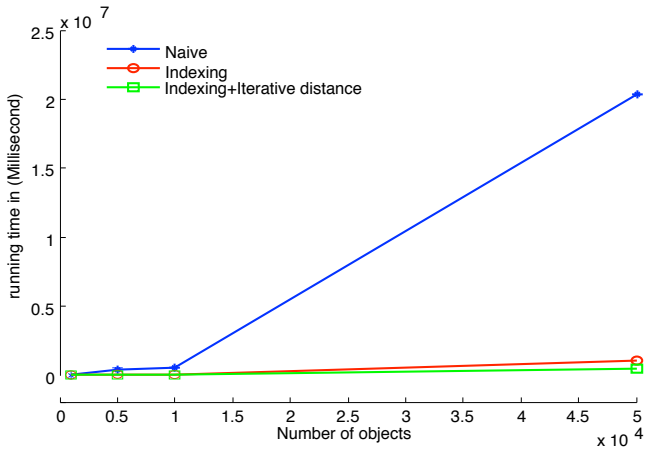


Figure 4.5: Running time of the original DBSCAN implementation and the two optimized implementations.

4.5.2 Case study I: Land cover change detection using remote-sensing data

The proposed G-CTC detection algorithm is useful in detecting land-cover changes. In this case study, we demonstrate the capability of the proposed method in wild fire detection using a remote-sensing dataset, the Enhanced Vegetation Index (EVI) dataset.

EVI is a remote-sensing dataset that are commonly used in wild fire detection [24, 67]. Roughly speaking, EVI indicates the "greenness" of the Earth's surface. The idea is a fire would change the greenness of an area drastically, and thus EVI would drop significantly. A major unsolved challenge is to detect fire in non-forest vegetation, such as Shrubland and Grassland. The greenness of these vegetation highly depends on the condition of mesoscale and microscale meteorology (*e.g.*, temperature and rainfall), which makes their EVI time series varying a lot. In addition, these land cover type recovers within 6 weeks. Hence, the drop in EVI due to fire can be insignificant compared with their nature variability.

The proposed G-CTC detection method is useful in distinguishing forest fires from events due to mesoscale and microscale meteorology (*e.g.*, droughts). In events like droughts, EVI time series of proximal patches show similar decreases. Therefore, there is no G-CTC occurred in the group of time series. In contrast, wild fires can only

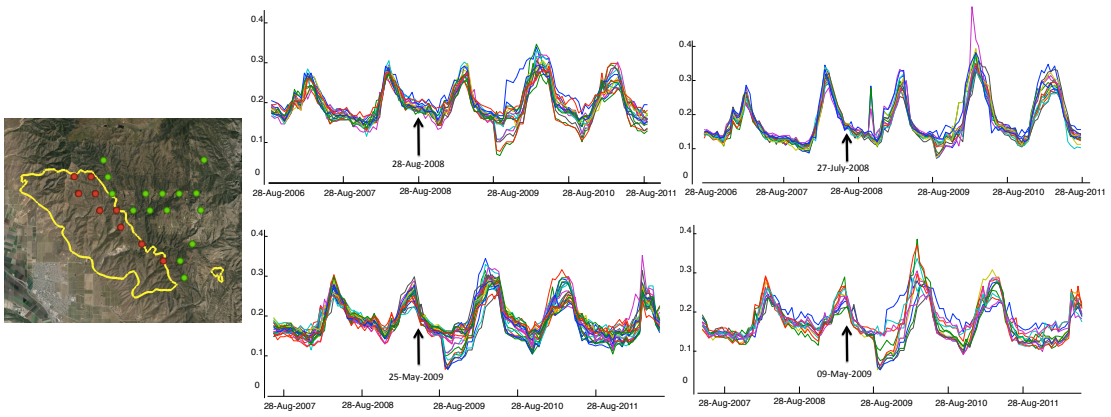


Figure 4.6: Four sets of EVI time series which disbands in August 2009 because of forest fire. Such disbanding pattern is useful to detect events from time-series datasets.

affect limited geographical regions. Hence, EVI time-series, especially the ones located close to the boundary of a burned area, typically exhibit contextual changes when fires occurred.

In this case study, we test the performance of our proposed framework in a region that is bounded by 36.5°N, 35.9°N, 121.2°W, and 122°W. This area is in the state of California (US) and contains 3345 time series. Each time series covers approximately 1 km^2 area. Twenty-five group disbanding events were detected between August 2008 to May 2011.

Figure 4.6 (b) - (e) shows four of these group disbanding events that correspond to the same fire event. We mark the locations of all plotted time-series by red (burned) and green (unburned) dots in Figure 4.6 (a). Besides, in Figure 4.6 (a), we use a yellow curve for the boundary of the whole burned area. In each of these events, EVI time-series that show similar patterns over two years before the change-point and then they split into roughly two groups around Aug 2009, when the patch bounded within the red line was burned. We check the time series that have low values around 28 Aug. 2009 manually and confirm that they are all located within the burned patch.

As Figure 4.6 shows, the four groups of time-series do not cover the left side of the burned area. The reason is as below. Locations in the left side covered by a different vegetation type compared with the locations on the right side. The time-series in the left side and the time-series on the right do not form a time-series cluster before the fire.

This phenomena the efficacy of context building in our algorithm. The airport shown in the figure is the Chalone Vineyard Airport region, which is different land-cover type than all the locations marked by yellow dots and is, again, not included in any of the groups for the same reason.

These four experimental results also show the limitation of the proposed method. We use two years data to calculate G-CTC scores. Hence, some change-points are detected earlier than the time when the event occurred. Event (b) and Event (c) are detected around Aug. 2008 and Event (d) and Event (e) are detected around May 2009.

4.5.3 Case study II: Event detection using stock price data

We test the proposed method using the daily closing price from NYSE of 5825 stock tickers between Apr. 1996 and Jul. 2013. We select $w = 60$ for this case study. Figure 4.7 shows one of the detected change-points. The change-point is in Jun. 2012. All the tickers in the group are Real Estate Investment Trust. In this group, two of the tickers significantly rise after June 2012. In contrast, the other time-series remain within the context and show stability for more than six months after the detected G-CTC.

We investigate the fork and find the two rising time-series are stock tickers from two self-service storage companies (EXR and SSS). The others are real estate companies in different parts of US and none of them does self-service storage business as per Google finance. The event started at June 2012 that is the usual time of the year for publishing the quarterly/annual financial reports.

4.6 Conclusion

We have proposed a framework for detecting group level contextual changes in a dataset of multiple related time series. The framework uses 3 components viz. (a) detect groups of time series (b) score events at each time instant and (c) report detected events. We also proposed 2 modifications to speed up the core AutoDBSCAN algorithm used in this framework. We demonstrated the results of applying this framework on 2 real world datasets, which detected plausible change events.

We identify the following four major items for future exploration. First, since AutoDBSCAN is based on DBSCAN, it can detect arbitrary-shaped clusters in which two

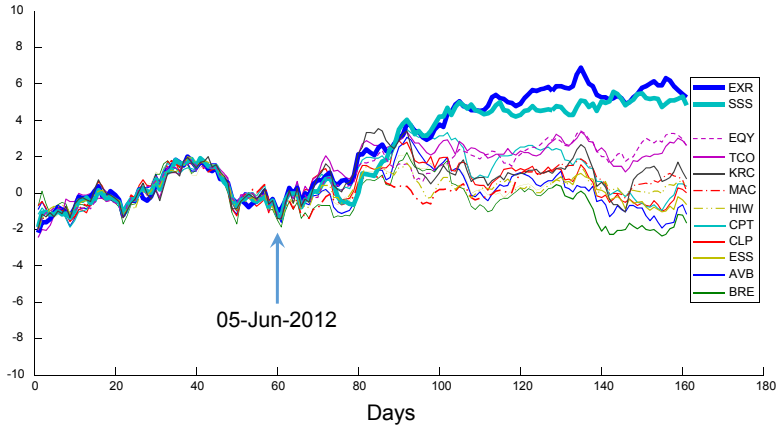


Figure 4.7: A G-CTC event in the stock price data showing a change in the grouping of REITs. The two Self-service Storage companies forked out from the general REIT context.

members of the same cluster can be very different from each other. In the 2 applications presented in this paper, we did not encounter this problem. However, it is a potential risk when using this framework in other applications. A potential work-around is to add a constraint on the largest pair-wise distance within a cluster. A future extension of this work is to design a clustering method that returns all possible time series groups in spherical clusters. Second, the framework as presented here has 5 user defined parameters viz. (a) w: window size used in constructing time series groups, (b) $[\epsilon_1, \epsilon_2]$: the range of the size of the $\epsilon-$neighborhood in the AutoDBSCAN method to detect groups of time series, (c) δ: step size in ϵ used in AutoDBSCAN, (d) b: window size of event scoring interval, and (e) the threshold used to report events. (Of these (b) and (c) are specific to the AutoDBSCAN clustering algorithm chosen for illustration). In this paper, all these parameters are empirically chosen based on domain knowledge. Techniques to automatically determine (or at least estimate reasonable values) of these parameters will be helpful in applying this framework to new domains. Third, although the framework assigns a higher similarity aware score for more significant splits in group disbanding or tighter clusters in group formation, it would be useful to estimate a confidence value for the detected change event (for example, the confidence value for a event detected from a random walk dataset should be much lower than a event detected from a normal time series dataset). Finally, we could extend this framework to detect other

interesting contextual events like *loops*, i.e., when one set of time series first disbands and then merges back. Such contextual events could provide interesting insights in fields of Ecology and Meteorology.

Chapter 5

Clustering dynamic spatio-temporal patterns in the presence of noise and missing data

5.1 Introduction

Clustering is one of the most common unsupervised data mining techniques. It has enjoyed tremendous success, especially for static data [48]. Yet, there is little work in the spatio-temporal setting where data is in the form of continuous spatio-temporal fields and the clusters are dynamic. Furthermore, spatio-temporal data that originate from earth-orbiting satellites, cell phones, and other sensors tend to be noisy, incomplete, and heterogeneous, making their analysis especially challenging [35]. When dealing with continuous spatio-temporal data, the clusters are embedded in the continuous spatio-temporal field, where these clusters or objects have no clear boundaries. The goal is to isolate such clusters from the background and continuously monitor them over time (see Figure 5.1).

In this work, we propose a novel spatio-temporal clustering paradigm to identify clusters in a continuous spatio-temporal field where clusters are dynamic and may change

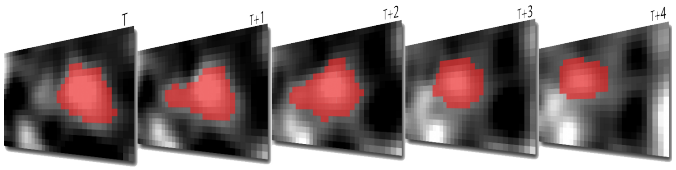


Figure 5.1: An example of a dynamic spatio-temporal cluster. In each time step (t_i) we must extract the cluster from the continuous field (red pixels) and track its evolution over time. Such patterns are common in the Earth Sciences, fMRI analysis, and other domains.

their size, shape, location, and statistical properties from one time-step to the next. Our paradigm stems from the observation that in numerous dynamic settings, although clusters may move or change shape, there are a number of points that do not change cluster memberships for a significant time-period. This observation allows us to autonomously extract dynamic clusters in continuous spatio-temporal data that may contain missing values, noise, or highly-variable features. We demonstrate our paradigm on a real-world application of monitoring in-land water bodies (e.g. lakes, dams, etc.) using remotely-sensed data on a global scale. We compare our method's performance to the K-MEANS and ExpectationMaximization (EM) clustering algorithms as well as the Normalized Cuts (NCUT) image segmentation algorithm, and find that our method's ability to leverage both spatial and temporal information makes it more robust to noise, missing data, and heterogeneity – common characteristics of emerging spatio-temporal datasets.

5.2 Background and related work

5.2.1 Problem formulation

The goal of this work is to autonomously extract dynamic clusters from a continuous spatio-temporal field. An fMRI recording or remotely sensed data are examples of continuous spatio-temporal fields, where each location has unique spatial coordinates and is characterized by uni/multi-dimensional time-series representing the evolution of the feature vector over time. Thus, given a continuous spatio-temporal field, our task is to identify all clusters in space and associate the clusters across time.

5.2.2 Existing clustering approaches

Clustering is a common data mining technique that groups similar points together to reveal high-level patterns in a dataset. Clustering algorithms may belong to two broad categories: feature-based clustering and constraint-based clustering.

Feature-based clustering algorithms group data based on their similarity in the corresponding feature space, without considering other information. Many popular clustering algorithms including K-MEANS [64], EM [66], LINKAGE (e.g., single-linkage [81], complete linkage [31]) and DBSCAN [34] all belong to this category.

Constraint-based clustering algorithms assign data to clusters based on additional constraints other than similarity in the feature space. For example, many image segmentation algorithms (e.g. [33, 80]) can be considered to be clustering methods with spatial constraints. They cluster data into spatially connected patches such that data from the same cluster have similar feature values and are also spatially connected. Other clustering algorithms, such as trajectory clustering [56], mining swarm/flock patterns [59] and moving clusters [58], are other examples of constraint-based clustering. They discover clusters of objects that have similar behavior over time.

Despite the wide applicability of these approaches, they do not address the fundamental needs of many spatio-temporal applications. For example, on the one hand, feature-based methods do not take into account spatial and temporal information that uniquely represent spatio-temporal clusters. Thus, in a feature-based setting, one either clusters the feature values or the spatial locations within data. On the other hand, in a constrained clustering setting, objects are already predefined and are grouped based on some constraints. However, in continuous spatio-temporal fields, there is no clear definition of an object, thus these methods have limited applicability. Finally, there have been works that cluster continuous spatio-temporal data into static clusters, such that the resulting clusters explain the data for the entire temporal duration (e.g. [7, 82]). Such approaches are not well-suited for the discovery of clusters over every time-step in the data, especially when the clusters are dynamic and may change size, shape, location and statistical properties over time.

A desirable solution is one that can isolate clusters that have similar feature values over space and time, while also keeping track of such clusters as they evolve. The most common approach to tackle this problem has been to either analyze the data in space

and then aggregate/associate over time, or by analyzing over time and then smoothing over space [75]. However, there is mounting evidence that such an approach may lead to false discoveries (e.g. [30, 39]).

5.2.3 Challenges

In addition to technical limitations, clustering spatio-temporal data faces significant data challenges in many real world applications. Figure 5.2 shows some of the common data challenges associated with analyzing spatio-temporal data. The data are routinely missing and noisy. Thus, analyzing the data on a snapshot-by-snapshot basis, or while disregarding spatial information would lead to inadequate performance.

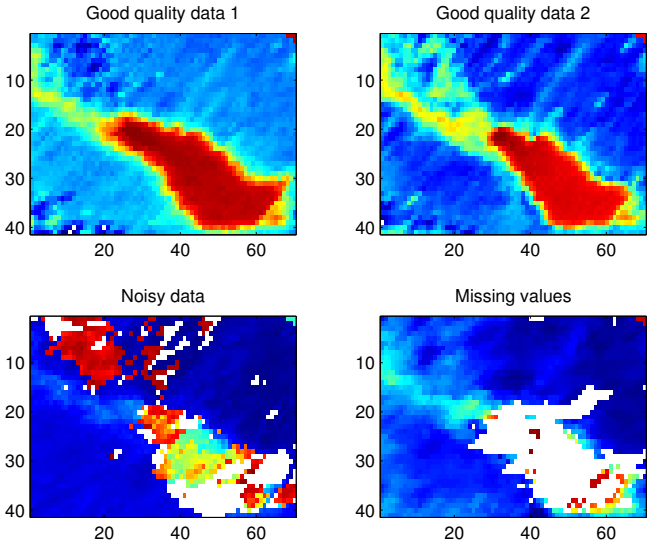


Figure 5.2: Examples of data challenges associated with spatio-temporal data. The data represent a remotely-sensed "wetness index" to estimate surface wetness. Each panel shows the wetness index of the same location at four different dates. As it can be seen, data quality often varies with time. The top row shows two dates where the data is of good quality, while the bottom row shows data with noisy and missing values (white pixels).

Another significant challenge is heterogeneity in space and time [36]. Heterogeneity in space refers to the case where data belonging to the different clusters may have the same feature values, despite being distinct "objects". Temporal heterogeneity refers to the instance where the feature values that uniquely discriminate a cluster change over

time for the same cluster. Figure 5.3 demonstrates the concept. The left panel shows the "wetness index" values for a region containing two lakes surrounded by land. On the top right panel, one notices a clearly distinguishable signal in the feature space (as seen by the bi-modal distribution of the feature values). However, in another time-step (bottom right panel) the feature space is not as informative due to spatio-temporal heterogeneity. Thus, relying solely on information in one time-step would yield inaccurate results.

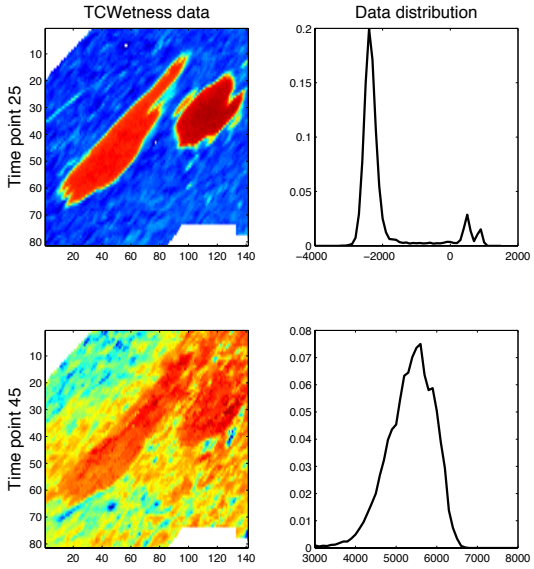


Figure 5.3: An example of data heterogeneity. Each row shows the "wetness index" values for the same lake at different time steps. The right panel shows the density of the feature values (pixel colors) on the left. In the top row, the water and land are easily distinguishable in the feature space however in the bottom row the two clusters are not distinguishable. Thus, relying solely on that time-step would yield inaccurate results.

5.3 A spatio-temporal clustering paradigm

To address the above-mentioned challenges, we propose a general spatio-temporal clustering paradigm that systematically leverages the very challenges that affect traditional methods to identify dynamic spatio-temporal clusters. Our paradigm consists of two main steps: identifying the most certain cluster memberships and iteratively finalizing the most uncertain points which will likely be at the cluster boundaries where dynamics

occur. Figure 5.4 outlines the four key steps. The first step specifies the clustering objectives such as separating certain activity from the background, or labeling the clusters with target labels. The second step is to identify "core points", which are the points that for a given time-window do not change cluster memberships. The key here is to choose an appropriate time window size. In practice, one could identify core points for each snapshot by examining data from the previous and upcoming time-steps. The third step is to finalize cluster memberships along the cluster's boundary. Given the dynamic nature of the clusters and the uncertainty in the data, the boundary points are going to be more challenging to cluster. While the exact approach may differ, the idea is to use information from the core points (especially the ones that are spatially nearby) to finalize cluster assignments. The fourth and final step is to post-process the cluster result in case not all points have been labeled.

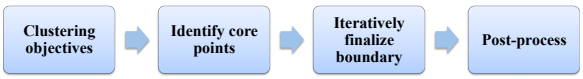


Figure 5.4: The proposed four-step spatio-temporal clustering paradigm.

5.4 Proposed method

The proposed paradigm takes advantage of the fact that in many domains, although the clusters may move, there are "core points" that never change cluster memberships for a given time window. This is an important observation when the data may be missing or noisy between time-steps, as such, although clusters might not be separable during every single time-step, borrowing stronger signals from other time steps helps overcome some of these challenges.

While there are many ways to implement this paradigm, this section presents one such realization in practice. We hope that introducing this paradigm to the community will allow us to leverage our collective creativity to design a host of methods to analyze space-time data.

5.4.1 Clustering objectives

The first step in the paradigm is to articulate the objectives of our clustering analysis. In the case of global water monitoring, we are given a single-dimensional spatio-temporal field without any notion of water or land. The goal is to extract clusters and their dynamics over a fifteen-year period and then label each cluster as water and land in the post-process phase. We monitor surface water using a "wetness index", known as `TCWETNESS`. `TCWETNESS` has been widely used in mapping and monitoring land use/land cover by the remote sensing community [27, 29]. The steps used to produce `TCWETNESS` are discussed in detail in [23].

5.4.2 Discover stable clusters

After specifying the clustering objective, the second step of the paradigm is to identify groups of data that rarely change cluster membership for a given time window. Points in any stable cluster are expected to be contiguous in space and also have similar temporal characteristics during the pre-defined time window. The main motivation behind stable clusters is that points are grouped together not only based on their spatial connectivity but also their long-term temporal similarity. This is critical in noisy and incomplete data as the features might not be informative at every time-step, but over a long enough period, similar time-series would emerge.

Our spatio-temporal method that identifies stable clusters is an extension of the traditional DBSCAN algorithm. DBSCAN [34] is a density-based clustering algorithm. It groups data that are closely packed together in the feature space. The algorithm identifies "core points" that have at least m neighbors within an ϵ distance in the feature space. Unlike DBSCAN which only considers distances in the feature space, our approach seeks to associate points that are adjacent in space and have similar feature values over a non-trivial time window. Similar to the ST-DBSCAN method proposed by [7], we use both spatial and temporal information in finding $\epsilon-$ neighbors. The main distinction between our approach and [7] is that we are interested in identifying clusters for every time-step and associating these clusters accross time, despite noise and missing data. While [7] returns only a single cluster for the entire time period. To do so, we propose a new spatio-temporal distance metric.

We use the notion of spatio-temporal $\epsilon-$ neighbors to denote two points that are spatially connected and have similar temporal characteristics. Thus, given a distance function that accounts for both spatial adjacency and temporal similarity, two points are spatio-temporal $\epsilon-$ neighbors if their spatial-temporal distance is smaller than ϵ. Points that have more than m ST $\epsilon-$ neighbors are defined as core points. Once the core points are identified, the algorithm iteratively associates the core points with their spatio-temporal $\epsilon-$ neighbors.

The steps of our method is shown in Fig. 5.5. The algorithm first detects core points (shown as red dots in Fig. (a)) based on a predefined distance function. Then it creates an edge between all core points and their own spatio-temporal $\epsilon-$ neighbors (as shown in Fig. (b)). Finally, it groups points that have been connected by an edge as clusters. In the given example, as shown in Fig. (c), two clusters are discovered (orange and green) and there is a single point (yellow) that does not belong to any cluster.

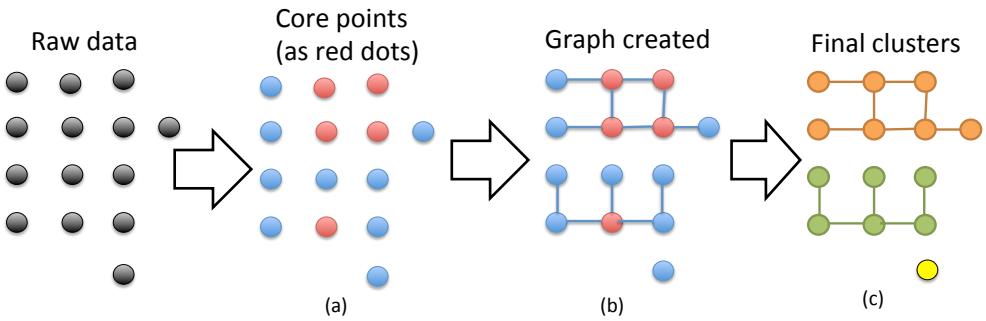


Figure 5.5: The steps of creating stables clusters from a spatio-temporal data using ST-DBSCAN.

The spatio-temporal distance function that we choose forces two spatio-temporal $\epsilon-$ neighbors to be spatially adjacent and have similar temporal characteristic. It is a function as below.

$$\mathrm{d}_{st}(x,y) = \begin{cases} \mathrm{d}_t(x,y) & \text{if x and y are spatial neighbors} \\ 0 & \text{otherwise} \end{cases}$$

where, $\mathrm{d}_t(x,y)$ is a time-series distance function.

The choice of time series distance function is related to the application. Commonly used time-series distance functions include (but are not limited to) Euclidean distance,

Pearson's correlation and kth order statistic [25]. Pearson's correlation is preferred when the trend of time-series is more important than the actual values. Euclidean distance is suceptible to noise and outliers [55]. The *kth* order statistic distance is a time series distance function that is robust to outliers. However, it requires an estimation of the outlier properties. In the scenario where the property of the data changes over time and space, the kth order statistic distance is not suitable.

We propose to estimate the distance between two time series based on how similar they are over a period of time.

Definition 5.1. ***(Temporal similarity)*** *Two time series are temporally similar if they have the same expected value for the entire duration.*

To use temporal similarity, we assume that our data follow the additive white noise model, i.e., any real observation of object x at time t, $x(t)$, is the summation of its true value $\hat{x}(t)$ and a random white noise signal $n(x,t)$ as shown below.

$$x(t) = \hat{x}(t) + n(x,t)$$

When two objects x and y are temporally similar, their true value at any time are identical. Hence,

$$x(t) - y(t) = n(x,t) - n(y,t)$$

Since we assume the white noise model, the expectation of any noise is zero. Thus,

$$\begin{aligned} E(x(t) - y(t)) &= E(n(x,t) - n(y,t)) \\ &= E(n(x,t)) - E(n(y,t))) \\ &= 0 \end{aligned}$$

Therefore, we can estimate the temporal similarity of two time series x and y as the $p - value$ of the following test.

$$H_0 : E(\mathbf{x} - \mathbf{y}) = 0$$
$$H_a : E(\mathbf{x} - \mathbf{y}) \neq 0$$

Since outliers may negatively impact expectations, an alternate test can be used when the data are susceptible to outliers.

$$H_0 : \text{median}(\mathbf{x} - \mathbf{y}) = 0$$
$$H_a : \text{median}(\mathbf{x} - \mathbf{y}) \neq 0$$

Thus, under the white noise assumption, we propose that two time-series are similar if their difference over the given duration is centered around zero. We can use the p-value of such a hypothesis test, e.g., the Kolmogorov-Smirnov test [65], as the measure of similarity.

An example of stables clusters discovered for an area containing two lakes is shown in Figure 5.6. The three stable clusters are highlighted in yellow, red and green. The dark blue locations around the clusters are points that we cannot yet assign to any cluster and we must rely on the third step in our paradigm to finalize cluster memberships. We refer to such points as "uncertain points"

The right panel of Figure 5.6 shows the temporal profile of the clusters in the figure's left panel. We highlighted the time-series with the same color of the cluster they were assigned to. The yellow and red time-series have very similar temporal profiles and overlap for almost the entire record. However, notice that data in the light blue (land) pixels have different feature values from the yellow and red (water) pixels only during some periods. This is where our temporal similarity over a long time window helps overcome spatio-temporal heterogeneity.

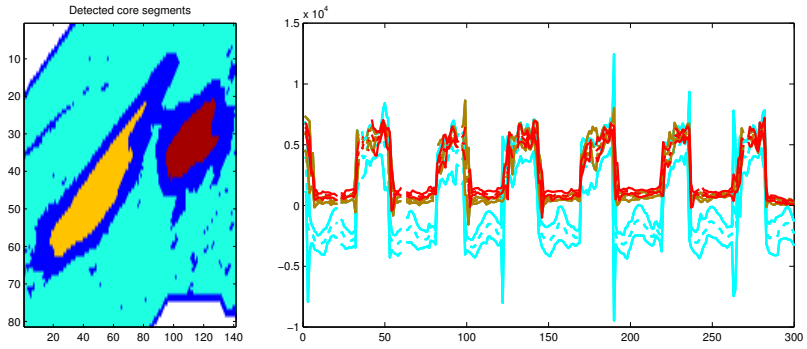


Figure 5.6: Core segments and their corresponding temporal profiles

Our temporal similarity measure is sensitive to the choice of time window length. Specifically, the window length w impacts the number of points that will be assigned

to stable clusters and/or the number of uncertain points. If we set w to 1 then, our method would be similar to many of the traditional clustering algorithm that disregard time (*e.g* NCUT) and it would not be able to cluster time-steps where data are missing. If we choose a too broad w then we might include too much uncertainty from the highly variable data signal or the changing properties of the dynamic cluster, which would increase the proportion of "uncertain points".

5.4.3 Growing and refining clusters for each time point

The third step of our paradigm finalizes cluster memberships by assigning "uncertain points" to clusters and correcting any assignment mistakes from the previous step. This step relies on the cluster assignments from the previous step to build a spatial predictive model. We use the constructed model to predict the cluster membership of unassigned points based on their feature values in the current time-step. However, given the spatial-heterogeneity in the data, we propose a layer-based classification method which iteratively assigns the uncertain points to existing stable clusters based on spatial proximity and the feature value at the current time-step. Unlike the first step in the paradigm, this classification step only uses information for the current time-step.

Assuming we identified k stable clusters in the previous step, we build a k-class classifier to determine the probability that a given uncertain point belongs to one of the k stable clusters. The model is trained using the feature values of the points in the stable clusters, with each point having a feature value and stable cluster membership. The algorithm then tries to classify the first uncertain point which is at the boundary of a stable cluster using the learned classifier. By assuming clusters are spatially contiguous, an uncertain point is more likely to have the same label as its stable neighbor. Thus after we classify an uncertain point, if its resulting label is the same as its neighbor from a stable cluster we accept that labeling, and move on the next uncertain point. If the label assigned to the uncertain point is inconsistent with its neighbor from the stable cluster, we label the point as "unknown". The idea is to delay an uncertain labeling until more data are available to make an unambiguous assignment.

Figure 5.7 illustrates our layer-based method. In this example, there are two stable clusters in green and light blue. The points in white are the uncertain points that we have not labeled. We attempt to assign these uncertain points to existing clusters in the

layer-based fashion by first assigning the points that are adjacent to existing clusters. The first layer of uncertain points to be classified are highlighted in red in the left panel of Figure 5.7. Then these uncertain points (red points) are classified based on their feature values using a classifier trained on the feature values of the points in the stable clusters. The classification results in this example are shown in the middle panel of Figure 5.7. Finally, the algorithm checks for spatial consistency such that any newly labeled point should have the same label as its neighbor from the stable clusters. We relabel any points with inconsistent labels as "uncertain" and we repeat the classification procedure until no points change class membership.

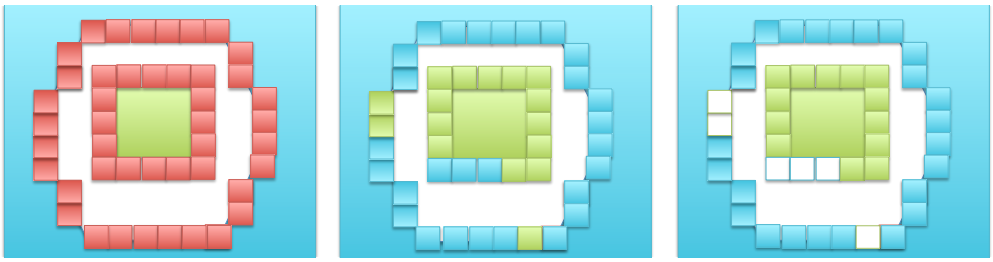


Figure 5.7: Illustrative example of layer based classifier

While numerous classifiers could be used, we chose a local Bayesian classifier. For any given time, we consider observations that are spatially nearby and also in the same cluster to follow a Normal distribution. Then, for each neighborhood, we train a Bayesian classifier. Its conditional probability for any cluster C as

$$p(x|x \in C, t) \sim N(\mu(C_R, t), \sigma(C_R, t))$$

where $\mu(C_R, t)$ is the sample mean of all points belonging to stable cluster C and within region R (i.e., a spatial region that is centered around x) at time t and similarly $\sigma(C_R, t)$ is the sample standard deviation of all points belonging to clustering C and in region R.

The prior probability of $p(x \in C|t)$ is a function of the spatial distance between x and the cluster C. It is independent of t. Specifically,

$$p(x \in C|t) = e^{-\frac{||x,C||_{space}}{\delta_d^2}}$$

where $||x, C||_{space}$ is the spatial distance between the point x and cluster C. δ_d is a

parameter that controls the weight of the spatial constraint. The larger δ_d is, the smaller the impact of the spatial distance on the prior probability. Thus for every uncertain point at the boundary of stable clusters, we would assign it to the cluster with the highest probability.

5.5 Experimental results

To test the performance of our approach, by autonomously extracting all in-land water bodies from 166 lakes regions on a global scale (see Figure 5.8). We compared our performance against that of Normal-cut (an image segmentation method) [80] and Gao et al.'s method [40] – a K-MEANS based approach used by the water resources management community. To compare the performance of the three methods, we use ground truth data from the Shuttle Radar Topography Mission's (SRTM) Water Body Dataset (SWBD), which consists of a global water body map for Feburary 2000. The SWBD data contains most water bodies for a large fraction of the Earth (60^o S to 60^O N) and is publicly available through the MODIS repository as the MOD44W product. For verification purposes, we compare each algorithm's output (i.e. water pixels and land pixels) for the Feburary 18th 2000 snapshot of `TCWETNESS` (the closest MODIS date near the time SWBD was collected) against the SWBD data

We evaluate the performance of the algorithms on each lake region independently using the F_1 score – measure that conveys the balance between a model's precision and recall [84]. To do so, we consider the water locations as the positive set and the land locations as the negative set. The F_1 score of each model can be obtained by comparing an algorithm's output and its corresponding validation set [84].

Figure 5.9 shows the overall performance of each algorithm across all 166 lakes. Overall, our proposed method is more robust compared to NCUT and Gao et al.'s method as seen by the higher median F1 score as well as a smaller inter-quartile range (the blue box).

To further understand our algorithm's performance in the presence of noise and missing values, we repeated the same experiment except that we segmented the data into groups of increasing difficulty. In the first experiment we grouped the test data of February 2000 into the three groups based on the percentage of missing data in

Figure 5.8: Positions of the 166 lake regions used in evaluating the performance of the proposed spatio-temporal clustering method.

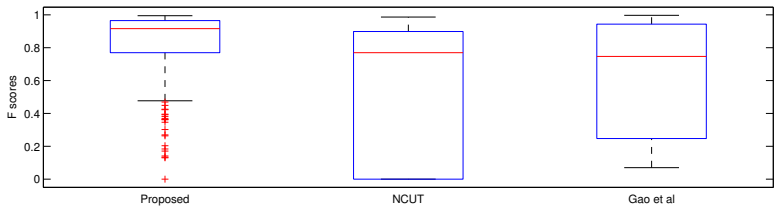


Figure 5.9: The performance of the proposed spatio-temporal clustering algorithm and the two baselines on the test 166 lakes.

that snapshot. The three groups were: (i) regions that had less than 10% missing data; (ii) regions that had more than 10% but less than 60% missing data; and (iii) regions that had more than 60% missing data. Figure 5.10 shows the performance of each algorithm as a function of the percentage of missing data. We find that when the data are relatively complete (left panel of Figure 5.10) all three methods perform similarly. However, as the percentage of missing data increases, our proposed method outperforms the baseline methods. Note than in extreme cases where most of the data are missing NCUT breaks down completely, while our method is able to recover thanks to its reliance on information from multiple time-steps.

In another experiment, we segmented the test data based on how noisy the data were. We considered the 107 lakes that had less than 10% missing data. We separated the 107

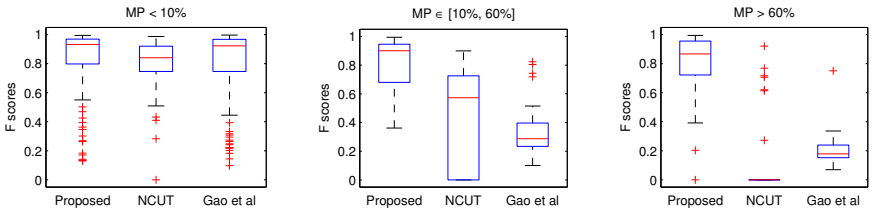


Figure 5.10: The performance of the proposed spatio-temporal clustering algorithm and the two baselines as a function of missing data.

lakes into three groups based on their classification difficulty (e.g. how sperable were the land and water pixels in the feature space). We used the Bhattacharyya coefficient (BC) [28] to measure how separable were the land and water `TCWETNESS` feature distributions. A BC measure of 0 means that the two distributions are completely separable. The three groups we evaluated were: (i) regions with a BC smaller than 0.05; (ii) regions with BC larger than 0.05 and smaller than 0.1; and (iii) regions with BC larger than 0.1 but smaller than 0.5. The performance of each algorithm on the different groups is shown in Figure 5.11. The results show that when water and land locations are high separable in the feature space, all three methods perform equally well. However, when the features from a single time-step are not discriminative enough, using both spatial and temporal information is better than using only information from the feature space.

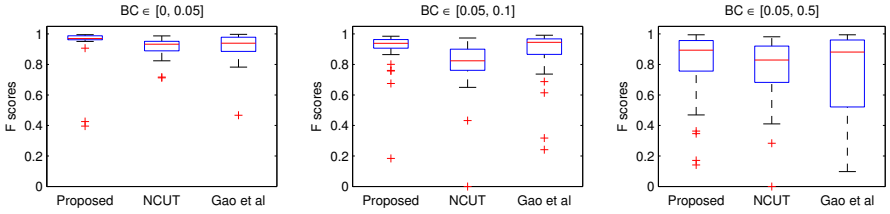


Figure 5.11: The performance of the proposed spatio-temporal clustering algorithm and the two baselines as a function of noise.

5.6 Conclusion

In this paper, we introduced a new spatio-temporal paradigm to identify dynamic clusters from a continuous spatio-temporal field where data might be missing or noisy. The intuition behind this work is that although the cluster may move slightly from time-step

to the next (and thus some points may change cluster membership), there are core points that never change clusters across the entire time. Our method used spatial contiguity and temporal similarity assumptions to overcome the limitations of non-space-time-aware methods especially in the presence of noise and missing values – two common characteristics of satellite products. We presented one implementation of the paradigm and expect numerous innovations on how to exactly carry it out. Our method can be used by domain scientists and sustainability experts to study the dynamics of in-land water availability and may potentially lead data driven resource management. One avenue of future work could be the automatic choice of the window size w. In our case we chose a time period of five years, but other more systematic ways to choosing the time window size should be explored.

Chapter 6

Conclusion and Discussion

6.1 Summary

Event detection in noisy, incomplete, and heterogeneous spatio-temporal data is a broad research area. Instead of exploring every aspect of this research area, this dissertation focuses on four novel algorithms that cover two types of events: change points and moving areas. Table 6.1 summarizes the four algorithms that are included in this dissertation.

Cha.	**Method**	**Type of events**	**Data issues**			
			Noise	Missing values	Heterogeneity	
					Temporal	Spatial
2	PC framework	Modeling-based change-points	✓			N/A
3	S-CTC detection	Singleton contextual change-points			✓	N/A
4	G-CTC detection	Group-level contextual change-points			✓	N/A
5	Spatio-temporal clustering method	Growing and shrinking moving regions	✓	✓	✓	✓

Table 6.1: The summary of the proposed algorithms in this dissertation.

Chapter 2 introduces a PC framework. The PC framework is proposed to detect modeling-based change points when time-series data are noisy and contain outliers. Most existing modeling-based change-point detection algorithms are sensitive to the

above data issues. The PC framework can be applied together with most existing modeling-based methods and increase their robustness to noise and outliers.

Chapter 3 and Chapter 4 introduce S-CTC and G-CTC detection algorithms respectively. These two algorithms are proposed to detect two different types of contextual change points. Most traditional change-point detection methods assume that observations in a time series have certain consistency with historical data in the same time series. Hence, when data contains temporal heterogeneity, traditional methods cannot work well. Contextual change points, in contrast, was defined according to the relationship among multiple time series. Since they do not assume a fixed relationship among observations at different time-steps in any time-series, contextual change points can be used in detecting certain types of events when time-series data is heterogeneous.

Chapter 5 presents a spatio-temporal clustering method that can discover moving regions that are shrinking and growing over time in spatio-temporal data. Most existing methods either analyze data in space and then aggregate over time or analyze data over time and then smooth the result over space. Since these methods cannot fully utilize the spatial contiguity and temporal similarity in the spatio-temporal data, their performance in noisy, incomplete, and heterogeneous data is poor. The proposed method is a clustering method that uses both spatial and temporal information together. It outperforms the other unsupervised methods especially in the presence of noise and missing values.

6.2 Future work

We have developed techniques to detect certain types of events from spatio-temporal data. Yet, event detection in noisy, incomplete, and heterogeneous spatio-temporal data is still quite an open research problem. Next, we conclude several research problems that are commonly faced by but still unsolved in the proposed methods. Besides, we also discuss several new extensions of the work that are present in this dissertation.

6.2.1 Parameter selection

The four proposed algorithms, similar to most existing unsupervised methods, have several user-defined parameters. Typically, the performance of an event detection method

depends on the selected parameters. In this dissertation, most parameters are empirically chosen based on domain knowledge for each application independently. This exercise, although it works in many applications, can be very time consuming and sometimes unreliable. Thus, a method that autonomously learns the user-defined parameters is a valuable and also commonly faced research question.

6.2.2 Scalable algorithms

Spatio-temporal data tends to be very large and ever-growing. For example, the volume of climate data in year 2010 is at least 10 Petabyytes (1 PB = 1,000 TB) and this number is projected to grow exponentially to about 350 Patabytes by year 2030 [37, 70]. This highlights the need for developing scalable versions of existing algorithms that can process these datasets within a reasonable time duration. Most existing methods, including the proposed methods, mainly focus on the accuracy of detection results. Such a research bias leaves a big open area on scalable event detection algorithms.

6.2.3 New extensions of current work

The four proposed algorithms are designed to detect certain types of events from spatio-temporal data. Many techniques in the proposed work can be modified and help in discovering other types of events. For example, we could extend S-CTC and G-CTC detection algorithms to discover contextual events like *loops* (*i.e.*, when one set of time series first disbands and then merges back). Such patterns could provide interesting insights in fields of Ecology and Meteorology. Another example is to detect moving regions that may also shift their centroids by combining estimation and filtering methods (*e.g.* Kalman filter) into the proposed spatio-temporal clustering method. These moving patterns have huge applications in climate (*e.g.*, tracking hurricanes) and social sciences (*e.g.*, understanding immigration or urbanization processes).

References

[1] California Department of Forestry and Fire Protection, Fire and Resource Assessment Program. `http://frap.fire.ca.gov`.

[2] Criminal justice policy research institute:portland crime data. `http://www.pdx.edu/crime-data/hotspot-past-5-years-11`. Accessed: 2016-10-30.

[3] R. P. Adams and D. J. MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[4] R. Aggarwal, C. Inclan, and R. Leal. Volatility in emerging stock markets. *Journal of Financial and Quantitative Analysis*, 34(01):33–55, 1999.

[5] M. Basseville, I. V. Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[6] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of KDD '94: AAAI Workshop on Knowledge Discovery in Databases*, pages 359–370, 1994.

[7] D. Birant and A. Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[8] R. Bolton and D. Hand. Unsupervised profiling methods for fraud detection. In *Credit Scoring and Credit Control VII*, 2001.

[9] S. Boriah. *Time Series Change Detection: Algorithms for Land Cover Change*. PhD thesis, University of Minnesota, 2010.

[10] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. Inferring causal impact using Bayesian structural time-series models. *Annals of Applied Statistics*, 9:247–274, 2015.

[11] G. Brown et al. Social–ecological hotspots mapping: a spatial approach for identifying coupled social–ecological space. *Landscape and urban planning*, 85(1):27–39, 2008.

[12] S. Budalakoti, A. N. Srivastava, R. Akella, and E. Turkov. Anomaly detection in large sets of high-dimensional symbol sequences. 2006.

[13] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.

[14] P. Casale, O. Pujol, and P. Radeva. Personalization and user verification in wearable systems using biometric walking patterns. *Personal and Ubiquitous Computing*, 16(5):563–580, 2012.

[15] S. Chainey and J. Ratcliffe. *GIS and crime mapping*. John Wiley & Sons, 2013.

[16] S. Chainey, L. Tompson, and S. Uhlig. The utility of hotspot mapping for predicting spatial patterns of crime. *Security Journal*, 21(1-2):4–28, 2008.

[17] V. Chandola and R. R. Vatsavai. A gaussian process based online change detection algorithm for monitoring periodic time series. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011*, pages 95–106, 2011.

[18] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[19] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences: a survey. *Knowledge and Data Engineering, IEEE Transactions on*, (99): 1–1, 2010.

[20] D. Chen, C.-T. Lu, Y. Kou, and F. Chen. On detecting spatial outliers. *Geoinformatica*, 12(4):455–475, 2008.

[21] J. Chen and A. Gupta. On change point detection and estimation. *Communications in Statistics: Simulation & Computation*, 30(3):665–697, 2001.

[22] X. Chen, V. Mithal, S. R. Vangala, I. Brugere, S. Boriah, and V. Kumar. A study of time series noise reduction techniques in the context of land cover change detection. Technical Report 11-016, Computer Science Department, University of Minnesota, 2011.

[23] X. Chen, A. Khandelwal, S. Shi, J. Faghmous, S. Boriah, and V. Kumar. Unsupervised method for water surface extent monitoring using remote sensing data. *Machine Learning and Data Mining Approaches to Climate Science: Proceedings of the Fourth International Workshop on Climate Informatics*, 2015.

[24] X. C. Chen, A. Karpatne, Y. Chamber, V. Mithal, M. Lau, K. Steinhaeuser, S. Boriah, M. Steinbach, V. Kumar, C. Potter, S. A. Klooster, T. Abraham, J. D. Stanley, and J. C. Castilla-Rubio. A new data mining framework for forest fire mapping. In *2012 Conference on Intelligent Data Understanding, CIDU 2012*, pages 104–111, 2012.

[25] X. C. Chen, K. Steinhaeuser, S. Boriah, S. Chatterjee, and V. Kumar. Contextual time series change detection. In *SDM*, pages 503–511. SIAM, 2013.

[26] T. Cheng and Z. Li. A multiscale approach for spatio-temporal outlier detection. *T. GIS*, 10(2):253–263, 2006.

[27] J. B. Collins and C. E. Woodcock. An assessment of several linear change detection techniques for mapping forest mortality using multitemporal landsat {TM} data. *Remote Sensing of Environment*, 56(1):66 – 77, 1996.

[28] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149. IEEE, 2000.

[29] P. R. Coppin and M. E. Bauer. Digital change detection in forest ecosystems with remote sensing imagery. *Remote Sensing Reviews*, 13(3-4):207–234, 1996.

[30] I. Davidson, S. Gilpin, O. Carmichael, and P. Walker. Network discovery via constrained tensor analysis of fMRI data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–202. ACM, 2013.

[31] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[32] J. Eck, S. Chainey, J. Cameron, and R. Wilson. Mapping crime: Understanding hotspots. 2005.

[33] A. J. Enright, S. Van Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research*, 30(7):1575–1584, 2002.

[34] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[35] J. H. Faghmous and V. Kumar. Spatio-temporal data mining for climate data: Advances, challenges, and opportunities. In W. Chu, editor, *Data Mining and Knowledge Discovery for Big Data: Methodologies, Challenges, and Opportunities*, pages 83–116. Springer, 2013.

[36] J. H. Faghmous and V. Kumar. A big data guide to understanding climate change: The case for theory-guided data science. *Big data*, 2(3):155–163, 2014.

[37] J. H. Faghmous and V. Kumar. Spatio-temporal data mining for climate data: Advances, challenges, and opportunities. In *Data Mining and Knowledge Discovery for Big Data*, pages 83–116. Springer, 2014.

[38] J. H. Faghmous, V. Mithal, M. Uluyol, M. Le, L. Styles, S. Boriah, and V. Kumar. Multiple hypothesis object tracking for unsupervised self-learning: An ocean eddy tracking application. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[39] J. H. Faghmous, H. Nguyen, M. Le, and V. Kumar. Spatio-temporal consistency as a means to identify unlabeled objects in a continuous data field. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 410–416, 2014.

[40] H. Gao, C. Birkett, and D. P. Lettenmaier. Global monitoring of large reservoir storage from satellite remote sensing. *Water Resources Research*, 48(9), 2012.

[41] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[42] A. Goldenberg, G. Shmueli, R. A. Caruana, and S. E. Fienberg. Early statistical detection of anthrax outbreaks by tracking over-the-counter medication sales. *Proceedings of the National Academy of Sciences*, 99(8):5237–5240, 2002.

[43] W. Gould. Remote sensing of vegetation, plant species richness, and regional biodiversity hotspots. *Ecological applications*, 10(6):1861–1870, 2000.

[44] M. Gupta, A. B. Sharma, H. Chen, and G. Jiang. Context-aware time series anomaly detection for complex systems. In *SDM Workshop on Data Mining for Service and Maintenance (2013)*, SDM'13, 2013.

[45] V. Guralnik and J. Srivastava. Event detection from time series data. In *KDD '99: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 33–42, 1999.

[46] F. Gustafsson. The marginalized likelihood ratio test for detecting abrupt changes. *Automatic Control, IEEE Transactions on*, 41(1):66–78, 1996.

[47] J. D. Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[48] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[49] J. Jones and T. Nguyen. *California's Drought of 2007-2009: An Overview*. State of California, Natural Resources Agency, California Department of Water Resources, 2010.

[50] M. C. Jun, H. Jeong, and C.-C. Kuo. Distributed spatio-temporal outlier detection in sensor networks. In *Defense and Security*, pages 273–284. International Society for Optics and Photonics, 2005.

[51] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD 2005: Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, pages 364–381, 2005.

[52] R. Killick and I. Eckley. changepoint: An R package for change point analysis. *Journal of Statistical Software*, 58(3):1–19, 2014.

[53] A. Kut and D. Birant. Spatio-temporal outlier detection in large databases. *CIT. Journal of computing and information technology*, 14(4):291–297, 2006.

[54] T. L. Lai. Sequential changepoint detection in quality control and dynamical systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(4): 613–658, 1995.

[55] L. J. Latecki, V. Megalooikonomou, Q. Wang, R. Lakaemper, C. Ratanamahatana, and E. Keogh. Partial elastic matching of time series. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[56] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[57] X. Li, Z. Li, J. Han, and J.-G. Lee. Temporal outlier detection in vehicle traffic data. In *ICDE '09: Proceedings of the IEEE International Conference on Data Engineering*, pages 1319–1322, 2009.

[58] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 617–622. ACM, 2004.

[59] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.

[60] Z. Li, J. Han, M. Ji, L. Tang, Y. Yu, B. Ding, J. Lee, and R. Kays. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):37, 2011.

[61] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[62] J. Lin, E. Keogh, A. Fu, and H. Van Herle. Approximations to magic: Finding unusual medical time series. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pages 329–334. IEEE, 2005.

[63] S. Liu, M. Yamada, N. Collier, and M. Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 43:72–83, 2013.

[64] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[65] F. J. Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[66] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[67] V. Mithal, A. Garg, I. Brugere, S. Boriah, V. Kumar, M. Steinbach, C. Potter, and S. Klooster. Incorporating natural variation into time series-based land cover change identification. In *CIDU11: Proceedings of the 2011 NASA Conference on Intelligent Data Understanding*, 2011.

[68] S. V. Nath. Crime pattern detection using data mining. In *Web Intelligence and Intelligent Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on*, pages 41–44. IEEE, 2006.

[69] D. B. Neill. Fast subset scan for spatial pattern detection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(2):337–360, 2012.

[70] J. T. Overpeck, G. A. Meehl, S. Bony, and D. R. Easterling. Climate data challenges in the 21st century. *science*, 331(6018):700–702, 2011.

[71] E. Page. Controlling the standard deviation by cusums and warning lines. *Technometrics*, 5(3):307–315, 1963.

[72] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1):100–115, 1954.

[73] J. H. Ratcliffe. The hotspot matrix: A framework for the spatio-temporal targeting of crime reduction. *Police practice and research*, 5(1):5–23, 2004.

[74] W. V. Reid. Biodiversity hotspots. *Trends in Ecology & Evolution*, 13(7):275–280, 1998.

[75] J. F. Roddick and M. Spiliopoulou. A bibliography of temporal, spatial and spatio-temporal data mining research. *ACM SIGKDD Explorations Newsletter*, 1(1):34–38, 1999.

[76] P. Rodrigues, J. Gama, and J. Pedroso. Hierarchical clustering of time-series data streams. *Knowledge and Data Engineering, IEEE Transactions on*, 20(5):615–627, 2008.

[77] A. O. C. Romero. Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach. *Mestrado, University of Twente*, 2011.

[78] G. J. Ross. Parametric and nonparametric sequential change detection in r: The cpm package. *Journal of Statistical Software*, 66, 2015.

[79] A. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, pages 507–512, 1974.

[80] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[81] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[82] M. Steinbach, P.-N. Tan, V. Kumar, S. Klooster, and C. Potter. Discovery of climate indices using clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 446–455. ACM, 2003.

[83] J. Takeuchi and K. Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE Trans. Knowl. Data Eng.*, 18(4):482–492, 2006.

[84] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005.

[85] P. Temin. The great recession & the great depression. *Daedalus*, 139(4):115–124, 2010.

[86] R. S. Tsay. Outliers, level shifts, and variance changes in time series. *Journal of forecasting*, 7(1):1–20, 1988.

[87] US Geological Survey and NASA. Land Processes Distributed Active Archive Center (LP DAAC). `https://lpdaac.usgs.gov/`.

[88] J. Verbesselt, R. Hyndman, G. Newnham, and D. Culvenor. Detecting trend and seasonal changes in satellite image time series. *Remote sensing of Environment*, 114(1):106–115, 2010.

[89] D. Weston, D. Hand, N. Adams, C. Whitrow, and P. Juszczak. Plastic card fraud detection using peer group analysis. *Advances in Data Analysis and Classification*, 2:45–62, 2008.

[90] Y. Zuo and R. Serfling. General notions of statistical depth function. *The Annals of Statistics*, 28(2):461–482, 2000.