Probabilistic Process Monitoring in Process-Aware Information Systems

by

Yihuang Kang

B.B.A., National Yunlin University, 2003

M.S., University of Pittsburgh, 2007

Submitted to the Graduate Faculty of

School of Information Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Information Science

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH

School of Information Sciences

This dissertation was presented

by

Yihuang Kang

2014

It was defended on

June 9, 2014

and approved by

Marek Druzdzel Ph.D, Associate Professor, School of Information Sciences

Stephen Hirtle Ph.D, Professor, School of Information Sciences

Subashan Perera Ph.D, Associate Professor, School of Medicine

Dissertation Advisor:

Vladimir Zadorozhny Ph.D, Associate Professor, School of Information Sciences

Probabilistic Process Monitoring in Process-Aware Information Systems

Yihuang Kang

University of Pittsburgh, 2014

ABSTRACT

Complex information systems generate large amount of event logs that represent the states of system dynamics. By monitoring these logs, we can learn the process models that describe the underlying business procedures, predict the future development of the systems, and check whether the process models match the expected ones. Most of the existing process monitoring techniques are derived from the workflow management systems used to cope with the logs generated by systems with deterministic outcomes. In this dissertation, however, I consider novel techniques that handle event log data, monitor system deviations, and infer the development of systems based on probabilistic process models. In particular, I present a novel process monitoring approach based on maximizing the information divergences of the system state dynamics and demonstrate its efficiency in detecting abrupt changes, as well as long-term system deviation. In addition, a new process modeling technique, Classification Tree hidden (semi-) Markov Model (CTHMM), is proposed. I show that CTHMM derived from Classification and Regression Tree and hidden semi-Markov model (HSMM) with hidden system states identified by Classification Tree can help discover and predict relevant system state sequences in temporal-probabilistic manners. The main contributions of this dissertation can be summarized as follows: 1) a new approach used in process monitoring that helps detect anomalies of dynamic systems from the point of views of both system change-point and long-term system deviation; 2) a unique HMM/HSMM learning technique that

solves the problem of hidden state splitting and estimates HMM/HSMM parameters simultaneously; 3) a novel temporal-probabilistic process model that generates human-comprehensible IF-THEN system state definitions used to help infer evolutions of discrete dynamic systems.

DEDICATION

To my parents

for teaching me to enjoy finding things out

To my wife

for encouraging me to always look on the bright sides of events in life

To my son

for bringing me new joy in life

ACKNOWLEDGEMENT

In the past years during my Ph.D. study, I have been struggling to find a balance among doing research, work, and family life. I am glad that I met many great people who have been helping me a lot with my life in Pittsburgh.

I would first like to express my appreciation to my dissertation committees, including Dr. Marek Druzdzel, Dr. Stephen Hirtle, Dr. Subashan Perera, and Dr. Vladimir Zadorozhny. This dissertation cannot be done without all of your help. I would like to say thank you to my advisor, Dr. Vladimir Zadorozhny, who patiently encourages me when I bog down in disappointing research results, and teaches me how to explain ideas in concise and easy-to-understand ways. I would like to say thank you to Dr. Michael Spring, who always gives me advice about life as a Ph.D. student in the U.S. and is always in his office to help students. I would like to say thank you to Dr. Druzdzel and Dr. Hirtle for their advice about doing research and the structure of my dissertation. I would like to say thank you to my supervisor in the Division of Geriatric Medicine, Dr. Subashan Perera, who always helps me when I am in need and kindly direct me to useful "models" not only those used to solve challenging data analysis problems in work but also those solutions when I faced the uncertainty of life.

In addition, I would like to say thank you to colleagues, doctors, and faculty members in School of Medicine, School of Public Health, School of Pharmacy, and the University of Pittsburgh Medical Center, especially Dr. Zachary Marcum, Dr. Susan Hardy, Dr. Carolyn Timberlake, Dr. Howard Degenholtz, and Dr. Joseph Hanlon. It is my pleasure to work with you all!

Last but not least, I want to express my deepest appreciation to my family. My parents are always my strongest support. They did not have chance to receive better education in their early life, but they are always trying to give their children the best. Also, I want to say thank you to my wife, the wisest woman I have ever met. She always reminds me to be flexible, open-minded, and willing to help others in need and embrace the changes in life.

Again, I would like to say thank you to my mentors, friends, colleagues, and family. I really enjoy life in Pittsburgh and this long journey to my Ph.D.!

TABLE OF CONTENTS

1.0 INTRODUCTION 15

1.1 MOTIVATION 15

1.2 RESEARCH QUESTIONS 17

1.3 OVERVIEW OF DISSERTATION 19

2.0 BACKGROUND AND RELATED WORKS 20

2.1 PROCESS MODEL AND PROCESS-AWARE INFORMATION SYSTEM 20

2.2 MARKOV MODELS AND THE DISCRETE DYNAMIC SYSTEM 23

2.3 DATA DISCRETIZATION AND TEMPORAL SEQUENCE GENERATION 27

2.4 CONCLUSION 32

3.0 PROCESS MONITORING USING MAXIMUM SEQUENCE DIVERGENCE 34

3.1 SEQUENCE DIVERGENCE MEASUREMENT 34

3.1.1 The Estimation of Symbol Probability Distribution 35

3.1.2 Measure Selection and the Generalized Jensen-Shannon Divergence 38

3.1.3 Significant Threshold and the Deviation Assessment 42

3.1.4 Stationary Sequence Probability Vectors and Higher-Order Markov Chain 49

3.2 APPLICATIONS, COMPARISONS, AND EXPERIMENTS 51

3.2.1 Sequence Similarity/Distance Measures and the Comparative Evaluation 52

3.2.2 Change-Point Detection 56

3.2.3 Deviation Detection 61

3.2.4 Using Stationary Sequence Probability Vectors 65

3.3 CONCLUSION 69

4.0 PROCESS MODELING USING CLASSIFICATION TREE HIDDEN MARKOV MODEL 71

4.1 BUILDING CLASSIFICATION TREE HIDDEN MARKOV MODELS (CTHMM) 71

4.2 MODEL SELECTION AND EVALUATION 83
4.2.1 Hidden State Splitting and Maximum Mutual Information Estimation 84
4.2.2 Accuracy of Viterbi Path Prediction 91
4.3 EXTENDING CTHMM WITH VARIABLE STATE DURATIONS- CLASSIFICATION TREE HIDDEN SEMI-MARKOV MODEL (CTHSMM) 95
4.4 EXPERIMENTS 99
4.4.1 Prediction of Most Probable Patient's Vital Sign State Transitions using CTHMM 103
4.4.2 Prediction with Variable State Duration using CTHSMM 108
4.5 CONCLUSION 113
5.0 SUMMARY 114
5.1 LIMITATIONS 114
5.2 DISCUSSIONS 115
BIBLIOGRAPHY 118

LIST OF TABLES

Table 1: Markov Models 25

Table 2: Markov Model tasks and applications 26

Table 3: Sequence Similarity Measures 54

Table 4: Weather system data without state splitting rules 74

Table 5: An HMM with 2 states and 3 observations 74

Table 6: An HMM with 3 states and 3 observations 75

Table 7: Weather data with 3 categorical variables 79

Table 8: State splitting rules and created observations matrix 84

Table 9: Weather data with mapped state number 84

Table 10: Confusion matrix 87

Table 11: Confusion matrix for the weather data 88

Table 12: Two confusion matrices with the same misclassification rate 89

Table 13: Cohen's Kappa and its interpretation 89

Table 14: A state transition matrix with one more different splits 91

Table 15: A observation matrix with one more different splits 92

Table 16: The presumable states and predicted state for testing data 95

Table 17: Weather data with state numbers and durations in hour 100

Table 18: An excerpt of CHP dataset 104

Table 19: Reference normal vital sign ranges for children aged 1-6 105

Table 20: Observation matrix with state definition rules for CTHMM$_{max\ MIo}$ 110

LIST OF FIGURES

Figure 1: Online Shopping Process Model in Petri net 21

Figure 2: Online Shopping Process Model as State Diagram 22

Figure 3: Model with Discrete Dynamic System 23

Figure 4: Markov Chain for the Temperature Feeling Sequence 24

Figure 5: A weather system HMM with parameters 26

Figure 6: Data discretization by SAX 29

Figure 7: Data discretization for weather measurement 30

Figure 8: Pattern Definition Rules for Classification Tree and SVM 31

Figure 9: Sequences and Symbol Transition Matrices 35

Figure 10: Conversion from Sequences to Steady-State Vectors 37

Figure 11: A Series of Steady-State Vectors with Weights 41

Figure 12: D_{GJS} vs. Number of Different State (k) 43

Figure 13: D_{GJS} vs. Number of Probability Distributions (m) 44

Figure 14: A series of steady-state vectors with the significant thresholds 47

Figure 15: Process of system deviation assessment using D_{GJS} 49

Figure 16: From sequences to SSV and n-order SSeqV 50

Figure 17: Pairwise sequence distances 53

Figure 18: Sum of pairwise sequence distances (given m = 3) 53

Figure 19: First 3,600 data points from DC dataset 57

Figure 20: D_{GJS} and the thresholds with different α 58

Figure 21: The AUCs of distance measures for DC dataset 59

Figure 22: First 10,000 data points of JM dataset 60

Figure 23: The AUCs of distance measures for JM dataset 61

Figure 24: CAF_BIH power consumption data 62
Figure 25: Sequences with sliding windows (m = 3) for CAF_BIH 63
Figure 26: AUCs with different parameters for CAF_BIH data 64
Figure 27: AUCs for measures with up to 3-order SSeqV for DC Dataset 67
Figure 28: AUCs for measures with up to 3-order SSeqV for JM Dataset 68
Figure 29: First 400 data points of STB dataset 69
Figure 30: AUCs for measures with up to 3-order SSeqV for STB Dataset 70
Figure 31: A tree splitting/growing example 78
Figure 32: Weather system classification tree with one split 80
Figure 33: The process of post-pruning a tree 82
Figure 34: A tree provides state splitting rules to divide data into 3 states 84
Figure 35: The tree with one more state to divide data into 4 states 85
Figure 36: V-fold cross-validation misclassification rate 90
Figure 37: Actual and Predicted States with Average Hit Ratio and LMRL Ratio for Table 16 96
Figure 38: Proposed CTHMM model selection and evaluation process 98
Figure 39: State duration distribution with pii= 0.8 100
Figure 40: The estimation of state transition probability of HSMM 101
Figure 41: Hit and LMRL Ratios for Variable Lengths of Actual and Predicted States 103
Figure 42: Tree models generated after post-pruning process 106
Figure 43: Maximum Mutual Information tree models 107
Figure 44: Average Hit and LMRL Ratio with different lengths of prediction 109
Figure 45: Most probable state sequences given different lengths and observations for Scenario 1 111
Figure 46: Most probable state sequences given different lengths and observations for Scenario 2 112
Figure 47: Most probable state sequences given different lengths and observations for Scenario 3 113
Figure 48: Average Hit and LMRL Ratio with different hours of prediction 114

Figure 49: State Duration Distributions of CTHSMM$_{max\ Mio}$ 115

Figure 50: Most probable state sequences given different patients' location within 18 hours for Scenario 1 116

Figure 51: Most probable state sequences given different patients' location within 18 hours for Scenario 2 117

Figure 52: Most probable state sequences given different patients' location and times for Scenario 3 ... 117

1.0 INTRODUCTION

1.1 MOTIVATION

Information systems generate huge amounts of operational event logs used in monitoring the conditions/states of dynamic systems. For example, Bedside Medical Device Interface provides a set of tools that can automatically record all of the information from devices at the patient's bedside on Intensive Care Unit (ICU) monitors; a disease outbreak detection system records the number of outpatient visits for some particular diseases; and a credit card fraud detection system monitors the suspicious credit activities. Most of these logs, however, are rarely created for further business process analyses, such as underlying procedure discovery and process auditing. These event logs contain patterns of interest that can be identified by the domain experts, discovered by pattern classification methods, or represented in meaningful symbols. By tracking the dynamics of these patterns over time, we can understand the system dynamics and how the system evolves, which motivates me to explore and develop the process monitoring techniques.

I define the process as *a series of activities or the state transitions of a dynamic system that produces some specific, either deterministic or probabilistic, outcomes*. Here, the tasks of *process monitoring* refer to a series of actions that collect the activities of instances (e.g. patients) or changes of states from the event logs of information systems; use the logs to build models that best describe the patterns; track the development of the systems by monitoring the changes of patterns; and evaluate the conformance of the development with expected models. Many approaches related to the process monitoring have been proposed in different fields, such as conformance check in process mining (Van Der Aalst, 2011), anomaly detection (Chandola, Banerjee, & Kumar, 2009), and statistical quality control(Montgomery, 2008). The process mining investigates how to use

event logs of multiple instances from information systems to discover the underlying business process models represented as the workflows and check whether logs conform to expected process models; the anomaly detection particularly focuses on finding "the patterns in data that do not conform to a well-defined notion of normal behaviors" (Chandola et al., 2009); and the statistical quality control concerns the controls of the processes by identifying the source of process/product variations.

I am particularly interested in developing process modeling and monitoring techniques that work on temporal sequence data. Due to the fact that the majority of the data generated by information systems are often numeric time-series, many existing approaches in these fields aim to directly cope with these series data streams. For example, *Box-Jenkins'* approach (Hanke & Wichern, 2005) is used to fit the forecasting models with past numeric values. On the other hand, the *Shewhart Control Chart* (Basseville & Nikiforov, 1993), Cumulative SUM (CUSUM) (Basseville & Nikiforov, 1993), and the *Generalized Likelihood Ratio* (Willsky & Jones, 1976) focus on detecting the anomalies/changes of systems by comparing models created in different times. These approaches and their extensions have been actively discussed in data mining communities for decades and are widely used in many real-world applications. However, the explosion of the data dimensions and numerosities in the recent years impedes the performance of these approaches, because the processing of high dimensionality/numerosity data requires more computational power as the amount of data grows. Many researchers have started to consider the data dimensionality and numerosity reduction using the pattern classification methods (Fodor, 2002) and time series representation techniques. The goal of these methods is to discretize the continuous features, keep the signatures (e.g. the distance measure) of original data in the transformed space, and adapt the data into patterns of interest (Daw, Finney, & Tracy, 2003a).

These pattern dynamics can be regarded as the development of the system and denoted by meaningful symbols—the temporal sequences. Consider a simple temperature classification. We discretize the temperature in degrees centigrade C, ($15 < C, 15 \leq C < 25, 25 \leq C$) into (***C***old, ***W***arm, and ***H***ot). Two temporal sequences like (***C***,***C***,***W***,***C***,***W***,***W***,***H***,***H***) and (***C***,***C***,***W***,***W***,***W***,***W***,***H***,***H***) reflect the transitions of the weather condition in terms of the feeling of the temperature in different periods. I am interested in discovering patterns by analyzing these temporal sequences.

Besides, unlike most of process monitoring approaches that construct models characterizing the sequential orders of underlying deterministic procedures of the system (e.g. workflows), I consider building process models from the point of view of system state transitions by using these temporal sequence data stream converted from the event logs of the information systems. In this dissertation, I propose two novel techniques that detect anomalies and infer developments of the system in temporal-probabilistic manners.

1.2 RESEARCH QUESTIONS

In this dissertation, I present my process monitoring approaches, evaluate the performances, and discuss the limitations with possible improvements. The proposed approaches aim at answering the following 5 research questions:

- *RQ1: Given a simple univariate temporal sequence as process log generated from an information system, how to detect the anomalies of the system from this temporal sequences? (Section 3)*

- *RQ2: Stationary state probability vector generated from first-order Markov chain is used in proposed process monitoring technique. Would using stationary sequence*

probability vector generated from higher-order Markov chain improve the accuracy of sequence anomaly detection? (Section 3)

- *RQ3: Most temporal sequence data generated from information systems are used to create deterministic process models. How to make use of these sequences to predict the dynamics of systems in probabilistic manners? (Section 4)*

- *RQ4: The hidden Markov model (HMM) is used here to create probabilistic process models. Most applications of HMM require well-defined state definitions or use pattern (state) identification techniques to create models with vague state definitions. How to define human-comprehensible state definition and determine appropriate number of states? (Section 4)*

- *RQ5: Typical HMM assume hidden state sojourning times are fixed, which implies that the hidden state durations are geometrically distributed and thus impractical. Can we extend proposed approach, Classification Tree Hidden Markov Model (CTHMM), by taking different hidden state durations into consideration? How does the extended model improve the prediction? (Section 4)*

RQ1 and RQ2 are addressed by proposed process monitoring technique in Section 3, where I present how to detect sequence anomalies by monitoring the changes of information divergences computed from stationary system states or state sequence probability vectors for different orders of Markov chain (Section 3.1). RQ3, RQ4, and RQ5, on the other hand, are answered by proposed

probabilistic process model, Classification Tree Hidden Markov Model (CTHMM), which is introduced in Section 4. I show that CTHMM can help determine appropriate number of system states, infer the changes of system states with IF-THEN state definitions, and predict most relevant system dynamics in temporal-probabilistic manners.

1.3 OVERVIEW OF DISSERTATION

The rest of this dissertation is organized as follows. In Section 2, I begin with reviewing current process modeling techniques, Markov models with their applications, and data reduction methods related to proposed approaches. I present my process monitoring approach in Section 3, which provides a new way of anomaly detections when we have a simple temporal sequence. In Section 4, I consider a more complicated situation that we cope with how to convert multivariate data stream into concurrent state-observation temporal sequences used to create process models. I introduce my approach that builds probabilistic process models to predict system dynamics with comprehensible system state definitions. The experimental results and evaluations are shown in the ends of both Section 3 and Section 4. In Section 5, I conclude with the discussion about how proposed approaches address the research questions.

2.0 BACKGROUND AND RELATED WORKS

In this section, I review the background knowledge and techniques related to my proposed approaches. I begin with the brief history of the deterministic process models and state transition diagrams. Then, the discrete event systems represented as Markov models used in my approaches are also introduced. As I particularly focus on temporal sequence analysis, the data reduction and discretization techniques applied to the generations of sequence data are also discussed here.

2.1 PROCESS MODEL AND PROCESS-AWARE INFORMATION SYSTEM

The concept of process monitoring arise from the trend of information system developments shifting from data orientation to process orientation (Dumas, Van der Aalst, & Ter Hofstede, 2005). In the 1980s, most of information systems are only designed to process and store the data. The business processes/procedures behind the scenes are often neglected. Later in the 1990s after the rise of business process reengineering, the attention of information system development was drawn to workflow management (van Hee, 2004), process-aware groupware (Dumas et al., 2005), and business process modeling (Weske, 2012). This transition led to the development of *process-aware information systems* (Dumas et al., 2005) that can automatically capture, store, manipulate, monitor, and present the information of business activities they support.

Process monitoring techniques are the products of this information system evolution. They are used to create abstract models that best describe the underlying business procedures. These process models are often represented by using graphical notation language, such as *Petri Net*, *Business Process Management and Notation* (BPMN), and *UML Activity Diagram*. Consider a simple online shopping process model represented as Petri net as shown in Figure 1.

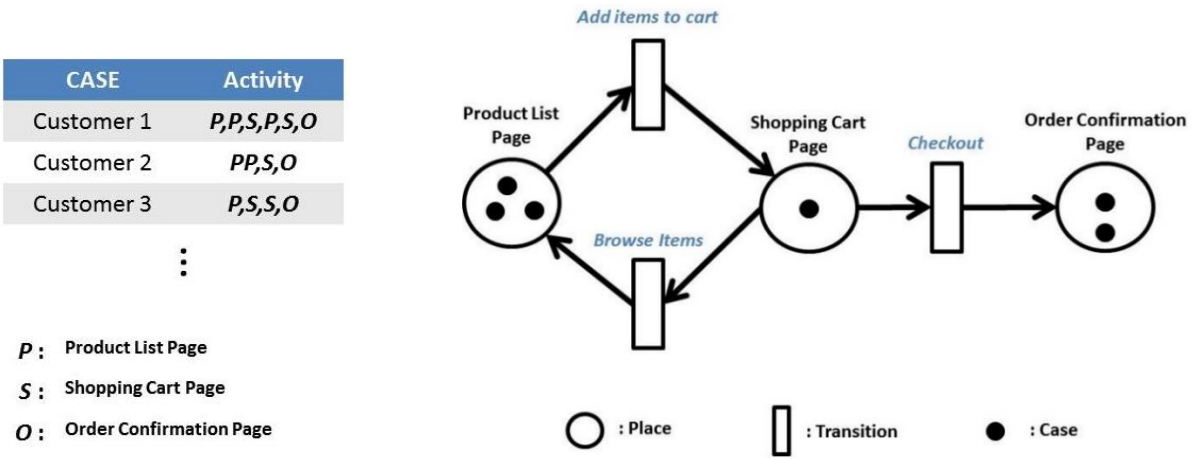


Figure 1: Online Shopping Process Model in Petri net

Note that I assume there are 3 *Places* and *Transitions* in this process model. The Place is the location of a *Case* (customer), whereas the Transition is the action a case can take moving to next Place. There may be multiple Cases in a Place. This process model is built based on the transitions (between two Places) of the Case activities in the left event log table.

Apparently, to build a process model, we need existing well-defined Places and Transitions that balances the completeness and complexity of the model. An over-specific process model with too many Places and Transitions may characterize too much detail of business activities and thus may not be generalized well as a reference model to analyze the process. On the other hand, an over-simplified process model with too few Places and Transitions may not capture essential procedures that could help understand the business activities and further improve the process. Also, these deterministic modeling techniques like Petri Net and BPMN are designed to build process models from the sequential and case-dependent event data. These techniques do not take into account the time spending in Places and the probabilistic nature of the Transitions of Cases. In the recent decades, several approaches, such as *State Transition Diagram* of UML and *Stochastic Petri*

Net (Haas & Haas, 2002), are proposed to solve these issues and improve the process models so that the models can be used in quantitative analyses and probabilistic inferences. Most of these techniques assume the system we monitor is a *discrete dynamic system* (Cassandras & Lafortune, 2008), which defines that the target system can be represented by the state transitions and may evolves with time. Figure 2 shows an example of the process model in Figure 1 represented as a state transition diagram.

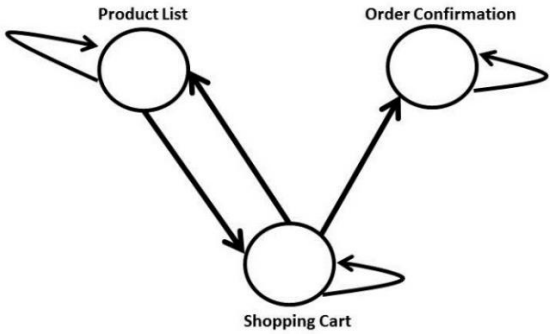


Figure 2: Online Shopping Process Model as State Diagram

Each state can transit to itself and to other states. For example, a customer could keep viewing the Product List pages for a period and the system is therefore staying in the state Product List. On the other hand, a customer could switch between state Product List and Shopping Cart until he or she decides to place the order (move to state Order Confirmation). As these state transitions could be probabilistic, we can model them using probabilistic modeling techniques, such as *Markov Models* and *Dynamic Bayesian Networks* of *Probabilistic Graphical Models* (Koller & Friedman, 2009). In this dissertation, I consider different types of Markov models commonly used in modeling discrete dynamic system to create my probabilistic process monitoring approaches.

2.2 MARKOV MODELS AND THE DISCRETE DYNAMIC SYSTEM

The aforementioned discrete dynamic system refers to the system that evolves with time and changes its state at discrete points in time (Cassandras & Lafortune, 2008). Consider an information system that monitors this type of system and keeps yielding temporal data stream with a set of variables. These variables are partial reflections of the phenomena we would like to investigate, as there are some latent variables/features we cannot directly measure/observe. We can build state-space process models for this system with finite states. Figure 3 shows that the estimator can build models from two different sequence data stream—"observable states only" and "observable states with estimated hidden states".

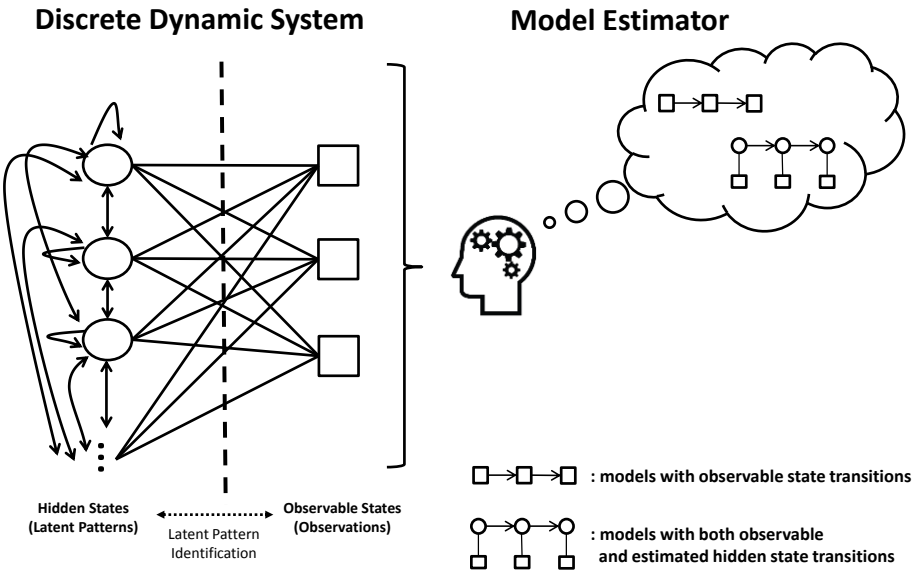
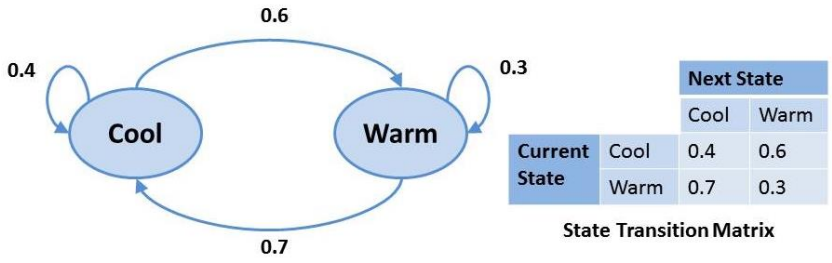


Figure 3: Model with Discrete Dynamic System

Modeling with observable states is literally to consider only sequences of pre-defined observable state transitions (e.g. the sequences of temperature transitions in Section 1). On the other hand, if we assume that those observable states depend on some underlying patterns/states or the

combinations of other latent variables, we can identify/estimate these hidden patterns by using supervised pattern classification techniques (Duda, Hart, & Stork, 2001) and further build the models from both observable and hidden state transitions.

Markov models are the simplest probabilistic models to cope with these temporal sequence data. They assume that the system is stochastic and with *Markov property*, which means the development of the dynamic system is assumed to be a random process that the current state of the system has the information about the next states. But, the next states only depend on the present state and are independent of past states. All the state transitions are probabilistic. Consider previous temperature feeling sequences but with only 2 states, Cool and Warm, for simplicity. By assuming the system that generates the sequences has Markov property, we can create a specialization of Markov models called *Markov Chain* in the following finite state machine chart with a state transition matrix (i.e. a right stochastic matrix) shown in Figure 4.



		Next State	
		Cool	Warm
Current State	Cool	0.4	0.6
	Warm	0.7	0.3

State Transition Matrix

Figure 4: Markov Chain for the Temperature Feeling Sequence

The figure shows that the system can stay in a state (Cool or Warm) or transit to other state with different probabilities as listed in the state transition matrix. There are various kinds of specializations of Markov models. Table 1 show the most common Markov models in terms of whether we have control over the state transition and whether the states are completely observable as defined by Littman (Littman, 1996)

Table 1: Markov Models

Markov Models		Have control over the state transitions?	
		Yes	No
States completely observable?	Yes	Markov Decision Process (**MDP**)	Markov Chain (**MC**)
	No	Partially observable Markov Decision Process (**POMDP**)	Hidden Markov Model (**HMM**)

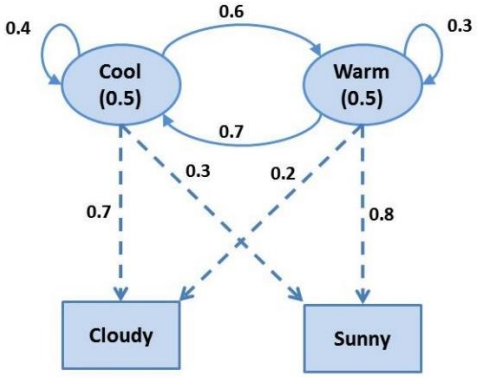
Taking the previous weather temperature system in Figure 4 as the example, we called it Markov chain because we assume that we do not have control over the temperature of the weather system but we can completely observe the temperature and its changes (transitions).

Based on Table 1, we can define a *discrete generalized Markov model* (***S***, ***A***, ***O***, ***R***, ***D***) (Cassandras & Lafortune, 2008) that consists of a set of states ***S***; a set of actions ***A***; a set of observations ***O***; a set of rewards for each state-action pair ***R***; state duration distributions ***D***; and model parameters ***(IM, OM, STM, RM)***—Initial Matrix (***IM***), Observation Matrix (***OM***), State Transition Matrix (***STM***), and Reward Matrix (***RM***). A discrete generalized Markov model is a temporal multivariate model with Markov property. It investigates the connection between hidden states (states) and observable states (observations) and the role of states and observations in the development of a dynamic system. Table 2 lists common tasks and applications for the aforementioned Markov models.

Table 2: Markov Model tasks and applications

Markov Models	Model	Features and Tasks	Application
Markov Chain	*(S, D)*	Steady-state analysis	Page Ranking
Hidden Markov Model	*(S, O, D)*	Evaluation: Forward Algorithm Decoding: Viterbi Algorithm Learning: Baum-Welch Algorithm	Speech Recognition
Markov Decision Processes	*(S, A, R, D)*	Optimal Policy (Optimal Value Function)	Travel route planning
Partially-Observable MDP	*(S, A, O, R, D)*	Belief state update Find approximate solutions	Robot traveling

Let's take a look a more complicated example of the weather system represented as a *Hidden Markov Model* (***S***, ***O***, ***D***) (HMM) as shown in Figure 5. The HMM of the weather system consists of the temperature as states (Cool, Warm) and weather conditions as observations (Cloud, Sunny). Here, I consider a typical HMM that the state duration distributions (***D***) are fixed, which means the durations of states sojourning/staying in themselves are identical.



		Prior Probability
State	Cool	0.5
	Warm	0.5

Initial Matrix

		Next State	
		Cool	Warm
Current State	Cool	0.4	0.6
	Warm	0.7	0.3

State Transition Matrix

		Observation	
		Cloudy	Sunny
State	Cool	0.7	0.3
	Warm	0.2	0.8

Observation Matrix

Figure 5: A weather system HMM with parameters

In Figure 5, I assume that the prior probability of each state (state probabilities we believed or pre-estimated) is 0.5 to create the Initial Matrix. The observations can be considered the outcome of the weather system, and we believe that there is a certain connection between states and observations, which means the observations (weather condition) depend on the states (temperature). The parameters of this HMM can be used to infer the possible dynamics of the weather system, such as the prediction of most probable state sequence give a specific observation sequence, which could be solved by *Viterbi Algorithm* (Forney, 1973). In this dissertation, I present my probabilistic process models based on Markov chain and hidden Markov model. The detail implementations will be elaborated in Section 3 and 4.

2.3 DATA DISCRETIZATION AND TEMPORAL SEQUENCE GENERATION

In this dissertation, I introduce my implementations of process monitoring techniques that particularly focus on dealing with the discrete temporal sequence data stream. As mentioned in Section 1, most of data generated by information systems are high-numerosity multivariate time-series instead of temporal sequences we need. Many existing data reduction and pattern classification methods can be used to convert these time-series data into temporal sequence data. For example, the *Discrete Fourier Transform* (Faloutsos, Ranganathan, & Manolopoulos, 1994), *Piecewise Aggregate Approximation* (PAA) (E. J. Keogh & Pazzani, 2000), *Shape Definition Language* (SDL) (Psaila & Wimmers Mohamed &It, n.d.), *Symbolic Aggregate approXimation* (SAX) (E. Keogh, Lin, & Fu, 2005; Jessica Lin, Keogh, Wei, & Lonardi, 2007), are proposed to reduce the data numerosity. Also, in the data mining communities, pattern classification methods, such as *Singular Value Decomposition* (E. Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001) and *Principal Component Analysis* (Jolliffe, 2002) and even the pattern definition induction of

Classification and Regression Tree (CART) (Breiman, Friedman, Olshen, & Stone, 1984), can be used to reduce the data dimensions. Here, I focus on those data reduction methods that discretize the raw data into sequences of symbols, as the benefits of analyzing the symbolic data stream are both the numerosity/dimensionality reduction and the measurement noise-insensitivity (Daw, Finney, & Tracy, 2003). Also, numerous sophisticated sequence analysis methods, such as *Permutation*, *Bernoulli*, and Markov models (Robin, Rodolphe, & Schbath, 2005), can be used to efficiently manipulate and perform the analysis on the symbolic data stream (J. Lin, Keogh, Lonardi, & Chiu, 2003). Besides, to preserve the essential information in the original series data, the data reduction method must also be able to keep the signatures (e.g. the distance measure) of the original data in transformed data space as discussed in the Section 1. That is, the distances among these transformed data stream are guaranteed to be similar to the distances in the original space. This property of a discretization method is also called *lower bounding* (Shieh & Keogh, 2008). In the later experiments, I used SAX to symbolize the time-series into sequences, as SAX is with the property. On the other hand, for the time-series data with a class label variable as the outcome, I used CART to learn the IF-THEN rules from the classification tree inductions to identify the hidden patterns/states as sequences. The detailed applications of both approaches are illustrated in Section 3 and Section 4.

The SAX is a method that discretizes a univariate real-valued time series and produces symbols with approximately equal probabilities. The time-series data is divided into *i* segments of equal length. Given that a normalized time series data has a Gaussian distribution, the distribution is divided into *k* equally-probable areas that are assigned *k* possible symbols. Each equal-length segment in the data is replaced with a symbol based on which area the average value of the segment is in. Figure 6 shows an example of how SAX reduces the data numerosity.

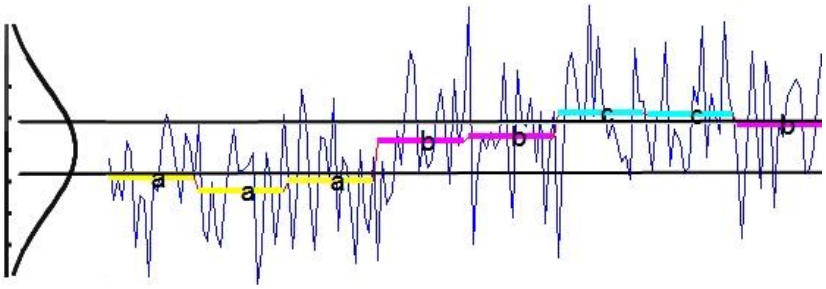


Figure 6: Data discretization by SAX

In Figure 6, the series is divided into 8 segments of equal length. The distribution of the series is divided into 3 equally-probably areas that are represented by 3 letters (***a***, ***b***, ***c***). Each segment is assigned a letter/symbol (***a***, ***b***, or ***c***) based on where its average is located. The series data in Figure 6 is then converted into ***aaabbccb***. One of major applications of SAX is to discover the time series discords (E. Keogh et al., 2005), which is to find the unusual patterns/subsequences within a time series. I use SAX in the later experiments because it is of the abovementioned advantages—numerosity reduction and lower bounding (E. Keogh et al., 2005; Jessica Lin et al., 2007). Unlike the application of discord discovery of SAX, my proposed process monitoring approach in Section 3 uses the sequence of symbols/letters generated by SAX to detect the significant changes of the symbol probability distributions. That is, I am interested in finding the deviation of a dynamic system by monitoring these sequences, not the specific abnormal patterns within these sequences.

Consider another data discretization example for a multivariate time-series data with a discrete class variable. Assuming that the class variables as dependent variable (outcome) and the rest of variables as independent variables (predictors), many existing supervised pattern classification methods, such as CART, *Support Vector Machine* (SVM) (Drucker, Burges, Kaufman, Smola, & Vapnik, 1996), and *Learning Vector Quantization Network* (LVQ) of artificial neural networks (Somervuo & Kohonen, 1999), can be used to generate sequences of patterns

(symbols). Figure 7 shows an example of data discretization for a weather measurement data with temperature time-series and the corresponding weather conditions.

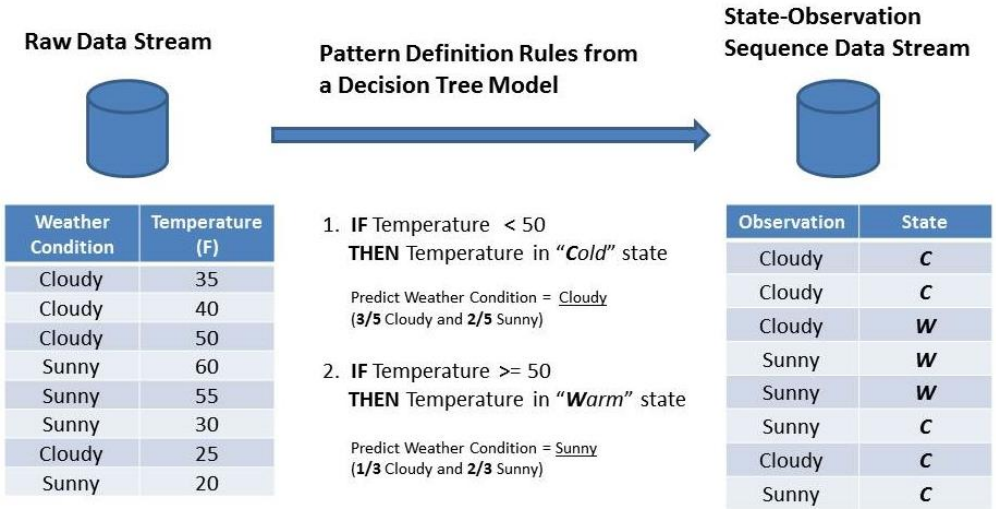


Figure 7: Data discretization for weather measurement

A tree-based pattern classification method like CART can learn the IF-THEN pattern definition rules that transform the predictor (temperature) into temporal state sequence as (***C***, ***C***, ***W***, ***W***, ***W***, ***C***, ***C***). Unlike the common uses of these pattern classification methods that learn classification models and then predict the target/outcome class of given testing datasets, the models are used to extract the rules/definitions between predictors and outcome that outcome is described as linear or non-linear combinations of predictors. In Figure 7, two IF-THEN rules are used to not only predict the weather condition but also define the areas of two patterns in data space. Two patterns are then called "**C**old" and "**W**arm" in Figure 7. These temporal state and observation (outcome) sequences could be used to further analyze the dynamics of the system we monitor.

Note that some other approaches, such as SVM and LVQ, may build classification models with lower misclassification rate and the same numbers of patterns (states) than the tree-based

approaches do. However, the rules of patterns generated by these approaches are often hard to interpret by humans. Consider another example of weather time-series data with temperature and atmospheric pressure as the predictors as shown in Figure 8.

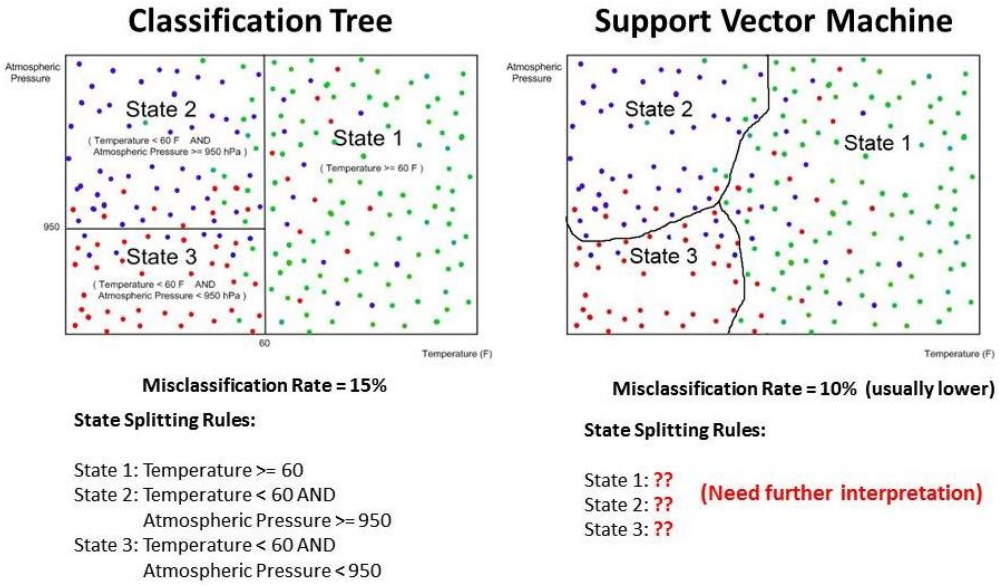


Figure 8: Pattern Definition Rules for Classification Tree and SVM

There are 3 different weather conditions as denoted by dots with 3 different colors. Both the tree and SVM models identify 3 states, which are 3 areas in data space. Apparently, the rules learn from tree-based model are easy to comprehend, whereas the rules from SVM are unclear and need further interpretations.

The SVM constructs a set of hyperplanes with the largest distances to data points in high dimensional space to separate data. Similar to SVM, trained artificial neural networks (e.g. LVQ) can also help distinguish data by using their adapted and weighted artificial neurons as separations in two or higher dimensional data space. One major reason that these two approaches often result in lower misclassification rates with the same number of patterns is that they do not create simple

vertical and horizontal lines in two dimensional space (as shown in Figure 8) or hyperplanes in higher dimensional spaces as tree-based approaches do. However, the rules from SVM and artificial neural networks are obscure because their classifiers are linear combinations of chosen linear or nonlinear kernel functions. They are not just simple IF-THEN rules as we have from tree-based models. Some work in literature, such as (Núñez, Angulo, & Català, 2002) and (Setiono & Liu, 1995), address this problem and provide possible solutions to extract simple rules from these approaches.

2.4 CONCLUSION

Process monitoring is not a new discipline but a derivative of various studies as discussed in this section. In this dissertation, the process is considered as a series of system dynamics—the state-observation sequences. Based on the aforementioned background knowledge and techniques, I propose two approaches that focus on the detection of the temporal sequence anomalies and the inference of future sequence patterns by tracking and modeling these temporal sequences.

Before the introduction of my process monitoring approach, let us begin with a simple research question about sequence anomaly detection. Assume that we have a monitoring information system that keeps generating symbolic data (sequences) that represents the conditions of the system we monitor (e.g. the temperature sequence in Section 1.1). Here, all information we have is just these sequences of symbols. How can we know that there is something wrong with the system? Specifically, we would like to know whether there is an abrupt change—a change point, and whether there is a latent gradual change—a long-term system deviation of the system.

The most intuitive way of discovering these anomalies is to compare sequences in different times, i.e. to compute the distances among sequences or the differences of the symbol occurrences.

However, there are two noticeable drawbacks of these approaches—1) these approaches do not take into consideration the differences of symbol orders/positions, which may contain crucial system dynamics that could result in both short-term and long-term system abnormality; 2) these approaches do not provide significant thresholds of the measures they use to signal the anomalies. By "significant threshold", I mean a critical point/value of the measure that indicates the emergence of anomalies. For example, a body temperature above 100 F (37.8 C) for an adult suggests that one may have fever and could take required actions if needed. The lack of the thresholds hinders the abovementioned approaches from real-world applications. Conversely, my proposed approach does not have these drawbacks. In the next section, I introduce my approach that combines Markov chain and Google PageRank (Langville & Meyer, 2006) algorithm with Generalized Jensen-Shannon Divergence (Grosse et al., 2002) as the distance measure, which will be proved to outperform other approaches/measures in the later experiments.

3.0 PROCESS MONITORING USING MAXIMUM SEQUENCE DIVERGENCE

In this section, I consider how to analyze simple univariate temporal sequences (e.g. temperature sequences) in order to learn the changes of a dynamic system. The anomalies of a system development can be identified by measuring the differences among sequences in different times. Here, I introduce my approach based on maximizing the information divergence.

3.1 SEQUENCE DIVERGENCE MEASUREMENT

Temporal sequences can be considered the snapshots of a dynamic system in different periods. As discussed in Section 2, the changes of the temporal sequences may reveal the evolution of the dynamic system we monitor and suggest whether the system deviates. Consider a simple process monitoring application, such as an information system that keeps generating sequences of symbols representing the current stock market index consisting of two possible symbols/letters of changes, namely **U** and **D** as the index goes "**U**p" and "**D**own" for simplicity. The sequence is empirically divided into 5 equally-sized subsequences as shown in Figure 9. These two symbols, **U** and **D**, are equally-probable in terms of relative frequencies in each sequence. At the very beginning (S_1), we can see that the index keeps iteratively up and down through the observation cycle. Then, the index becomes more stable. The goal is to detect the change of symbol dynamics—the *deviation of a system development.* As discussed in previous sections, many approaches have been proposed to discover the differences/changes among paired or multiple sequences. Here, I suggest finding the differences among the stationary symbol probability distributions generated from sequences.

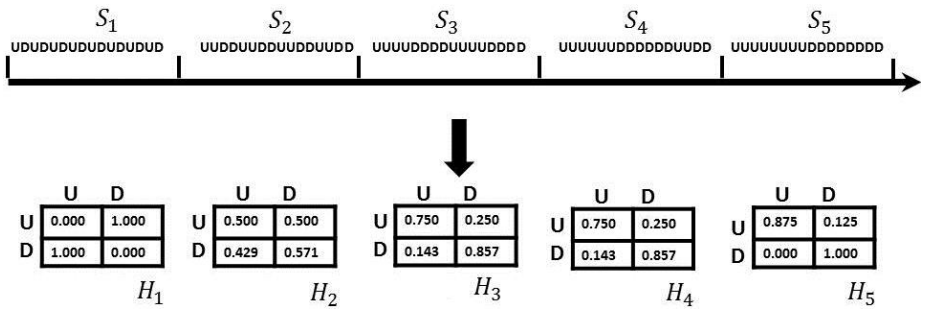


Figure 9: Sequences and Symbol Transition Matrices

3.1.1 The Estimation of Symbol Probability Distribution

To obtain the stationary symbol probability distributions, I first build the first-order discrete-time Markov Chain with a symbol probability transition matrix (denoted by $\mathbf{H}_m$ in Figure 9) for each sequence. Assuming the development of this system is a random process of Markov property, for each right stochastic matrix of a Markov Chain, we can obtain a unique stationary probability distribution, also called *Steady-State Vector* (SV), if the matrix is ergodic (Cassandras & Lafortune, 2008). That is, any state (symbol) can return to itself in one step and also can be reached from any other states in the stochastic matrix. In this case, the Markov chain we created is fully connected and each state transition has a non-zero probability. Evidently, there is no guarantee that we can create such matrices from all the sequences, as some state transitions may never occur within a time frame (sequence). Here, I consider constructing the stochastic matrix for each sequence and then convert these matrices into Google Matrices (Langville & Meyer, 2006). The Google Matrix is an ergodic and stochastic matrix originally used by Google's PageRank algorithm to deal with very large sparse matrices that represent the links between web pages. The Google Matrix **G** can be computed as:

$$\mathbf{G} = d\mathbf{H} + (d\mathbf{a} + (1 - \mathrm{d})\mathbf{e})\frac{1}{k}\boldsymbol{e}^T \qquad \text{(Eq. 1)}$$

where **H** is the original stochastic (symbol transition) matrix created from a sequence. The **a**, **e**, and *k* denote the binary dangling node vector, the rank-one teleportation vector, and the number of possible states/symbols respectively. And *d* is the damping factor that is between 0 and 1. Note that I use *d* instead of α found in most literature in order to be distinguished from the statistical significant level α used in the upcoming paragraphs. As an **G** is dense and fully connected, we can then obtain a unique SV, also called the *PageRank vector* (Langville & Meyer, 2006). Instead of being used to rank the pages, the generated PageRank vectors are considered the symbol probability distributions in the later experiments.

The SVs contain the fixed probabilities of each state (symbol) when a Markov chain operates for a sufficiently long period (Cassandras & Lafortune, 2008). Here, the continuously-created stochastic matrices and the SVs can be considered snapshots of the system transitions and the system development for long run. One advantage of considering the changes of the symbol probability distributions in SVs instead of these from the frequency of symbols in the sequences is that the SVs also take the orders of symbols (transitions) into consideration, which is valuable when SVs are used in the detection of abnormal transitions. Figure 10 shows how to create the SVs from these 5 sequences (S_1, S_2, S_3, S_4, S_5) Figure 9. I convert **H** into **G** with damping factor *d* = 0.99. For each **G**, we can obtain a unique SV. These 5 sequences are then transformed into a series of 5 SVs.

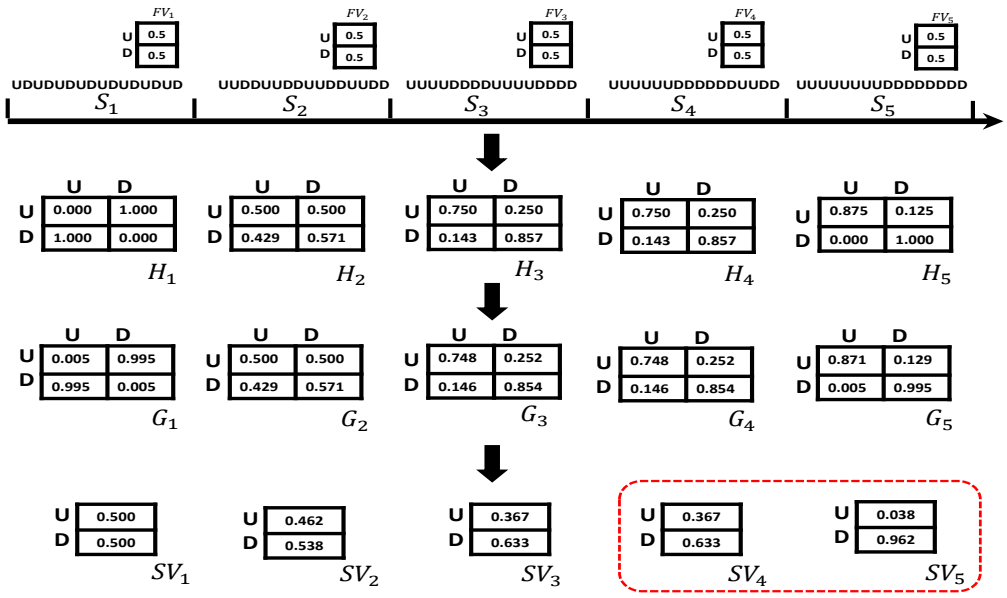


Figure 10: Conversion from Sequences to Steady-State Vectors

Note that the damping factor (d = 0.99) plays an important role here. It is originally used to control the rate that the random page surfers follow the hyperlink structures or jump to a random new page (Langville & Meyer, 2006). Here, the damping factor is considered the rate that moves the probabilities from those high-probable to lower-probable (or absent) symbols transitions. The original damping factor used in the PageRank algorithm is 0.85, which balances the efficiency and the effectiveness of performing the power method to obtain the SV (Langville & Meyer, 2006). However, the choice of damping factor in my approach depends on how well the sequences we analyze reflect the actual dynamics of a system in different monitoring periods. That is, we use a lower damping factor if we believe those low- or zero- probable cells of a stochastic matrix should be higher, because the sequence we use may not represent the actual transitions of the system conditions. In most cases, I suggest using a high damping factor instead to avoid padding too high probabilities into these low- or zero-probable cells of a stochastic matrix so that we can maximize the differences among these SVs generated from different sequences, as a higher damping factor

increases the sensitivity of the resulting vectors that are able to detect the smaller changes of the system (Langville & Meyer, 2006). Here, I use $d = 0.99$ for all the later experiments. Also, as the size of the stochastic matrix in the proposed approach is determined by the number of possible states/symbols and is usually much smaller than the page link matrix (e.g. a 2 by 2 stochastic matrix in Figure 10), the high damping factor with small matrix does not requires significant computation time to obtain the steady-state vectors.

In Figure 10, we can see that the SVs change in terms of the probabilities of symbol (**U** and **D)** and show a trend that the market index is going down at the end of the monitoring period. The approach to use the PageRank vectors, instead of relative frequency vectors (FV), captures the possible different long-term symbol transitions and maximizes the deviation of system development in different time frames. Figure 10 also shows that there is not only a gradual deviation, but also a noticeable change between SV_4 and SV_5. The goal of process monitoring is to be able to detect both of them.

3.1.2 Measure Selection and the Generalized Jensen-Shannon Divergence

By monitoring the *Information Divergence* among the discrete probability distributions of these SVs, we are able to assess the deviation of the system. The "Information Divergence" here is the notion of distance that indicates the difference among two or more symbol probability distributions. Most divergence measures do not satisfy the strict conditions as a true distance metric in mathematics, i.e. the *symmetry* and *triangle inequality*, which means these measures should not be used as a regular distance metric to compare arithmetically. To select an appropriate measure, I define the first two requirements of a distance/divergence $D(P_x,P_y)$ we need:

$$D(P_x, P_y) \geq 0 \qquad \text{(Eq. 2)}$$

$$D(P_x, P_y) = 0, \text{iif } P_x = P_y \qquad \text{(Eq. 3)}$$

where P is a discrete symbol probability distribution vector (e.g. the SV in our approach). $P = [p_1, p_2, \ldots, p_k]$ and $\sum_k p_k = 1$. At first glance, we can just use a divergence that meets the Eq. 2 and Eq. 3 in our monitoring system. However, as discussed, there are some popular divergences widely used in various fields, but not all of them are the appropriate deviation measures for our purpose. Take the Kullback-Leibler Divergence (D_{KL}), also known as *relative entropy*, as an example. The divergence is defined as:

$$D_{KL}(P_x, P_y) = \sum_{i=1}^{k} P_x(i) \log_k \frac{P_x(i)}{P_y(i)} \qquad \text{(Eq. 4)}$$

where the base k is the number of discrete probabilities (the number of components in an SV). If we have two SVs, P_1=[0.5 0.5] and P_2=[0.9 0.1], for example, the $D_{KL}(P_1, P_2) = 0.737$. It appears D_{KL} is an applicable measure, but an asymmetric divergence like D_{KL} cannot provide a common metric to evaluate the same set but different permutation of SVs. D_{KL} is proved to be asymmetric and semi-bounded (Jianhua Lin, 1991), which means:

$$D_{KL}(P_x, P_y) \neq D_{KL}(P_y, P_x) \qquad \text{(Eq. 5)}$$

$$0 \leq \mathrm{D}_{KL}(P_x, P_y) \leq +\infty \qquad \text{(Eq. 6)}$$

For the previous example with two SVs, the $D_{KL}(P_1, P_2)$ is 0.737, but $D_{KL}(P_2, P_1)$ is 0.531. Also, there is no maximum limit of D_{KL} for any given two probability distributions. That is, for example,

$$D_{KL}([0.9, 0.1], [0.1, 0.9]) = 2.5359$$

$$D_{KL}([9 * 10^{-10}, 1 * 10^{-10}], [1 * 10^{-10}, 9 * 10^{-10}]) = 33.2193$$

…

$$D_{KL}([9*10^{-n}, 1*10^{-n}], [1*10^{-n}, 9*10^{-n}]) \to \infty, \text{where } n \to \infty$$

These two properties (i.e. Eq. 5 and Eq. 6) make D_{KL} an inappropriate measure for our purpose. Again, the goal is to assess the significance of deviation for a system by monitoring a divergence measure from these continuously-created SVs. Here, I restate the two required properties of the divergence. A divergence $D(P_x, P_y)$ we need must be *Bounded* and *Symmetric*, which means it must not only satisfy Eq. 2 but also Eq. 7 and Eq. 8 as follows:

$$0 \leq D(P_x, P_y) \leq \mathrm{a}, \mathrm{a} \in \mathrm{Q}^+ \qquad \text{(Eq. 7)}$$

$$D(P_x, P_y) = D(P_y, P_x) \qquad \text{(Eq. 8)}$$

A bounded divergence measure provides certain limits of the deviation that simplify the magnitude evaluation when it is used in numerical applications, whereas a symmetric divergence ensures the identity of the deviation for the same set of symbol probability distributions, which means different permutations of the same set of probability distributions must have the identical deviation. Note that the measure we need must also be able to cope with a set of probability distributions, as it is used in an online monitoring system to track the changes of a dynamic system. This requirement calls for the need of the other two important properties of a divergence—*Generalizability* and *Weightability* (Patil & Rao, 1978; Patil, 2006) as discussed below.

Consider another example, such as we have a real-time monitoring system that keeps converting a symbolic data stream into *m* SVs $(SV_1, SV_2, \ldots SV_m)$ with 3 different states/symbols (k = 3) as shown in Figure 11. We need a measure that can assess the deviation of the system by continuously calculating the divergences from the changes of discrete probability distributions in a set of SVs. That is, the divergence measure must be able to be generalized to compare multiple

symbol probability distributions in different times—the generalizability of a measure. In Figure 11, for example, the divergence should allow for the comparisons of $D(SV_1, SV_2)$, $D(SV_1, SV_2, SV_3)$, $D(SV_1, SV_2, \ldots. SV_m)$, i.e. any combination of the SVs. Also, a practical application of online monitoring systems is that only part of (usually the most recent) developments/activities of a system are important. That is, more recent activities of a system weigh higher. I suggest that a divergence measure should also be able to assign a weight value $\boldsymbol{\pi}_m$ for each distribution we compare—the weightability of a measure. For example, if we believe that monitoring the divergences that compare the latest 4 SVs in Figure 11 is enough to evaluate the system deviation, we can only assign the weights to these 4 SVs and keep the weight of the rest SVs as 0.

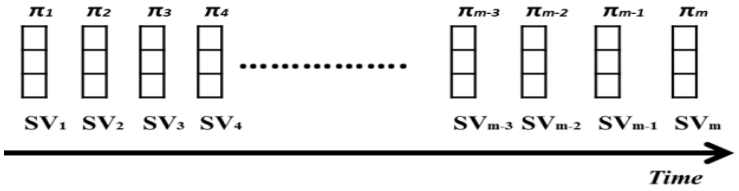


Figure 11: A Series of Steady-State Vectors with Weights

I chose the Generalized Jensen-Shannon Divergence (D_{GJS}) (Jianhua Lin, 1991) as the measure used in my process monitoring approach, because the D_{GJS} possesses all of the four properties mentioned above. D_{GJS} is a symmetric measure that ranges between 0 and 1 (Jianhua Lin, 1991; Lamberti, Majtey, Borras, Casas, & Plastino, 2008) The D_{GJS} is defined:

$$D_{GJS}(P_1, P_2, \ldots, P_m) = H\left(\sum_{i=1}^{m} \pi_i P_i\right) - \sum_{i=1}^{m} \pi_i H(P_i) \qquad \text{(Eq. 9)}$$

where $\boldsymbol{\pi_i}$ is the weight and $\sum \boldsymbol{\pi_i} = 1$. The P_m is the discrete symbol probability distribution vectors we compare. In my proposed approach, P_m are the SVs from different sequences. $H(x)$ is the *k-ary Shannon Entropy* that is defined as:

$$H(\mathrm{x}) = -\sum_{i=1}^{k} P(x_i) \log_k P(x_i) \qquad \text{(Eq. 10)}$$

The D_{GJS} can compare arbitrary number of the SVs. All the SVs can also be weighted. Take the five SVs of previous market index example in Figure 10 as an example. If we want to track all the changes of the market ups and downs, we will take all the SVs into consideration. That is, to compute $D_{GJS}(SV_1, \ldots, SV_5)$ with the equally-weighted $\boldsymbol{\pi} = 0.2$ for all SVs. In this case, the D_{GJS} is 0.1061. Another example is that we want to monitor the recent abrupt changes of the market and believe the last two SVs (i.e. SV_4 and SV_5) are important. Then, we compute $D_{GJS}(SV_4, SV_5)$ with $\boldsymbol{\pi_4} = \boldsymbol{\pi_5} = 0.5$ to obtain $D_{GJS} = 0.1362$. It is certain that higher D_{GJS} indicates higher deviation. However, we also need a magnitude guideline (i.e. how high D_{GJS} is too high) for the D_{GJS} to assess the significance of the deviation so that we can make a decision based on it.

3.1.3 Significant Threshold and the Deviation Assessment

There is no fixed value of D_{GJS} as a threshold that indicates the divergence is "high enough" to take action. From the definition of D_{GJS}, we can see that the D_{GJS} varies dramatically based on 3 factors—(i) the number of components in an SV $\boldsymbol{k}$ (the number of different symbols); (ii) the number of distributions/SVs $\boldsymbol{m}$ we compare; (iii) the weights for all the distributions/SV $\boldsymbol{\pi}$. That is, even for the same symbolic data, the number of different symbols/states k we choose when we symbolize/discretize the raw data, the number of sequences/SV m we compare, and the weights of SVs $\boldsymbol{\pi}$ we assign, can noticeably increase or decrease the D_{GJS}. Here, I first illustrate how the number of states k affects the D_{GJS} by an example. Suppose we have two distributions/SVs SV_1 and SV_2. Both of them have k probabilities (for k states/symbols) and a dominant state with the probability of 1.0. Nevertheless, the dominant state in SV_1 is the first state, whereas the dominant

state in SV_2 is the second state. The probabilities for the rest of the states for SV_1 and SV_2 are all zero. These two SVs are equally-weighted, i.e. $\pi_1 = \pi_2 = 0.5$. Figure 5 shows the D_{GJS} decreases as k increases when we compare these two distributions.

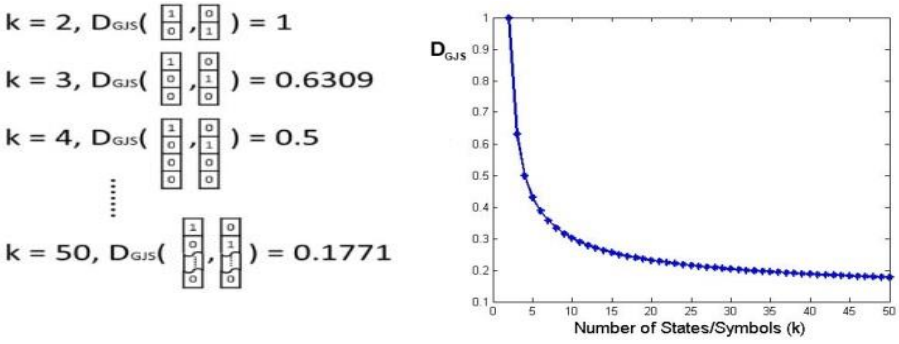


Figure 12: D_{GJS} vs. Number of Different State (k)

Apparently, we can explain the negative association by the change of the data granularity. When applying a data discretization method to symbolize the data, a higher number of different symbols (the number of states in an SV) will increase the granularity and proportionally diminish the differences among the symbol probability distributions we compare. We can also explain it by the definitions of the k-ary Shannon Entropy Eq. 10. As the base k increases, the entropy decreases. Correspondingly, the D_{GJS} decreases as k increases.

The number of SVs m and the weights for these SVs $\boldsymbol{\pi}$ also have a great impact on the D_{GJS}. From the definition of D_{GJS} , we can see D_{GJS} allows multiple weighted SVs (distributions). Consider a simple example that we have a monitoring system continuously comparing the equally-weighted (i.e. $\boldsymbol{\pi}_1 = \boldsymbol{\pi}_2 = \ldots = \boldsymbol{\pi}_m = 1/m$) m SVs created from a symbolic data stream. Figure 13 shows the example with the distributions of the last SV different from all previous SVs with $k = 3$. Note that the second probability is 1 instead of 0 in the last SV for different m up to 50.

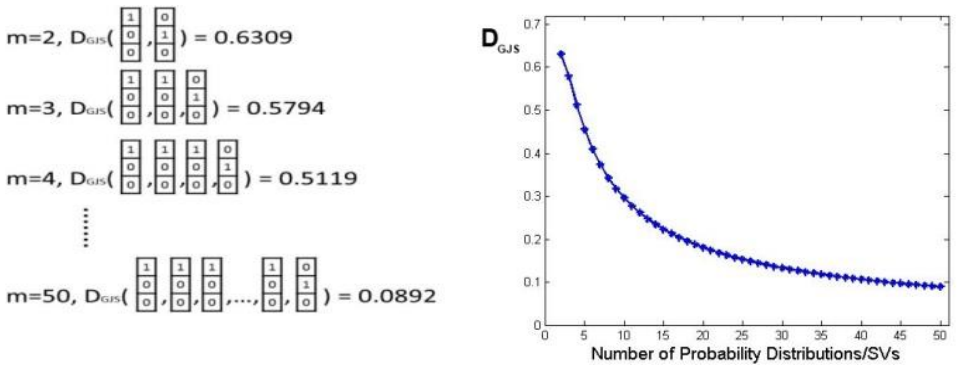


Figure 13: D_{GJS} vs. Number of Probability Distributions (m)

We expect the monitoring system should report that the D_{GJS} is "significantly high" for m SVs when it compares the last SV with all previous ones, because the probability distribution of the last SV is significantly different from previous SVs. However, it is clear the threshold to define the "significance" should also depend on m and $\boldsymbol{\pi}$. Again, we can explain this by the definitions, i.e. Eq. 9 and Eq. 10 . In the example, the weights $\boldsymbol{\pi}$ for all m SVs are equal. If the number of SVs m increases, the influence of each SV reduces so that the D_{GJS} decreases. On the other hand, if we assign a very high weight to the last SV, the D_{GJS} will increase dramatically as the influence of the last SV increases. Therefore, the number of the SVs m and the choice of the weight $\boldsymbol{\pi}$ also play an important role when we determine the significant threshold of the D_{GJS}.

The significant threshold of the D_{GJS} is a certain value of D_{GJS} that answers the question—"what is the probability that the D_{GJS} is higher than the threshold?". The probability here is the critical p-value (significant level α) commonly used in Statistics. I use $D_{GJS|k,m}$ to denote the threshold. Evidently, the $D_{GJS|k,m}$ is essential for the practical use of the D_{GJS} as the deviation measure. Before introducing the $D_{GJS|k,m}$, I first state the settings and assumptions again. The process monitoring system continuously receives a symbolic data stream and divides it into sequences S_m of total N symbols with k different possible symbols denoted by $A = (a_1, a_2, \ldots, a_k)$.

The sequences S_m are equally-sized ($S_1, S_2, \ldots, S_m$) (i.e. the length of each sequence $n_1 = n_2 = \ldots = n_m$). We can then create m first-order Markov Chains and the transition probability matrices from these sequences. These transition probability matrices are then transformed into the Google Matrices G_m. As G_m are ergodic and small, m unique SVs can be easily obtained by the Power Iteration (Langville & Meyer, 2006). Then, we assign a weight π for each SV depending on different applications as shown in Figure 11 to create a k cells by m SVs table (k = 3 in Figure 11). These SVs are the snapshots of the system we monitor and are of probabilities of these states (symbols). Consider this k by m table, we would like to know how much Information that k symbols and m sequences/SVs share from this table—*Mutual Information* (I). In (Grosse et al., 2002), the task of obtaining the D_{GJS} is interpreted as the task of obtaining the Mutual Information in a symbol a_k about an sequence S_m. That is, provided we know the probability distributions (SVs) of these symbols and what symbol a_k we have drawn from these sequences, how much information I about "which sequences S_m we draw". Here, the mutual information I is defined:

$$I \equiv \mathrm{D}_{GJS}(P_1, P_2, \ldots, P_m) \equiv \sum_{i=1}^{k}\sum_{i=1}^{m} P(x_{ij}) \log_k \frac{P(x_{ij})}{\pi_j P(x_i)}$$
$$= \sum_{i=1}^{k}\sum_{i=1}^{m} \pi_j P(x_i|S_j) \log_k \frac{\pi_j P(x_i|S_j)}{\pi_j P(x_i)} \qquad \text{(Eq. 11)}$$

where $P(x_i \mid S_j)$ is the conditional probability of finding a symbol a_i given a sequence S_j. We expect high variance of $P(x_i \mid S_j)$ if the system we monitor is deviated. On the other hand, if the system we monitor is stable, we expect that the probabilities of each symbol a_k in different SVs are very close, and therefore both the I and D_{GJS} are close to zero.

As also described in (Grosse et al., 2002), the D_{GJS} in Eq. 11 can be analytically approximated by using the Taylor expansion as

$$\mathrm{D}_{GJS} \equiv \sum_{i=1}^{k}\sum_{i=1}^{m} \pi_j P(x_i|S_j) \log_k \frac{\pi_j P(x_i|S_j)}{\pi_j P(x_i)} \simeq \sum_{i=1}^{k}\sum_{i=1}^{m} \frac{(\pi_j P(x_i|S_j) - \pi_j P(x_i))^2}{\pi_j P(x_i)(2\ln k)} \qquad \text{(Eq. 12)}$$

Let us take a close look at Eq. 12 with Figure 11.The *m* SVs with *k* states/symbols (*k* = 3 in Figure 11) can be considered an *k* by *m* contingency table if we multiply each SV by its weight and *N* (the total number of symbols in *m* sequences). The Eq. 12 can thus be expressed by the Chi-square statistic χ^2 (Grosse et al., 2002), (Herzel & Große, 1997) as Eq. 13:

$$\chi^2 \equiv N \sum_{i=1}^{k}\sum_{i=1}^{m} \frac{(\pi_j P(x_i|S_j) - \pi_j P(x_i))^2}{\pi_j P(x_i)} \simeq 2N(\ln k) D_{GJS} \qquad \text{(Eq. 13)}$$

We can then rewrite Eq. 13 to obtain the expected D_{GJS} as shown in Eq. 14:

$$D_{GJS} \simeq \frac{\chi^2}{2N(\ln k)} \qquad \text{(Eq. 14)}$$

Therefore, given a certain significant level α, the number of SVs *m*, and the number of states *k*, we can derive an asymptotical approximate threshold for the D_{GJS}, the $D_{GJS|k,m}$, as:

$$P(D_{GJS} \le D_{GJS|k,m}) \simeq F(2N(\ln k) D_{GJS|k,m}, df) \Rightarrow D_{GJS|k,m} \simeq \frac{\chi^2_{df,1-\alpha}}{2N(\ln k)} \qquad \text{(Eq. 15)}$$

where *F* is the Chi-square cumulative distribution function given the degree of freedom *df* = (*k* - 1)(*m* - 1). $P(D_{GJS} \le D_{GJS|k,m})$ denotes the probability of the D_{GJS} less or equal to the threshold

$D_{GJS|k,m}$. The $D_{GJS|k,m}$ in Eq. 15 is used as the criterion to determine whether the system deviation is significant.

In Figure 14, I provide an example that shows how the $D_{GJS|k,m}$ works as the thresholds in my proposed monitoring approach, given that we have series of SVs shown in Figure 10.

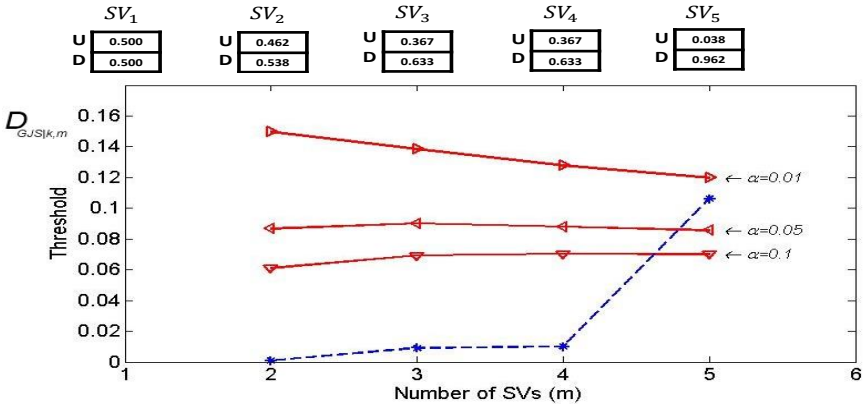


Figure 14: A series of steady-state vectors with the significant thresholds

Note that I consider all the SVs in Figure 14 are equally-weighted, which means, at the time when the system generates m SVs, we have $\boldsymbol{\pi}_1 = \boldsymbol{\pi}_2 = \ldots = \boldsymbol{\pi}_m = 1/m$. That is, for example, the weights for 3 SVs are all 1/3. Each SV is created from a sequence of 16 symbols as shown in Figure 10. The total length of all the sequences, N, increases as the system keeps converting the symbolic data stream into SVs. Note that N must be sufficiently large to avoid obtaining the Chi-square statistic in Eq. 15 that may commit a Type II error. To calculate the $D_{GJS|k,m}$ when we have 2, 3, 4, and 5 SVs in Figure 14, for example, the N is 2 * 16 = 32, 3 * 16 = 48, 4 * 16 = 64, 5 * 16 = 80, respectively. Thus the $D_{GJS|k,m}$ for 2, 3, 4, and 5 SVs in Figure 14 with $\alpha = 0.01$ are $\frac{\chi^2_{(2-1)(2-1),(1-0.01)}}{2*(2*16)*(\ln 2)}$ = 0.1496, $\frac{\chi^2_{(2-1)(3-1),(1-0.01)}}{2*(3*16)*(\ln 2)}$ = 0.1384, $\frac{\chi^2_{(2-1)(4-1),(1-0.01)}}{2*(4*16)*(\ln 2)}$ = 0.1279, and $\frac{\chi^2_{(2-1)(5-1),(1-0.01)}}{2*(5*16)*(\ln 2)}$ = 0.1197,

respectively. In Figure 14 I provide 3 threshold lines for 3 different significant levels, α = 0.01, 0.05, and 0.1. These 3 lines can also be interpreted as the probabilities of the D_{GJS} higher than these lines, which are 0.01, 0.05, and 0.1 respectively. The lower the significant level α, the higher the threshold $D_{GJS|k,m}$. Also in Figure 14, we can see the actual D_{GJS} (dashed line) that compare (SV_1, SV_2), (SV_1, SV_2, SV_3), and (SV_1, SV_2, SV_3, SV_4), are all lower than the $D_{GJS|k,m}$. However, as expected, the D_{GJS} at the time when we compare (SV_1 , ... , SV_5) is much higher than previous ones, as the SV_5 is significantly different from previous SVs. Given α = 0.05 or α = 0.1, the monitoring system will give an alert that the system we monitor may deviate.

By modifying the aforementioned parameters, the proposed monitoring approach can not only be used in general anomaly and change-point detection, but also any activity monitoring. Those steps to create the monitoring system are illustrated in Figure 15. We first choose appropriate data reduction methods and the data granularity (the *k* number of possible symbols that represent *k* patterns of interest). The raw data stream is then discretized into sequences of symbols. The next step is to calculate the symbol stationary probability distributions (SVs) in different times (sequences). We keep dividing all the symbols into *m* equally-sized sequences of *n* symbols. The size of each sequence should depend on whether the sequence can represent the wanted dynamics of the system we monitor in a period. Then, *m* by *k* SVs are generated. In step 3, we compute the actual D_{GJS} from SVs and the significant thresholds of the D_{GJS} given *k*, *m*, and an appropriate significant level α (i.e. 0.1, 0.05, or 0.01). When the actual D_{GJS} is higher than the threshold, the monitoring system gives an alert that indicates the system deviation is critically high and the deviation very unlikely occurs by chance. That is, the probability that the actual D_{GJS} is higher than the threshold is the α we choose.

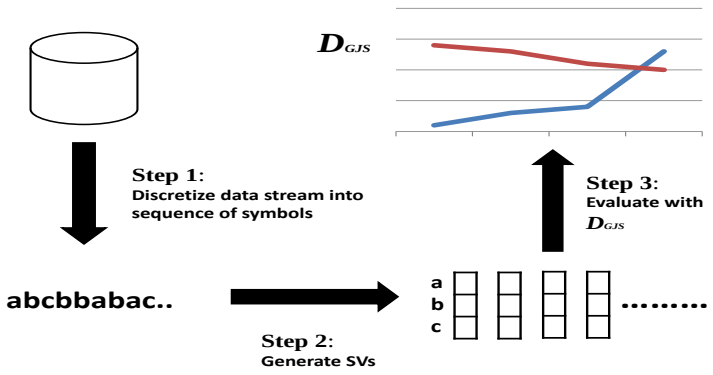


Figure 15: Process of system deviation assessment using D_{GJS}

Again, note that the data reduction methods in Step 1 and the way to estimate the symbol probability distributions in Step 2 can be replaced by other approaches. In the beginning of Section 2, I only introduced the SAX, as I will use it in the experiments shown in the next section. As discussed, many existing pattern classification methods and time-series representation techniques can be used in Step 1. I suggest using a data reduction method that can symbolize the data without losing too much information of interest. Also, some other approaches, such as relative frequency vector (FV) that counts the numbers of occurrences of the symbols, can also be used to obtain the symbol probability distributions. However, the proposed approach to use SVs as the symbol probability distributions in different times is unique. It is proved to outperform others in later experiments in Section 3.2 by comparing my approach to other measures used in sequence anomaly detection.

3.1.4 Stationary Sequence Probability Vectors and Higher-Order Markov Chain

As discussed in Section 3.1.1, the first-order Markov chain is used in the proposed approach to generate stationary state probabilities (steady state probabilities in an SV). However, in a special case that the changes of the probabilities of higher-order state transitions (e.g.

probabilities for temperature state transition sequences like ***CC, CCC***, …, etc.) could reveal the anomalies of a system, using stationary state sequence probability vector (SSeqV) from higher-order Markov chain, instead of using SV, may improve the accuracy of anomaly detection. Consider a simple example that we have 2 sequences with 2 possible states/symbols (***k*** =2) used to create SV and *n*-order SSeqV as shown in Figure 16.

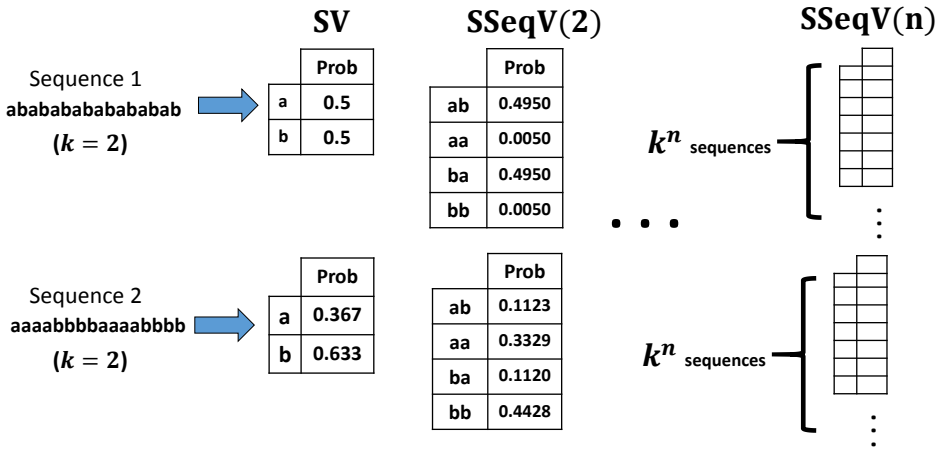


Figure 16: From sequences to SSV and *n*-order SSeqV

Similar to the way I created SVs from the sequences in Figure 10, we can calculate the transition probabilities of *n*-order state sequences from their relative occurrences, convert the transition probability matrices into Google matrices, and then apply the Power Iteration to obtain the *n*-order SSeqV. In Figure 16, we can see that state ***a*** and ***b*** are equally-probable in terms of relative frequency. And SV can help detect the difference between two sequences as it takes into account the transitions of states. However, assuming that these 2 sequences are consecutively generated by a system (i.e. Sequence 1 followed by Sequence 2), the proposed process monitoring technique may not be able to detect one particular system change—the system stability characterized by the

n-order transitions. Figure 16 shows that SSeqV can help detect this change, as the probabilities of ***aa*** and ***bb*** increase significantly from Sequence 1 to Sequence 2.

The advantage of using SSeqV from higher-order Markov chains is very clear, as it can capture more details about the dynamics of a system represented by longer system transitions. However, we can see that sequences with *k* possible states/symbols for *n*-order SSeqV generate k^n stationary sequence probabilities in Figure 16, which is calculated from an k^n by k^n Google Matrix using the Power Iteration. Even though we use a high damping factor (d = 0.99), a large Google Matrix will move/add too many probabilities to those low or zero probability cells of the matrix, which may still result in creating SSeqV that does not represent the actual sequence transitions of the system. Also, higher *k* or *n* will require substantial amounts of computation time to obtain these sequence probabilities and therefore impractical. In Section 3.2.4, I will show the applications and limitations of using SSeqV from higher-order Markov chain.

3.2 APPLICATIONS, COMPARISONS, AND EXPERIMENTS

I investigated the applicability, limitations, and performance of the proposed approach by applying it to two different applications and comparing it to other existing sequence distance measures commonly used in the studies of sequence data anomaly detection and the DNA/Protein sequence evolution. Here, I first define two possible applications of my approach—the change-point and the deviation detection for sequence data. The major differences between these two applications are the number of sequences we compare and the types of system changes we are interested in learning about. In general, the change-point detection is about finding the significant high pairwise sequence distance, whereas the deviation detection is about evaluating the similarity/distance among multiple (more than two) sequences. Here, I consider the change-point

detection as the discovery of abrupt changes. On the other hand, the deviation detection is the notion of finding non-obvious evolutionary relationships among multiple sequences—the gradual deviation of a system development. The experiments in this section include two aforementioned applications with both real-world and artificial datasets.

3.2.1 Sequence Similarity/Distance Measures and the Comparative Evaluation

To perform the comparative evaluation, I enumerate applicable distance measures from the literature about the sequence similarity analysis in different fields. Due to the fact that most of these measures are proposed to obtain the distance between two sequences, I consider the sum of the adjacent pairwise distance for each of them in comparison with the D_{GJS} that can calculate evolutionary distance among multiple sequences. The sum of adjacent pairwise distances is defined in Eq. 16:

$$D(S_1, S_2, \dots, S_m) = \sum_{i=1}^{m-1} D(S_i, S_{i+1}) \qquad \text{(Eq. 16)}$$

where m is the number of sequences we use to compute the distance. Also, a sliding window is used to keep generating the pairwise sequence distances, $D(S_x,S_y)$, of each paired sequences. As the actual changes/anomalies may occur anywhere in sequences, I use another fixed length monitoring window, denoted by $\Delta[i,j]$, to label whether anomalies occur within the period. Figure 17 shows the sliding window $\boldsymbol{W}$ keeps shifting among sequences. The $D(S_x,S_y)$ denotes a distance measure calculated after I collect symbol data from sequences S_x and S_y.

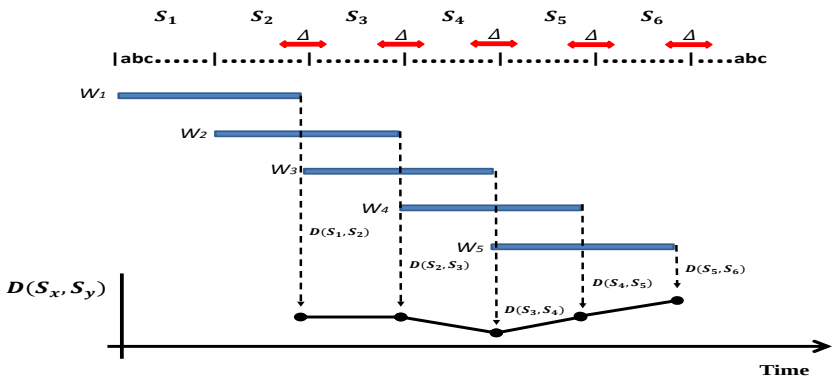


Figure 17: Pairwise sequence distances

Note that each Δ ranges beyond the boundaries of sequences. That is, I also consider a case of early warning that the anomalies may not only exist in the two sequences used to compute $D(S_x,S_y)$ but also may happen in the beginning of the next sequence. For example, $D(S_1,S_2)$ is computed after we have S_1 and S_2. However, the first Δ is across the boundaries between S_2 and S_3, as we believe that the anomalies could occur somewhere around the end of S_2. Besides, consider the application of deviation detection that takes more than two sequences into account to find the gradual changes, we use the sum of adjacent pairwise distances for each distance measure instead. Figure 18 shows an example that computes the gradual distance by comparing 3 sequences (m = 3).

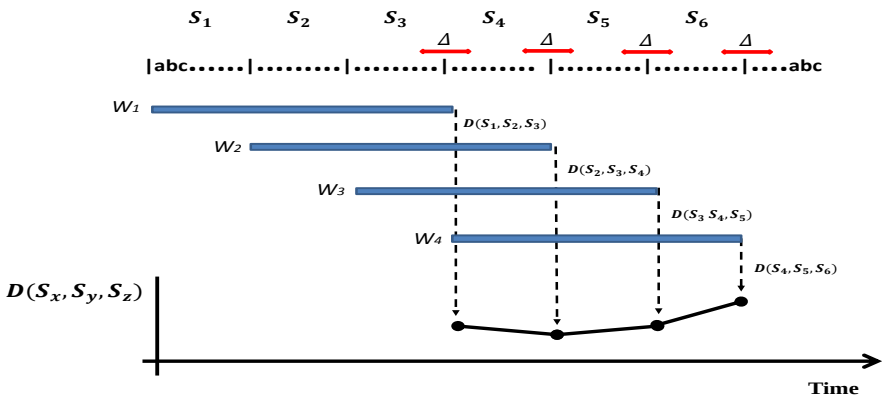


Figure 18: Sum of pairwise sequence distances (given m = 3)

Note that, for example, the first distance $D(S_1,S_2,S_3)$ is the sum of two adjacent pairwise distances, i.e. $D(S_1,S_2) + D(S_2,S_3)$. Again, if there is any anomaly within a Δ, the target/outcome label of the corresponding distance measure will be positive (anomaly). These distances and labels are then used in the performance evaluation in the next sections to create the ROC curves and compute the Area Under the ROC Curves (AUC).

As the proposed approach is related to sequence similarity analysis found in various fields, in Table 3**,** I list the aforementioned divergences and five applicable distance measures with their notations used in later experiments. I choose these measures based on whether they can be applied to the comparison of the sequences with absent symbols. That is, those measures should allow for the comparison of two sequences in which a symbol that represents a system state may never occur. For example, consider two sequence S_1 (***a,b,c,a,b,c***) and S_2 (***a,b,a,b,a,b***). The symbol ***c*** is absent in S_2. The measure we use must be able to compute the distance/similarity that reflects the absentness of the symbol ***c***. Therefore, some of the measures, such as Paralinear distance (Lake, 1994) used in the calculation of distance of DNA/Protein sequences, are not applicable to the evaluation.

Table 3: Sequence Similarity Measures

Measure	Notation	Equation
Generalized Jensen-Shannon Divergence on Steady-State Vectors	D_{GJS} + SV	(Eq. 9)
Generalized Jensen-Shannon Divergence on Relative Frequency Vectors	D_{GJS} + FV	(Eq. 9)
Kullback-Leibler Divergence on Steady-State Vectors	D_{KL} + SV	(Eq. 4)
Kullback-Leibler Divergence on Relative Steady-State Vectors	D_{KL} + FV	(Eq. 4)
Normalized length of Levenshtein distance	nLevD	$nLevD(S_1, S_2) = \frac{LevD(S_1, S_2)}{\sqrt{\lvert S_1 \rvert * \lvert S_2 \rvert}}$ (Eq. 17)
One minus Normalized length of the Longest Common Subsequence	1-nLCS	$1 - nLCS(S_1, S_2) = 1 - \frac{LCS(S_1, S_2)}{\sqrt{\lvert S_1 \rvert * \lvert S_2 \rvert}}$ (Eq. 18)
Cosine Distance on Steady-State Vectors	CosDist + SV	$CosDist(S_1, S_2) = 1 - \frac{V(S_1) \cdot V(S_2)}{\lVert V(S_1) \rVert * \lVert V(S_2) \rVert}$ (Eq. 19)
Cosine Distance on Relative Frequency Vectors	CosDist +FV	Eq. 19
p-Distance	D_p	$D_p(S_1, S_2) = \frac{d}{n}$ (Eq. 20)
Jukes-Cantor distance	D_{JC}	$D_{JC}(S_1, S_2) = \begin{cases} -\frac{k-1}{k} \ln\left(1 - \frac{k}{k-1} D_p\right), \text{if } D_p \leq \frac{k-1}{k} \\ +\infty, \text{if } D_p > \frac{k-1}{k} \end{cases}$ (Eq. 21)

In Table 3, I define the *normalized length of Levenshtein distance* (*nLevD*) as Eq. 17, which is a measure that computes the ratio of edit distance (the number of insertions/deletions/substitutions operations needed to convert a sequence into another) between two sequences. $LevD(S_1,S_2)$ denotes the amount of edit distances between two sequences, whereas $|S_1|$ and $|S_2|$ are the length of sequences. I consider *nLevD* the degree of mismatch of two sequences and use it as a distance measure to detect the changes of a system. Similar to *nLevD*, *normalized length of the Longest Common Subsequence* (*nLCS*) (Budalakoti, Budalakoti, Srivastava, Otey, & Otey, 2009) is a measure derived from the algorithm to find the *Longest Common Subsequence* (*LCS*). The *LCS* is a common but not necessarily consecutive subsequence among two or more sequences. It can be used to assess the similarity of sequences. In Eq. 18, $|LCS(S_1,S_2)|$ denotes the length of the longest common subsequence. The *nLCS* ranges from 0 to 1. The higher *nLCS* indicates higher similarity between sequences. In later experiments, I use *1 – nLCS* as a distance

measure and only consider the case of comparing two sequences, as finding the *LCS* for more than two sequences is an NP-hard problem (Jiang & Li, 1995) and thus impractical. Eq. 19 defines *Cosine Distance* as *1- Cosine Similarity*. The Cosine Similarity is a common measure used to find the similarity between two documents in the field of text mining (Srivastava & Sahami, 2009). Instead, I use it to measure the similarity between two probability vectors, i.e. the discrete symbol probability distributions from the relative frequency vectors (FV) and the steady state vector (SV). In Eq. 19, $V(S)$ and $\|V(S)\|$ denote the symbol probability vector generated from a sequence and the norm of the vector respectively. As all the cells/components of the probability distributions (vectors) are always greater or equal to 0, the Cosine Distance ranges between 0 and 1.

As discussed in previous sections, many similarity/distance measures are proposed to help find the evolutionary distance of DNA/Protein sequences. In later experiments, I also consider two applicable measures—the *p-distance* (D_p) and *Jukes-Cantor distance* (D_{JC}) (Durbin, 1998) as defined in Eq. 20 and Eq. 21. The D_p is the proportion of locations that differ between two sequences. In Eq. 20, d is the number of one-to-one mismatched symbols and n is the length of a sequence. Note that the lengths of two sequences must be the same. The D_p is simple and easy to compute, but it underestimates the possible substitution of each symbol at each location. Consider three possible symbols (***a***, ***b***, ***c***) in a sequence as an example. Each symbol in the sequence can be replaced by two other symbols. That is, for k possible symbols in a sequence, each symbol in a sequence can be replaced by $k-1$ symbols. The p-distance does not reflect the granularity of the sequence data that contributes to the distance of two sequences. The D_{JC} is proposed to correct the problem. It is originally assumed that the nucleotide symbol substitution rate (replacement rate) and symbol frequency are all equal. It is also applicable in our experiments, as all symbols are assumed equal-probable before the discretization. In Eq. 21, k is the number of possible symbols

(states). Note that, by the original definition, the D_p in Eq. 21 is expected to be smaller than (k-1)/k. Instead, I use $+\infty$ when the D_p is higher than (k-1)/k.

3.2.2 Change-Point Detection

I created two synthetic datasets called DC (which denotes "Distribution Change") and JM (which denotes "Jumping Mean"), to demonstrate how the D_{GJS} and other measures we discuss perform when they are applied to the detection of the change-points. The DC dataset is created as follows. I randomly generate 900 uniformly-distributed data points between -3 and +3, followed by another 300 normally-distributed random data points with the parameters (μ = 0, σ = 1). This process is repeated 25 times and all generated data are then concatenated to form the DC dataset with 30,000 data points. Figure 19 shows the first 3,600 data points of the DC dataset.

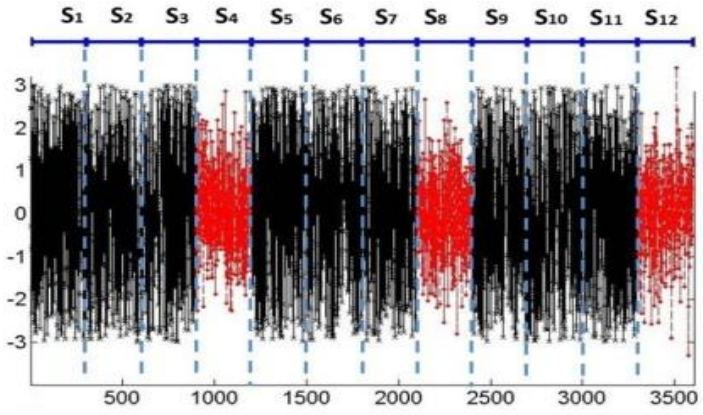


Figure 19: First 3,600 data points from DC dataset

The goal of using the DC dataset is to see whether the measures can detect the changes of the data distributions. I symbolize the dataset using the SAX with different numbers of possible symbols (k). The segment size to create a symbol in the SAX is 3 data points, i.e. there are a total

of 10,000 symbols generated from the DC dataset. The length of each sequence, *n*, is set to 100 symbols. Therefore, for example, there are 12 sequences (1,200 symbols) from the data points in Figure 19. Apparently, the change points are at 901, 1201, 2101, 2401, and 3301 (i.e. at the beginning of the sequence S_4, S_5, S_8, S_9, and S_{12}). The pairwise D_{GJS} and other distance measures $D(S_x,S_y)$ are then computed. As we calculate the distances after receiving pairs of sequences, for total *m* sequences we can obtain (m – 1) distances. Figure 20 shows the actual pairwise D_{GJS} in Figure 19.

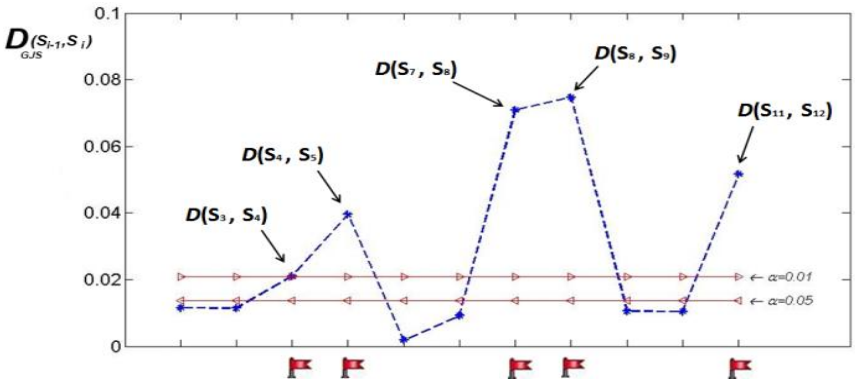


Figure 20: D_{GJS} and the thresholds with different α

Note that the number of possible symbols (*k*) in Figure 20 is 3, and 11 pairwise D_{GJS} are computed. In my approach, the significant threshold $D_{GJS|k,m}$ is used to help detect the abrupt changes. Figure 20 also shows the thresholds with different α (0.05 and 0.01). Note that the thresholds in different times are all the same (which are $\frac{\chi^2_{(3-1)(2-1),(1-0.05)}}{2*(2*100)*(\ln 3)}$ and $\frac{\chi^2_{(3-1)(2-1),(1-0.01)}}{2*(2*100)*(\ln 3)}$ for $\alpha = 0.05$ and $\alpha = 0.01$ respectively), because we continuously compare two sequences. The number of sequences we compare (*m* in Eq. 9) and the total number of symbols in two sequences (*N* in Eq. 9) are thus always 2 and 200 respectively. Provided that the significant level $\alpha = 0.05$, we can see

the actual D_{GJS} are higher than the thresholds at the times when two sequences with different distributions are compared. The monitoring system based on proposed approach can thus alert us for the abrupt changes.

As all the scales of the aforementioned measures are not equal, I consider using the AUC to compare our approach with other distance measures. For DC dataset, the Δ is set to [-5, +5] at the end of each sequence, which certainly can capture the change points and generate the positive anomaly labels used in plotting ROC curves. Figure 21 shows the AUCs of the measures with different number of possible symbols (*k*).

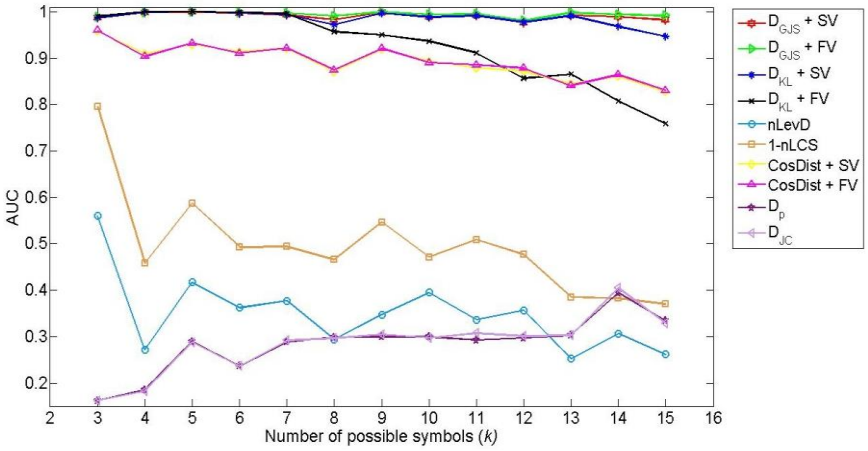


Figure 21: The AUCs of distance measures for DC dataset

Figure 21 suggests that those measures based on finding the (mis)matched symbols perform poorly compared to those based on computing the distances of two symbol distributions. One major reason is that the DC dataset is randomly generated. In this case, the proportion of matches between two sequences is usually lower. Besides, we can see that the AUCs of most of the measures (except D_{GJS}) decreases as the number of possible symbols (*k*) increases, which also indicates that the

performance of these measures declines when they are applied to high-granularity temporal sequence data. Apparently, they should not be used as the measures in the process monitoring. On the other hand, the advantages of using D_{GJS} and SV are clear. Even with higher k, the AUCs of D_{GJS} are nearly constant when D_{GJS} is used in the comparison of two (m = 2) sequences. Also, the D_{GJS} + *FV* performs slightly better than D_{GJS} + *SV*, as FV is a better estimate than SV when they are both applied to measuring the symbol probabilities from data points randomly generated from a given distribution. Another interesting result is that the AUCs of D_{KL} + *SV* is higher than the AUCs of D_{KL} + *FV*, which suggest that SV can improve the performance of D_{KL} when D_{KL} is used in measuring the differences of higher-granularity sequence data.

Consider a different application to detect jumping means. The JM dataset shown in Figure 22 consists of 30,000 data points generated by the following auto-regressive model borrowed from (Yamanishi & Takeuchi, 2002).

$$X_t = 0.6\, X_{t-1} - 0.5\, X_{t-2} + \varepsilon_t$$

where ε_t is the Gaussian random variable with mean μ and standard deviation σ = 1. The change points are inserted at time 1,000x (x = 1, 2, …, 29). The mean μ at time t is defined as:

$$\mu_t = 3 \left\lfloor \frac{t}{1000} \right\rfloor$$

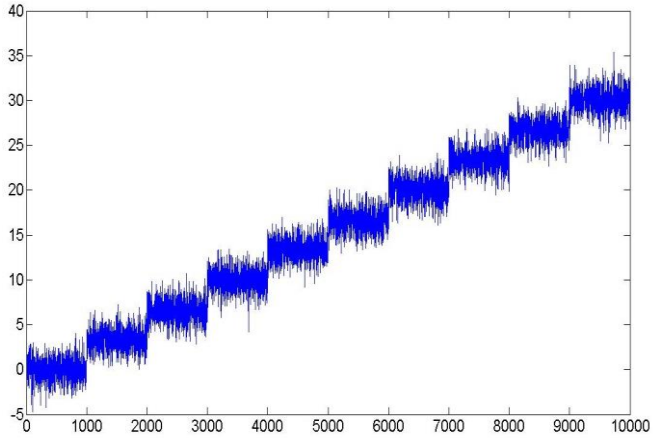


Figure 22: First 10,000 data points of JM dataset

The goal of using JM dataset is to see how the aforementioned measures perform when they are applied to the sequences generated from SAX given different numbers of possible symbols(*k*). The segment size to create a symbol is set to 2 data points, and the length of each sequence, *n*, is set to 50 symbols. Figure 23 shows the AUCs of the measures with different numbers of possible symbols (*k*) for JM dataset. Due to the nature of SAX, we expect that higher k (higher granularity) for SAX will lead to higher AUCs. Figure 23 shows that the D_{GJS} is a better measure. With higher granularity sequence (higher *k*), using D_{KL} with SV can also improve the performance in terms of AUC.

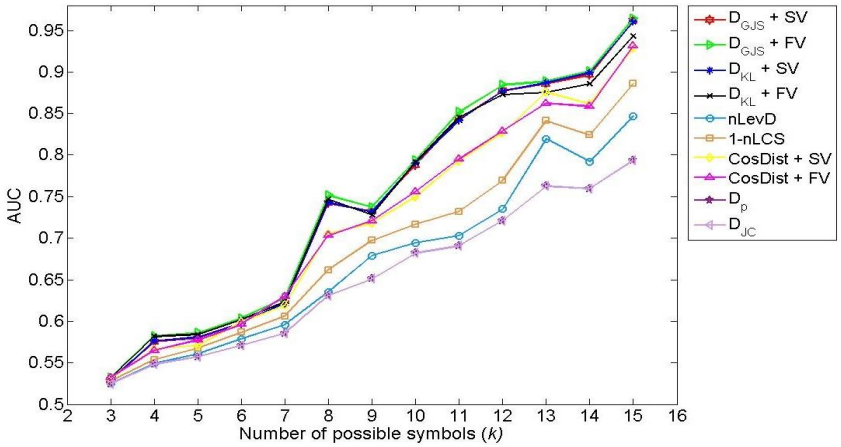


Figure 23: The AUCs of distance measures for JM dataset

3.2.3 Deviation Detection

The third dataset I used is a real-world dataset collected by the power stations on the border between Croatia and Bosnia. Those stations in different locations recorded the measurement (Megawatt Hour, MWh) of the power transmission/consumption every 15 minutes from 2005 to 2008. I select one dataset from an active station. The dataset is then named CAF_BIH consisting of 137,568 data points. The goal is to see how my approach can identify the deviation of the power usage development, the sequential pattern changes (symbol transitions), to help detect the power surges/spikes (the anomalies). As the power surges (spikes) are expected to be rare, I use the cut-points to discretize the CAF_BIH data instead of using the SAX. I first consider any data points greater than 20 MWh as the power surges as shown in Figure 24. Then, the cut-points are used to discretize the data and determine the number of possible symbols (k). In Figure 24, I provide two cut-points (the green dashed lines at 10 and 20) as an example that discretizes the CAF_BIH data into a symbolic data stream with k = 3. That is, 137,568 data points become a long string that consists of 3 possible symbols (***a***, ***b***, ***c***), based on which area a data point is in.

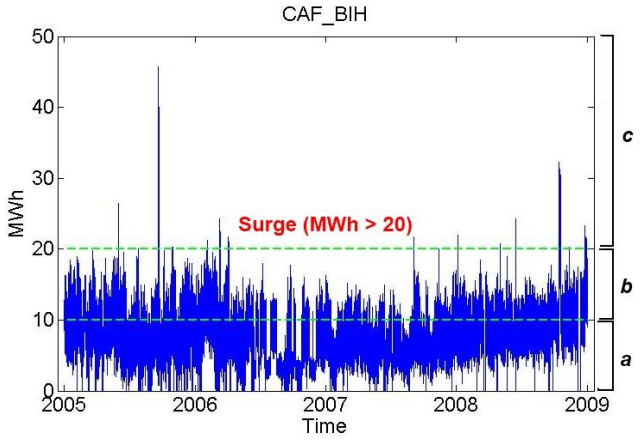


Figure 24: CAF_BIH power consumption data

The next step is to determine the size (length) of each sequence (*n*) to compare. Apparently, a long sequence may result in the delay of the alarm, whereas a short sequence may not be able to discover the symbol transition patterns (e.g. the cycle of the symbol transitions). I empirically choose 48 symbols (*n* = 48), which is equal to 12 hours, as the length of for each sequence. Figure 25 shows a sample of the settings used in the experiments on the CAF_BIH data.

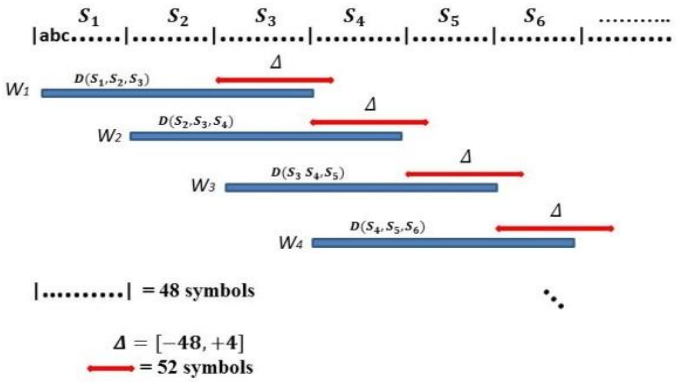
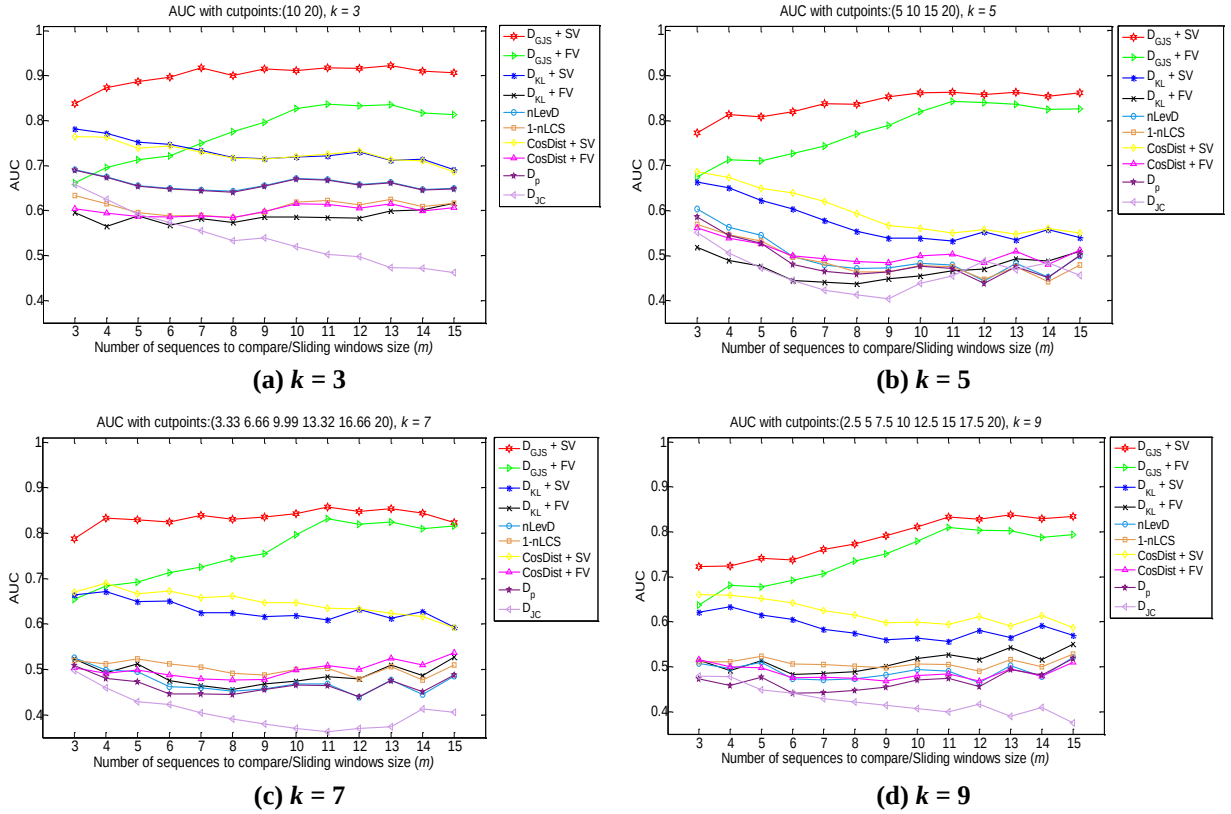


Figure 25: Sequences with sliding windows (m = 3) for CAF_BIH

Note that the Δ is set to [-48, +4] (52 symbols), which is minus 12 hours/plus 1 hour at the point when the distance measures are computed. In this case, the surges occurring within this range will

be captured and used as the positive anomaly labels in the later ROC analysis to calculate the AUCs. The length of the Δ could vary among different applications. In Figure 25 and all of our experiments for the CAF_BIH data, I consider the case that we have been monitoring the power usage development for a certain period (e.g. m = 3 sequences = 36 hours in Figure 25). We expect that the surges may occur in the last sequence (last 12 hours) or in the near future (after 1 hour) as the early warnings.

To discretize CAF_BIH data with different k, I use different numbers of cut-points to divide the data below the surge line (MWh <= 20) into equally-sized areas marked as symbols. Those data points above the surge line are also labeled as surges. Figure 26 shows that the measures based on comparing symbols probability distributions from Steady-State Vectors perform better than those based on counting the ratio of (mis)matches of symbols do, especially when all measures are applied to the comparisons of multiple sequences to obtain the evolutionary distances.



(a) *k* = 3 **(b) *k* = 5**

(c) *k* = 7 **(d) *k* = 9**

Figure 26: AUCs with different parameters for CAF_BIH data

In addition, I suggest using D_{GJS} instead of D_{KL}, as I observe that the generalizable and symmetric measures are better metrics to assess the deviation of a system development. Figure 26 shows most of AUCs for D_{GJS} are higher than those for D_{KL}. Also, we can see that using SV results in higher AUCs for D_{GJS}, D_{KL}, and *CosDist* in most cases, as SV can maximize the differences among the symbol probability distributions we compare. On the other hand, using FV could perform comparably well, especially when we compare more sequences. It can be explained by the nature of SV and FV. As discussed, the SV can be considered a snapshot of a system development. The comparison of fewer sequences can maximize the system deviation within a smaller time frame (the length of the all sequences we compare). Nevertheless, when more

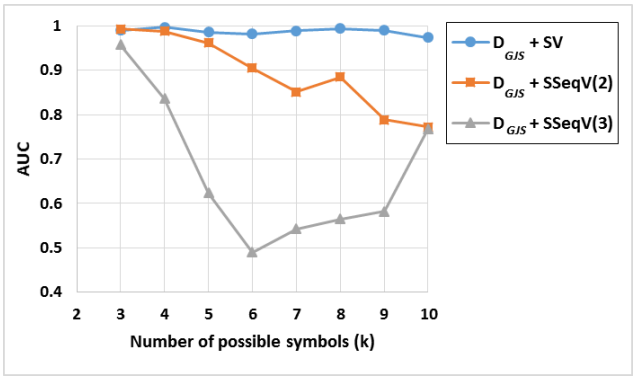
sequences are used in the comparison (higher *m*), the measures that use FVs might collect large enough symbol probabilities close to the real symbol probability distribution. Also, I assume that using SV to estimate symbol probability is better than using FV in most cases, because SV can also capture the unusual symbols transition. However, in a special case that there are only a few symbols transitions in a long sequence, FV might be a better estimator, since FV represents the actual symbol probabilities but SV from the Google Matrix estimates symbol probabilities by padding some probabilities to those low- or zero- probable cells of the stochastic matrix.

Last but not least, the data discretization method used in symbolizing the data also has an impact on the performance of these distance measures in a monitoring system. In Section 3.2.2 and 3.2.3, I used two different methods, SAX and cut-points, but both suggested that finer data granularity (higher *k*) may result in lower AUCs for all the distance measures. It is not always the case. Better classifiers (the discretization rules) identified by the domain experts or pattern classification methods may help discover the lurking sequential pattern dynamics, which can also result in higher AUCs regardless of the number of possible symbols/patterns (*k*).

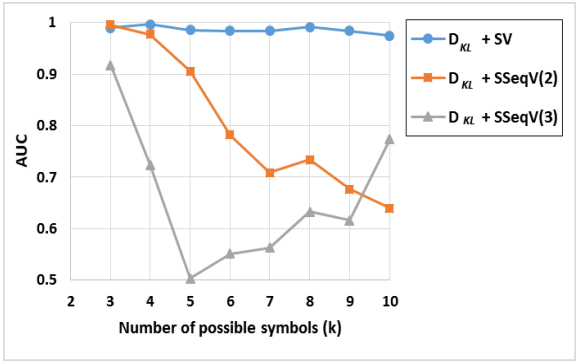
3.2.4 Using Stationary Sequence Probability Vectors

As discussed in Section 3.1.4, the proposed approach to create SV can also be applied to higher-order (longer) state transitions to obtain the stationary sequence probability vector (SSeqV). Using those distribution-based distance measures (i.e. D_{GJS}, D_{KL}, and *CosDist*) introduced in Table 3 with *n*-order SSeqV may increase the accuracy of sequence anomaly detection in the case that the sequence shows a long-term changes of higher-order sequence patterns (longer state transitions). Consider the synthetic dataset, DC, in Section 3.2.2 with identical experimental settings. Figure 27 shows that using those measures with SSeqV does not increase the accuracy of

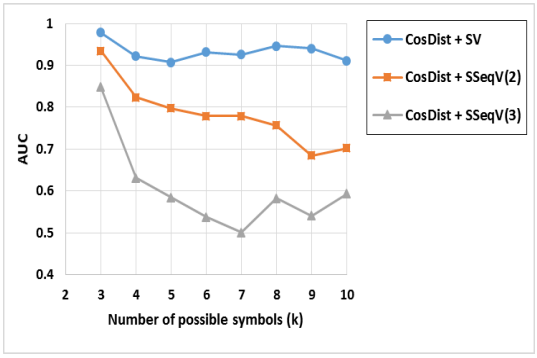
detection. Higher data granularity (*k*) would even decrease the accuracy most likely because the stochastic matrices used to generate the Google matrices and are too large and too sparse. In this case, we move/add to many probabilities to those cell with low or zero probabilities. Also, the way we created DC dataset can explain these results, too. The probabilities for all possible higher-order sequence patterns (e.g. ***aa, bb***, ***ab***, ***ba*** for 2 symbols with 2nd-order sequence) are very close, as we created these data points randomly. And the sequences generated from DC dataset could barely show the significant differences of the probabilities of these higher-order sequence patterns.



(a)



(b)



(c)

Figure 27: AUCs for measures with up to 3-order SSeqV for DC Dataset

Figure 28 also show that using aforementioned measures with SSeqV does not result in higher accuracy of detection when coping with the jumping means. Again, we can explain it from the data generation point of view. With higher *k*, the symbols/states generated by SAX between the jumping points are most likely be the same. That is, for example, the stationary probability for same-state transitions (***aa***, ***aaa***, ***aaaa***,…, etc.) would be very close before the jumping points. In this case, using SSeqV and using SV would both generate probability vectors with similar changes of probabilities, which therefore results in close accuracies of predictions as shown in Figure 28.

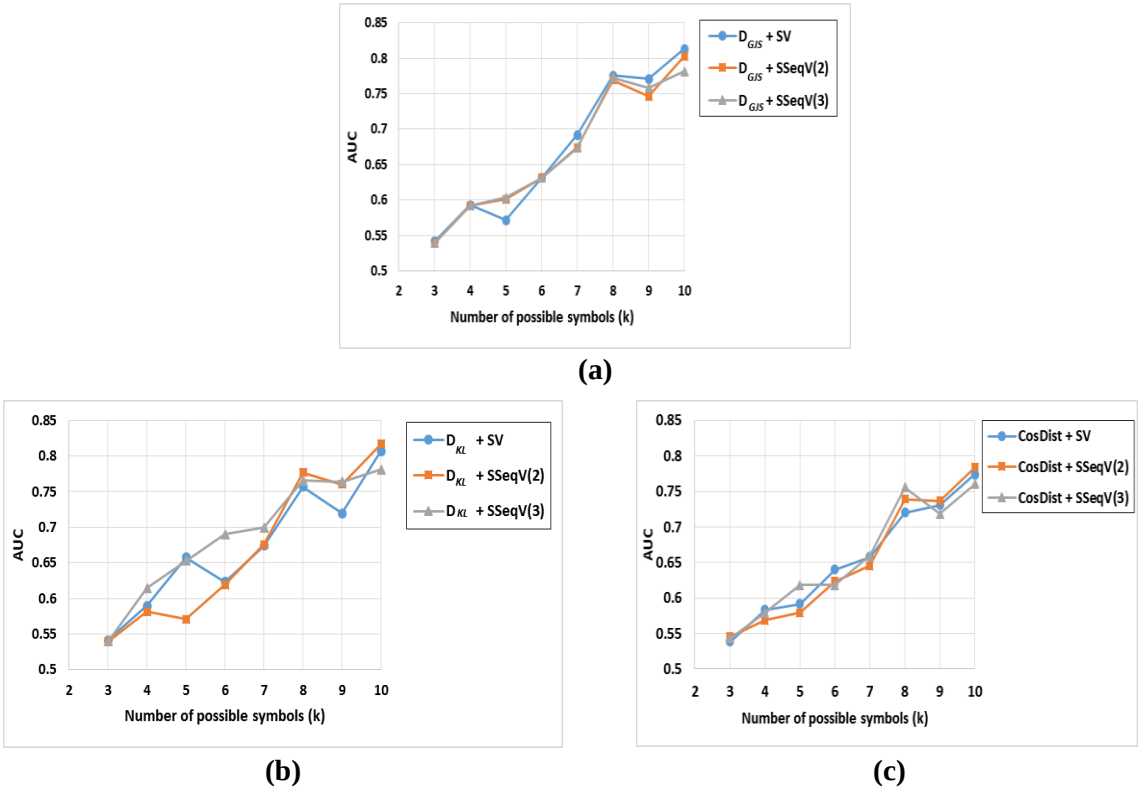


Figure 28: AUCs for measures with up to 3-order SSeqV for JM Dataset

It appears using SSeqV does not show any advantage. However, as discussed in Section 3.1.4, SSeqV may be useful when we monitor a system that presents long-term changes of longer

state transitions (sequence patterns). Let us consider another synthetic dataset called STB that shows the changes of system stability.

The STB dataset is created as follows. I first randomly generate 100 uniformly-distributed data points between 0 and 1, which I call "unstable" area. Then uniformly-distributed 10 data points between 0 and 0.5 with another 10 data points between 0.5 and 1 are concatenated. And then I repeated this process for 10 times to create another "more stable" area with 200 data points. By generating this two "areas" for 20 times, we can create a series dataset with 6,000 data points. Figure 29 show the first 400 data points of the STB dataset.

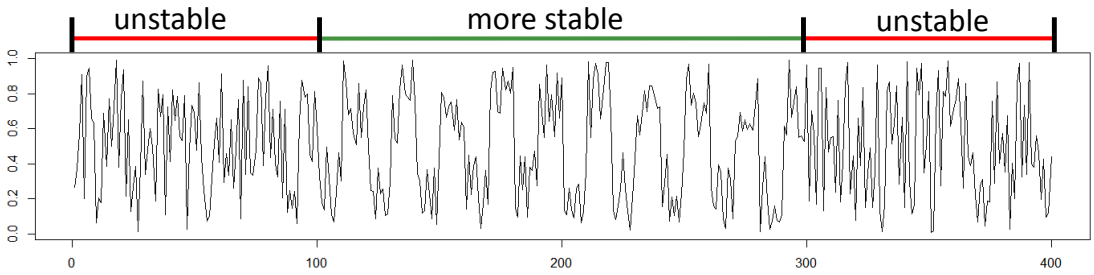


Figure 29: First 400 data points of STB dataset

It is obvious that the change points are between those "unstable" and "more stable" areas. Then, I symbolize the dataset using the SAX with different numbers of possible symbols (k) again. The segment size to create a symbol in the SAX is 1 data point, i.e. there are total 6,000 symbols generated from the STB dataset. The length of each sequence is set to 100 symbols and the Δ is set to [-5, +5] at the end of each sequences to generate the positive anomaly labels used in later ROC analysis. Figure 30 shows the AUCs of three measures with SV and SSeqV for different number of possible symbols (k).

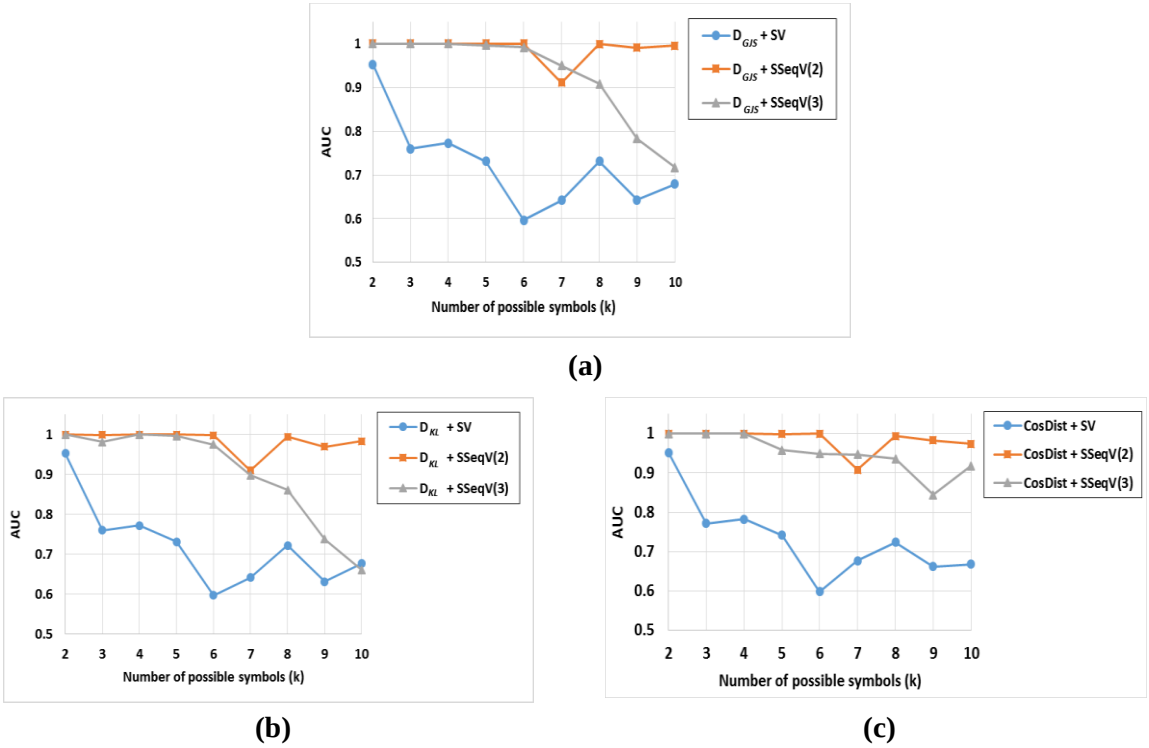


Figure 30: AUCs for measures with up to 3-order SSeqV for STB Dataset

Interestingly, even with higher data granularity, using SSeqV instead of SV can improve the accuracy of detecting the change points between these "unstable" and "more stable" area. It is simply because, as discussed previously, using SSeqV in proposed approach can capture the changes of longer state transitions in the long-run. That is, for example, if we apply SAX to discretize/symbolize STB data with 2 possible symbols, "a" and "b", Most likely, we will get more "***ab***" and "***ba***" transitions in "unstable" area and get more "***aa***" and "***bb***" in "more stable" area. In such cases, using proposed approach with SSeqV is certainly more sensitive to the changes of longer state transition than it with SV.

3.3 CONCLUSION

In Section 3, I proposed a novel process monitoring approach that help detect anomalies in the symbolic data—a simple temporal sequence data stream, which are the patterns of interests identified by the domain experts, pattern classification methods, or time-series representation techniques. I begin with the introduction of measures used to calculate the distance of sequence data. Then, the comparative experimental results are presented. The major contributions of proposed approach in this section are to:

1. *Introduce a novel approach used in process monitoring that helps detect the anomalies of a dynamic system from the point of views of both system change-point and long-term evolutionary system deviation.*

2. *Demonstrate that comparing stationary symbol probability distributions (SV) generated by Google's PageRank algorithm instead of the discrete probability distributions from the frequency of symbols can maximize the information divergence, especially when measuring an unstable system with frequent state transitions.*

3. *Present that General Jensen-Shannon Divergence (D_{GJS}) outperforms other measures in terms of the accuracy of system changes/deviations detection.*

4. *Show that the significant threshold of the General Jensen-Shannon Divergence can be used as a criterion to determine the system long-term anomalies and short-term abrupt changes.*

5. *Prove that using Stationary Sequence Probability Vectors (SSeqV) could improve the accuracy of detection only when the system we monitor shows the changes of longer system state transitions (sequence patterns).*

In addition, the roles of four important properties (i.e. Boundedness, Symmetry, Generalizability, and Weightability) of a similarity/distance measure used in the assessment of system deviation are discussed here. The D_{GJS} is proved to be an outstanding measure to monitor system dynamics and

assess the significance of deviation in probabilistic manner. The combination of D_{GJS} and SV as the measure in the monitoring system is also proved to outperform others.

The proposed approach is particularly used to detect the anomalies of the system we monitor. However, analyzing a simple temporal sequence does not always provide us much information about the hidden dynamics of a system behind the scene. For some applications to predict the future sequence patterns so as to further support decision making, we need more generalized probabilistic modeling techniques. In Section 2, I introduced the concept of generating and identifying concurrent state-observation sequences using pattern classification methods. As we can learn from the classification models that there is a connection between the state and observation sequences, we can employ the hidden Markov models (HMM) to infer the sequence patterns as discussed in Section 2. In the next section, I consider a classification tree-based hidden Markov model that copes with concurrent state-observation sequences data, which combine CART and HMM to provide intuitive and useful definitions of sequence pattern dynamics to decision makers.

4.0 PROCESS MODELING USING CLASSIFICATION TREE HIDDEN MARKOV MODEL

Consider we have a state-observation sequence data that indicates the development of a dynamic system as discussed in Section 2. Suppose there is a certain connection between state and observation sequences identified by pattern classification techniques or domain experts, we could infer the future or most relevant sequence changes by learning discrete-time hidden (Semi-) Markov models (HMM/HSMM). In this section, I introduce my approach that builds a classification tree-based HMM (CTHMM) and its extension (CTHSMM), how to choose the best CTHMM/CTHSMM, and evaluate model applicability and limitation.

4.1 BUILDING CLASSIFICATION TREE HIDDEN MARKOV MODELS (CTHMM)

To build an HMM, we first need to define states and observations in a dataset. I assume that the observations are from possible values of a categorical dependent/target variable of a dataset, but the states are not directly-observable (hidden). Consider previous weather system data again as an example, Table 4 is an artificial weather data consisting of one categorical variable, Weather Condition, and one continuous variable, Temperature. Suppose Weather Condition is a dependent variable as the observation and Temperature is an independent variable that we will use as state in the HMM. Apparently, the Temperature is continuous and we need state splitting rules to discretize it as categorical states.

Table 4: Weather system data without state splitting rules

Weather Condition	Temperature (F)
Sunny	78
Sunny	60
Cloudy	45
Rainy	42

Observation:
Weather Condition (Sunny, Cloudy, Rainy)

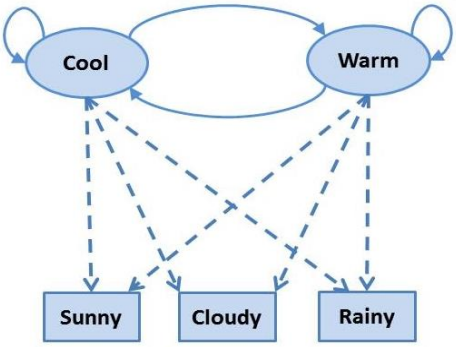
State:
Temperature(???)

However, we can still define "common sense" state splitting rules to create an HMM. Let's say if the Temperature is less than 50, we call it state **Cool**. On the other hand, if the Temperature is greater than or equal to 50, we call it state **Warm**. Then, we have 3 observations and 2 states to create an HMM as shown in Table 5.

Table 5: An HMM with 2 states and 3 observations

State **Cool: IF** Temperature < 50
State **Warm: IF** Temperature >=50

Weather Condition	Temperature
Sunny	Warm
Sunny	Warm
Cloudy	Cool
Rainy	Cool

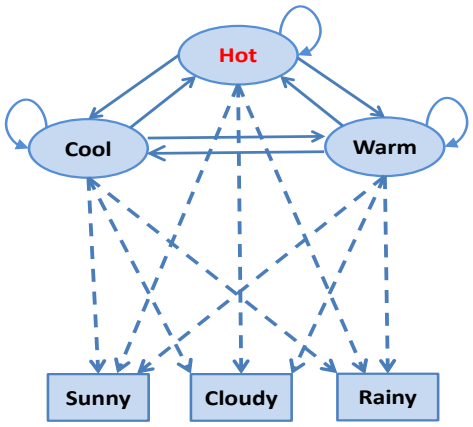


The state splitting rules look reasonable, but one may argue that the following rules with one more state called **Hot** would make it more practical. Then, we will have a new HMM with 3 states and 3 observations as shown in Table 6:

Table 6: An HMM with 3 states and 3 observations

State **Cool: IF** Temperature < 50
State **Warm: IF** Temperature >=50
AND Temperature < 70
State **Hot: IF** Temperature >=70

Weather Condition	Temperature
Sunny	Hot
Sunny	Warm
Cloudy	Cool
Rainy	Cool



The question here is which model we should choose. Also, is Temperature the best variable to determine the Weather Condition? An expert in meteorology may say Atmospheric Pressure is more important. Perhaps we should consider more than one variable that may affect the Weather Condition. So, what are the best state splitting rules and the number of states? Obviously, we need a state splitting method to generate commonly acceptable state splitting rules. The method must be able to help us divide data points into states and also choose more significant independent variables that determine/classify the observations. Below I summarize 3 possible solutions to select state splitting methods:

1. Using domain expert's knowledge. Based on their knowledge, the experts in most of the fields know what variables are significant and may have some existing classifications for those variables. For example, market researchers know how much annual income would be considered a "high-level" income and what the best variables are to predict whether a customer will buy their products. In addition to this, doctors should know what body temperature is critical for a patient and what vital signs are the most important to determine whether a doctor should send a patient to an intensive care unit (ICU).

2. Using supervised pattern classification methods. Some existing pattern classification methods, such as Linear Discriminant Analysis and Naïve Bayesian classification (Han, Kamber, & Pei, 2011), are widely used in various fields of studies and proven to be effective. The identified patterns and predicted classes generated by these methods can be used as states and observations to create HMM.

3. Using combined strategies. In most situations, noisy data is inevitable wherever the data comes from. Redman (Redman, 1998) has suggested that we should expect at least 1-5% error rate of data. Just applying the pattern classification methods often results in bad classifications. Data preprocessing, such as imputation, cleansing, and adaptation based on domain experts' knowledge, to improve data quality before applying those classification methods, can help improve the accuracy of classification (Redman, 1998). In most cases, we can have more reliable classification models by applying both the experts' knowledge and pattern classification methods.

Therefore, before creating HMM, we need to find a state splitting (classification) method that can automatically choose independent variables, map these independent variables (predictors) into states, and then generate all the required HMM parameters. To achieve this, we should first know what classification problems we are dealing with. Here, I assume that we have a categorical dependent variable and multiple continuous and categorical independent variables as

$$T = f(X_1, X_2, \dots, X_n, A_1, A_2, \dots, A_m)$$

where $\boldsymbol{T}$ denotes the categorical target/response variable, $\boldsymbol{X}$ and $\boldsymbol{A}$ denote continuous and categorical variables as predictors. We believe there are some connections between the target and predictors. Also, because we already have the outcome/observation, (i.e. the target variable with possible values), the state splitting methods can learn from these outcome during model training,

which is also called a supervised learning process. Taking the previous weather data as an example, the classification model is:

$$Weather\ Condition = f(Temperature, Atmospheric\_Presssure)$$

Here, Weather Condition is a categorical dependent variable, whereas Temperature and Atmospheric Pressure are the continuous independent variables. In the example, we believe that Temperature and Atmospheric Pressure can determine Weather Condition and we can then use pattern classification methods to learn models from data to describe the connection.

In this dissertation, I suggest to use Classification Tree induction as state splitting method. Classification Tree originated from the Classification and Regression Tree (CART) algorithm (Breiman et al., 1984). The tree induction process iteratively splits the tree (and data points) into two subsets of trees and creates the binary tree structured classifiers. The process keeps selecting descendant subsets to minimize the *Impurity* (to maximize the purity) of these subset data, which means the data in subsets are usually purer than those in parent subsets (Breiman et al., 1984). Here, the Impurity means the degree of dispersion for observation probabilities in a tree node. For instance, in the weather data, we have 3 possible observations (classes) for Weather Condition. We can construct trees by repeatedly splitting to minimize the Impurity of Weather Condition in the data. Figure 31 shows an example of the splits (growth) of the trees for the weather system data.

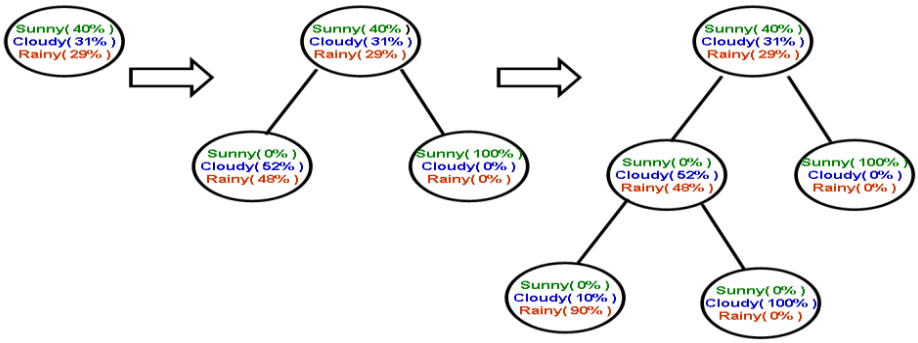


Figure 31: A tree splitting/growing example

We can see the tree has only one root node with the probabilities for Weather Condition (Sunny 40%, Cloudy 31%, Rainy 29%) initially. As the tree grows into 3 nodes, the probabilities of Weather Condition in the right leaf node are Sunny 100%, Cloudy 0%, and Rainy 0%. The Impurity of Weather Condition decreases, because the probability of Sunny increases to 1 and becomes a dominant observation. Also, in the tree with 5 nodes, the impurity decreases again since there are also dominant observations with higher probabilities in the bottom leaves.

To measure the Impurity of subsets of a data *D*, Classification Tree uses *Gini Diversity Index* (*Gini*) and considers a binary split for each variable *V* as shown in Eq. 22:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$Gini_v(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

$$\Delta Gini(V) = Gini(D) - Gini_v(D) \qquad \text{(Eq. 22)}$$

where *m* is the number of classes (observations), and p_i is the percentage of tuples that belongs to class *i* in data *D*. $|D|$ is the number of data in current node, whereas $|D_1|$ and $|D_2|$ are the number of data points in the subset D_1 and D_2 (two child notes). The split criterion is to choose a variable

V and a split point (for continuous variables) or a nominal value (for discrete variables) that maximizes $\Delta Gini(V)$. That is, to find the minimum $Gini_v\ (D)$.

The following is an example with 3 categorical variables for previous weather data to show how a tree chooses the splitting variables. Suppose we have data as shown in Table 7.

Table 7: Weather data with 3 categorical variables

Weather Condition (Sunny, Cloudy, Rainy)	Temperature (Warm, Cool)	Atmospheric Pressure (High, Low)
Sunny	Warm	High
Sunny	Warm	High
Sunny	Cool	Low
Cloudy	Cool	High
Cloudy	Warm	Low
Sunny	Warm	High
Cloudy	Warm	Low
Rainy	Cool	Low
Rainy	Cool	High
Cloudy	Warm	High

There are 3 possible observations (Sunny, Cloudy, Rainy) for Weather Condition as dependent variable. Both Temperature and Atmospheric Pressure have 2 possible values, (Warm, Cool) and (High, Low) respectively. To find the first split, we need to calculate the *Gini* for complete data and also *Gini* for two different subsets of data divided by two different values for each independent variable (Temperature and Atmospheric Pressure). Then we select the variable that maximizes the difference, $\Delta Gini(V)$, as

$$Gini(Weather) = 1 - \left(\frac{4}{10}\right)^2 - \left(\frac{4}{10}\right)^2 - \left(\frac{2}{10}\right)^2 = 0.64$$

$$Gini_{Temperature}(Weather) = \frac{4}{10}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) + \frac{4}{10}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) + \frac{2}{10}\left(1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2\right)$$

$$= 0.15 + 0.15 + 0 = 0.3$$

$$\Delta Gini(Temperature) = 0.64 - 0.3 = 0.34$$

$Gini_{Atmospheric\ Pressure}(Weather)$

$$= \frac{4}{10}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) + \frac{4}{10}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) + \frac{2}{10}\left(1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2\right)$$

$= 0.15 + 0.2 + 0.1 = 0.5$

$\Delta Gini(Atmospheric\ Pressure) = 0.64 - 0.5 = 0.14$

The result shows that Temperature has higher $\Delta Gini$, which means it can decrease the impurity of data more than Atmospheric Pressure can do. As a result, we select Temperature as the first split to create a tree as shown in Figure 32:

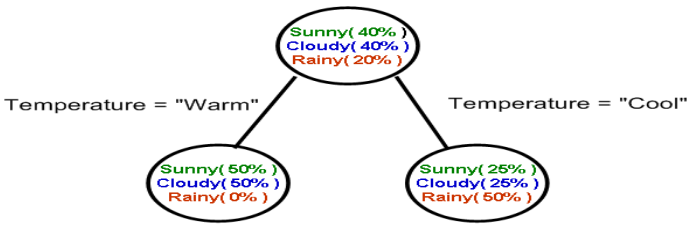


Figure 32: Weather system classification tree with one split

The tree can continue splitting and it iteratively chooses a variable and splitting criterion that has higher $\Delta Gini$. Obviously, we can keep the tree growing (splitting) until all the leaves are pure (have an observation with 100%). However, more splits means higher complexity of the tree. It also results in characterizing too much detail on a particular dataset. In order to solve this problem and obtain optimal trees, CART provide two types of pruning methods—*Pre-pruning* by halting the tree construction early and *Post-pruning* by pruning subtrees from a fully-grown tree using *cost-complexity pruning* algorithm. By setting a minimum number of data points in a node as a threshold, Pre-pruning can help decide whether the tree should further split at a given node. However, it is difficult to choose an appropriate threshold. As Breiman et al (Breiman et al., 1984) and Quinlan (Quinlan, 1993) indicate, "too high a threshold can terminate division before the

benefits of subsequent splits become evident, while too low a value results in little simplification". That is, high thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification(Han et al., 2011). I will discuss this issue when we consider HMM generation later in this section.

The Post-pruning method is more popular and commonly used in various decision tree approaches. The cost-complexity post-pruning algorithm in CART is to find the balance between tree splitting cost and tree complexity. Here, the tree splitting cost is *Misclassification Rate* (MR) of the tree, whereas tree complexity is the number of leaves of the tree. The cost complexity of a tree is a function of the number of leaves in the tree and the MR of the tree. MR is simply the percentage of misclassified data points for a tree model. I will provide more detail about MR in the next section. The cost-complexity *Cost* is the number of pruned leaves over the increased MR of the tree as shown in Eq. 23.

$$Cost = \frac{MR_{pruned} - MR_{orig}}{|leaves_{orig}| - |leaves_{pruned}|} \qquad \text{(Eq. 23)}$$

The MR_{pruned} and MR_{orig} denote the MR of the pruned and original tree. The $|leaves_{pruned}|$ and $|leaves_{orig}|$ denote the number of leaves in the pruned and the original trees. The process of the algorithm is to find a subtree that has the minimal cost if it is pruned. The new (pruned) tree can also be pruned until the tree model has only one node—the root node. Figure 33 shows how the process works.

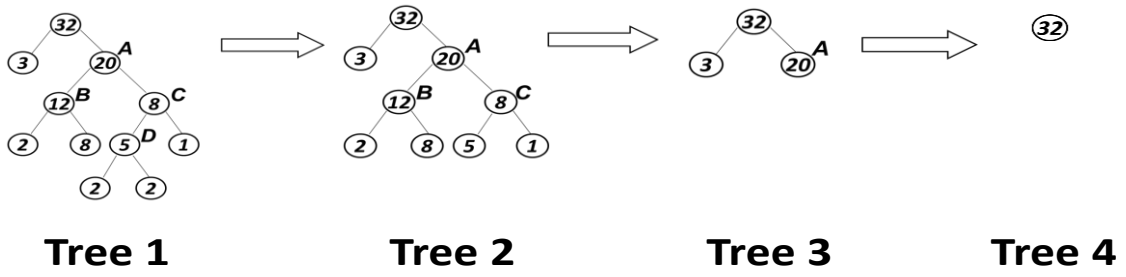


Figure 33: The process of post-pruning a tree

In the Figure 33, I assume the total number of data record is 100. The numbers in each node denote the number of misclassified data record. In the beginning, we have a fully-grown tree (Tree 1) with 4 subtrees (A, B, C, and D) and 6 leaves. The cost-complexity algorithm first calculates the cost for each of the subtrees if they are pruned, then prunes the subtree with minimal cost. In Tree 1, the cost for each of the subtrees is:

$$Cost_A = \frac{\frac{(3+20)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-2} = 0.0125$$

$$Cost_B = \frac{\frac{(3+12+2+2+1)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-5} = 0.02$$

$$Cost_C = \frac{\frac{(3+2+8+8)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-4} = 0.015$$

$$Cost_D = \frac{\frac{(3+2+8+5+1)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-5} = 0.01$$

The cost of subtree D is of the minimal cost, so we first prune subtree D to get Tree 2. Similar to the previous calculation, subtree A is of the minimal cost in Tree 2 as:

$$Cost_A = \frac{\frac{(3+20)}{100} - \frac{(3+2+8+5+1)}{100}}{5-2} = 0.0166$$

$$Cost_B = \frac{\frac{(3+12+5+1)}{100} - \frac{(3+2+8+5+1}{100}}{5-4} = 0.02$$

$$Cost_C = \frac{\frac{(3+2+8+8)}{100} - \frac{(3+2+8+5+1)}{100}}{5-4} = 0.02$$

So, we next prune subtree A to get Tree 3. We can always continue to prune the tree until a tree model has only root node (e.g. Tree 4) to get all the possible tree models. Here I call all the trees, including fully-grown and only root node trees, as *pruned-trees*. These pruned-trees (total 4 pruned-trees in Figure 33) with different numbers of leaves will be evaluated and used in the HMM model generation.

The next step is to use these pruned-trees to build the HMMs. In a tree, each non-leaf node in a tree represents a splitting variable and criteria, whereas each leaf node contains the probabilities of all possible classes (observations). The class with maximum probability is used as the representative class/label in each leaf node to minimize the MR of the tree model. The result of classification tree induction is used to create the state splitting (definition) rules. A leaf node represents a state with the probabilities over observations, and the path to a leaf contains the rules of each state. Those state definitions are simple IF-THEN rules. For example, Figure 34 shows how a tree splits the data into 3 states and provides IF-THEN rules for those states in previous weather system data.

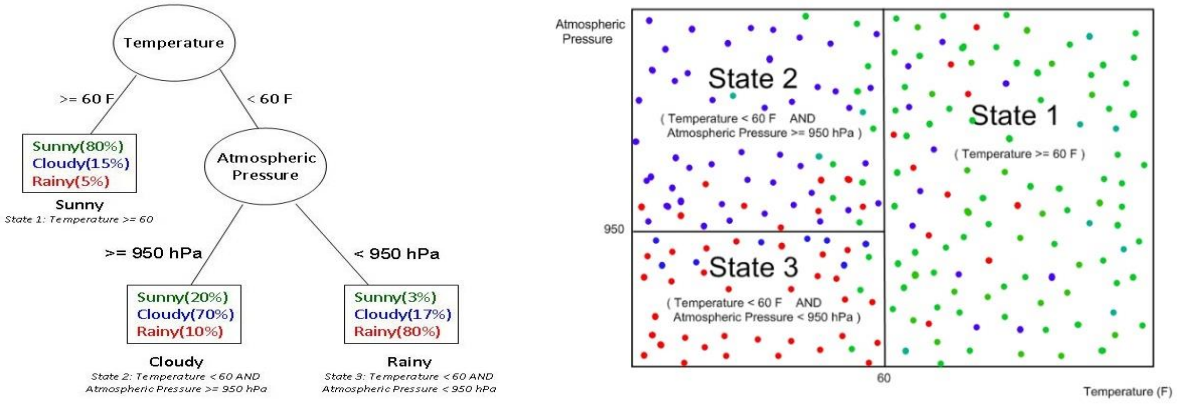


Figure 34: A tree provides state splitting rules to divide data into 3 states

Given state numbers, the leaves are used as states, and the target variable (i.e. Weather Condition) is used as observation in my proposed tree-based HMM generation. Based on the tree model in Figure 34, we can create state splitting rules for all the states and the observation/emission matrix of the HMM as shown in Table 8.

Table 8: State splitting rules and created observations matrix

State **1: IF** Temperature >= 60 F
State **2: IF** Temperature < 60 F
AND Atmospheric Pressure >= 950 hPa
State **3: IF** Temperature < 60 F
AND Atmospheric Pressure < 950 hPa

	Sunny	Cloudy	Rainy
State 1	0.80	0.15	0.05
State 2	0.20	0.70	0.10
State 3	0.03	0.17	0.80

We can then convert each data point into a state with number as shown in Table 9.

Table 9: Weather data with mapped state number

Weather Condition	Temperature (F)	Atmospheric Pressure (hPa)	State Number
Cloudy	62	982	*1*
Rainy	50	950	*2*
Rainy	48	930	*3*
Cloudy	55	950	*2*
Sunny	65	980	*1*

Thus, we have the temporal sequences of states numbers and observations (Weather Condition). Instead of iteratively re-estimating state transition matrix using *Expectation-Maximization*-like algorithm (e.g. *Baum-Welch* algorithm) (Rabiner & Juang, 1986), we can simply calculate the relative frequency of state transitions to obtain the state transition probabilities of the tree-based HMM from the sequences of states.

In the HMM generation process, I use leaf nodes from pruned-trees as states. The number of states is the number of leaves in a tree, but the initial probabilities of these states are not equal. They are determined by a pre-pruning threshold—the minimum number of data points in a leaf node (*minbucket*). By setting the pre-pruning threshold as a certain percentage of training data, we can decide the minimum initial probability of each state. For example, we can define the threshold as 1%, 3%, or 5% of the number of total training data so that the initial probabilities of states from a tree will not be lower than the threshold we set. Consider the previous weather data in Figure 34 again. Figure 35 shows the tree with one more split (leaf) and how the tree divides the data space.

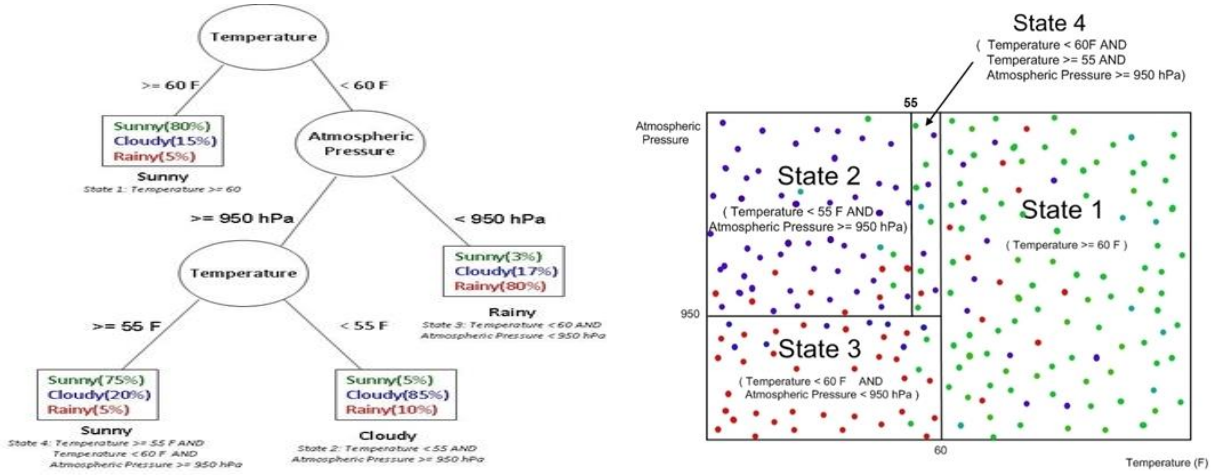


Figure 35: The tree with one more state to divide data into 4 states

We are not sure whether this "one more split" is good, but apparently we can keep the tree splitting until all the leaves are pure as we discussed. However, if a tree has too many splits and characterizes too much detail of the noise of data, it may cause overfitting the training data and result in a bad tree with too many number of states. Therefore, we can set the threshold depending on what we need since the threshold is also the minimum initial probability of each state. In the later experiments, I set the threshold *minbucket* as 1% of the total number of training data records, as I am only concerned about a state with the probability more than 1%. I will also discuss the role of *minbucket* in the later sections about model selection.

Using Classification Tree learning as state splitting method, we can generate all required model parameters for HMM. However, we may have several classification trees with different pruning methods, and these tree models create different HMMs. What is the best tree-based HMM? In the next section, we discuss how to evaluate these tree-based HMMs and the model selection.

4.2 MODEL SELECTION AND EVALUATION

A state generated by the splitting rules from a tree is defined by a combination of independent variables with a range of values, which also represents the current condition (state) of a system. An HMM with only one state means the system never changes its status, because the state stands for all possible conditions. On the other hand, an HMM that has too many states is not useful, because most of states have low probabilities and the system continuously transits back and forth among states. In addition, the probabilities of observations for a state also play a significant role in model selection. A state with a dominant observation means there are strong connection between the state and observation. On the contrary, a state with virtually equally-probable observations means there is almost no connection between the state and observations. Such HMMs

do not help us understand the system dynamics and the connection between states and observations. Therefore, we need methods and criteria to evaluate generated trees and decide which tree with how many states is the best. In this section, I discuss the model evaluation in terms of the accuracy of hidden state splitting and the accuracy of Viterbi path prediction. In the end of this section, I also introduce the model selection steps and measures to help understand the generated models.

4.2.1 Hidden State Splitting and Maximum Mutual Information Estimation

The accuracy of state splitting is about how well the connection between the states and observations. As Classification Tree of CART is used here to generate the model parameters for HMMs, several measures commonly used in the evaluation of supervised pattern classification methods are also applicable. The most straightforward way is to construct a confusion matrix as shown in Table 10 and see how many predicted observations match the actual observations in the data.

Table 10: Confusion matrix

		Actual Class	
		Yes	**No**
Predicted Class	**Yes**	True Positive (TP)	False Positive (FP)
	No	False Negative (FN)	True Negative (TN)

In Table 10, the total number of data points is $T = TP + FP + FN + TN$. The Accuracy of a tree model is the percentage of total correctly-predicted data point, and the Misclassification Rate (MR) is one minus the Accuracy as:

$$Accuracy = \frac{TP + TN}{T}$$

$$MR = 1 - Accuracy = 1 - \frac{TP + TN}{T}$$

Taking the previous weather data as an example, we can create a confusion matrix as shown in Table 11.

Table 11: Confusion matrix for the weather data

		Actual Class		
		Sunny	**Cloudy**	**Rainy**
Predicted Class	**Sunny**	A_{SS}	A_{SC}	A_{SR}
	Cloudy	A_{CS}	A_{CC}	A_{CR}
	Rainy	A_{RS}	A_{RC}	A_{RR}

There are 3 classes/observations (Sunny, Cloudy, Rainy), so we have a 3 by 3 confusion matrix. The Accuracy and MR are:

$$\text{Accuracy} = \frac{A_{SS} + A_{CC} + A_{RR}}{\sum_{i,j \in (S,R,C)} A_{ij}}$$

$$\text{MR} = 1 - \text{Accuracy}$$

Nevertheless, the Accuracy and MR are simply the percentage of correctly and incorrectly classified data respectively. They are not reliable measures to evaluate classification models, because they do not take the chance agreement (correctly predicted by chance) into consideration (Cohen, 1960). For example, suppose the total number of data points is 100, the following 2 confusion matrices in Table 12 have the same MR, 0.5.

Table 12: Two confusion matrices with the same misclassification rate

		Actual Class		
		Yes	No	
Predicted Class	Yes	25	25	50
	No	25	25	50
		50	50	100

Matrix A (MR = 0.5)

		Actual Class		
		Yes	No	
Predicted Class	Yes	25	1	26
	No	49	25	74
		74	26	100

Matrix B (MR = 0.5)

The trees that create these 2 matrices are not necessarily of the same accuracy of classification, because Matrix A may be from a random classification tree and the matched data may be just by chance. The matched and unmatched data points are equally distributed, which means the tree may have nothing to do with the training data and is not helpful to the classification. Thus, Accuracy/MR is not a robust measure to evaluate a classification model. Let's consider another measure, Cohen's Kappa coefficient(*K*) (Ben-David, 2008). *K* takes into account the chance agreement and correct the degree of agreement by subtracting the portion of the counts that may be attributed to chance (Ben-David, 2008). Table 13 shows the interpretation of *K* and how I calculate *K* from a confusion matrix.

Table 13: Cohen's Kappa and its interpretation

$$P_o = Observed\ Accuracy = 1 - MR = \frac{TP + TN}{T}$$

$$P_e = Expected\ (chance) Accuracy$$

$$= \frac{\frac{(TP + FN) * (TP + FP)}{T} + \frac{(FP + TN) * (FN + TN)}{T}}{T}$$

$$K = \frac{P_o - P_e}{1 - P_e}$$

K	Interpretation
< 0	Poor agreement
0.0 — 0.20	Slight agreement
0.21 — 0.40	Fair agreement
0.41 — 0.60	Moderate agreement
0.61 — 0.80	Substantial agreement
0.81 — 1.00	Almost perfect agreement

The *K* for the Matrix A and B in Table 12 are 0 and 0.1873 respectively. The tree that creates Matrix B has higher accuracy than the one that creates Matrix A in terms of Cohen's Kappa

coefficient. *K* is between -1 and 1. Landis and Koch (Landis & Koch, 1977) recommend that *K* should exceed 0.7 before doing further data analyses.

However, if we keep the tree splitting and set the pre-pruning threshold to allow just 1 data point in a leaf node, both *K* and Accuracy will increase to 1, because in each state/leaf node, all the data points belong to a class/label. In this case, there is no misclassified data point. As we discussed before, it will also result in overfitting the training data and getting a useless tree, which means the tree has high accuracy of classification in terms of *K* but only works for the training data. Thus, we should also consider how a tree model learned from training data works for testing data when doing model selection.

Using V-fold Cross-validation could solve this problem. V-fold Cross-validation is commonly used for assessing how the result of classification models will generalize to other independent datasets. As shown in Figure 36 , it partitions a dataset into V subsets and uses one subset as testing data and others as training data. Then, training and testing is performed V rounds for cross-validation to calculate average MR for all different partitions.

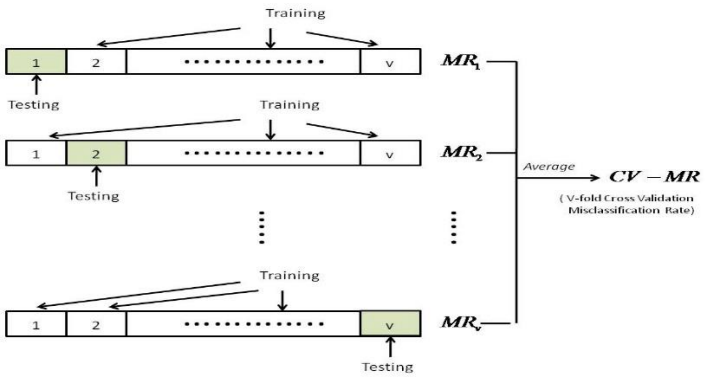


Figure 36: V-fold cross-validation misclassification rate

Breiman (Breiman et al., 1984) and Kohavi (Kohavi, 1995) suggest that using 10 to 20 folds is reasonably good. Here, we only consider 10-fold Cross-validation Misclassification Rate (CV-MR) in later experiments.

CV-MR seems like the best measure for model selection, but it is not necessarily the best choice for my proposed approach. We are going to use the selected tree models as our state splitting methods to generate HMMs. The parameters for each HMM will be used to solve the HMM problems and then help us understand the underlying connections between states and observations. Again, a model with too few (e.g. just one state) or too many states is useless for us, because an HMM with too few states provides less information, whereas an HMM with too many states results in a complicated model and may cause an unstable HMM. To solve this problem, we should consider what would be acceptable model parameters—the state transition matrices and observation matrices. Let us start by considering the *stability* of an HMM. In Table 14, there are one 2 by 2 state transition matrix (Matrix A) from a tree and two 3 by 3 state transition matrices (Matrix B and C) from the same tree but both with one more split.

Table 14: A state transition matrix with one more different splits

	State 1	State 2
State 1	1/2	1/2
State 2	1/2	1/2

Matrix A

	State 1	State 2	State 3
State 1	1/3	1/3	1/3
State 2	1/3	1/3	1/3
State 3	1/3	1/3	1/3

Matrix B

	State 1	State 2	State 3
State 1	1/3	1/3	1/3
State 2	1/3	1/3	1/3
State 3	0.99	0.005	0.005

Matrix C

The "one more split" provides one more state for Matrix A, so we have State 3 in Matrix B and C. However, this one more split in Matrix B is not good, because all state transitions are still equally-probable. Besides, the probabilities from State 3 to any states are equal, which means the system

is also in an unstable status and will transit to any states arbitrarily if the system is in State 3. On the other hand, the system is very likely to transit from State 3 to State 1 in Matrix C, which means this one more state stabilizes the HMM and provide us a valuable information—if the system is in State 3, we are quite confident that the system will most likely move to State 1.

Likewise, we can use the same concept to evaluate the observation matrix. One difference is that we evaluate the *connection* between state and observation from the observation matrix instead of the *stability* of the HMM from the state transition matrix. Consider an example for previous weather system data again. In Table 15, there are one 2 by 3 observation matrix (Matrix X) from a tree and two 3 by 3 observation matrices (Matrix Y and Matrix Z) for the same tree with one more split.

Table 15: A observation matrix with one more different splits

	Sunny	Cloudy	Rainy
State 1	0.7	0.2	0.1
State 2	0.1	0.1	0.8

Matrix X

	Sunny	Cloudy	Rainy
State 1	0.7	0.2	0.1
State 2	0.1	0.1	0.8
State 3	0.33	0.33	0.34

Matrix Y

	Sunny	Cloudy	Rainy
State 1	0.7	0.2	0.1
State 2	0.1	0.1	0.8
State 3	0.005	0.99	0.005

Matrix Z

We can see a similar situation in these matrices. In Matrix Y, if the system is in State 3, we will only know that the weather conditions are approximately equally probable, which means State 3 may have nothing to do with the weather conditions. Conversely, State 3 in Matrix Z provides us more information, because we know if the system is in State 3, the weather condition will most likely be Cloudy.

We can evaluate proposed classification tree-based HMMs (CTHMMs) by using the above concepts. Generally, we expect that the best HMM is the one that gives us more information—higher state stability and higher state-observation connection. Both of them can be measured from

the abovementioned matrices by the Mutual Information introduced in Section 3. Here, I define the Mutual Information MI_S and MI_O for state transition matrix and observation (emission) matrix of a discrete-time HMM respectively as:

$$MI_S = \sum_{i=1}^{s}\sum_{j=1}^{s} P(S_i, S_j) \log_2 \frac{P(S_i, S_j)}{P(S_i)P(S_j)} \qquad \text{(Eq. 24)}$$

$$MI_O = \sum_{i=1}^{o}\sum_{j=1}^{s} P(O_i, S_j) \log_2 \frac{P(O_i, S_j)}{P(O_i)P(S_j)} \qquad \text{(Eq. 25)}$$

where $P(S_i, S_j)$ and $P(O_i, S_j)$ are State-to-State and State-to-Observation joint probabilities, which can be obtained by multiplying the prior probabilities of the states $P(S_j)$ as defined by Bayes' theorem, i.e. $P(S_i, S_j) = P(S_i|\, S_j)\, P(S_j)$ and $P(O_i, S_j) = P(O_i|\, S_j)\, P(S_j)$. Take the matrices in Table 14 and Table 15 as examples, assuming that prior probabilities of all states are equal, we can compute the MI_S and MI_O for the matrices. The MI_S for the Matrix A, B, and C are 0, 0, and 0.3530 bits, whereas the MI_O for the Matrix X, Y, Z are 0.4184, 0.3090, and 0.8312 bits, which also suggests that CTHMMs with Matrix C and Matrix Z are better models. It is obviously that we could use MI_S and MI_O as the measures that help choose better proposed CTHMMs. Here, instead of using the aforementioned post-pruning method that generates a set of pruned-tree models, we can pre-prune the tree models by setting appropriate minimum numbers of records/data points in a leaf node (*minbucket*) to limit the tree growth as discussed in Section 4.1. That is, we consider classification tree-based HMMs (CTHMMs) with different numbers of states that have maximum MI_S and MI_O given the parameter *minbucket* as:

$$minbucket_{maxMI} = \underset{minbucket \in [\,1\,,\ |train\_data|\ ]}{\arg\max} MI(\ CTHMM(minbucket, train\_data\ )\ ) \qquad \text{(Eq. 26)}$$

where *train_data* and | *train_data* | are the training dataset used to build the model and the number of records/data points in the dataset. For each CTHMM, we can compute an MI_S and an MI_O. The

goal is to choose an *minbucket* (between 1 and | *train_data* |) that maximizes the objective function (MI) that compute MI_S and an MI_O of an CTHMM. Here, the process of obtaining best CTHMM parameters is called *Maximum Mutual Information Estimation* (MMIE). Also note that we may obtain 2 different $CTHMM_{\max\ MI}$ here, as they are selected based on maximum MI_S and MI_O.

The pre- and post- pruning methods adapted by proposed approach provide us a better way to generate CTHMMs with appropriate number of states. The cost-complexity algorithm helps post-prune a tree and obtain a set of pruned-trees, whereas $CTHMM_{\max\ MI}$ help pre-prune a tree and select the best trees with maximum state stability and state-observation connection. However, as discussed in previous sections, the selected CTHMMs will be used to predict the future state dynamics of the system we monitor. So far, we are not sure whether selected CTHMMs would result in the high accuracy of prediction when they are actually applied to the prediction of Viterbi path (Forney, 1973) with different state sequence lengths. In next section, I discuss this application of proposed CTHMM and how to evaluate the accuracy of prediction.

4.2.2 Accuracy of Viterbi Path Prediction

Our aim is to choose the best CTHMM that provides us more information (the IF-THEN state definitions and the state dynamics) and higher accuracy of state sequence prediction. In previous section, we have discussed some measures used in evaluating the accuracy of state splitting for CTHMMs we created from training dataset. Here, we want to know how these models work when they are applied to testing datasets. We first use the state splitting rules learned from training dataset to convert the predictors of the testing dataset into states numbers as *presumable state sequence*, and then find the most probable state sequences, *predicted state sequence*, for the corresponding observation sequences from the testing dataset, which is the task to find the Viterbi

path solved by the Viterbi algorithm (Forney, 1973) as discussed in Section 2. Again, consider previous weather system data. Suppose we have an excerpt of testing data as shown in Table 16.

Table 16: The presumable states and predicted state for testing data

Weather Condition	Temperature (F)	Atmospheric Pressure (hPa)	Presumable State Number (by splitting rules)	Predicted State Number (Viterbi Path)
Rainy	55	945	***3***	***3***
Rainy	60	950	***1***	***2***
Cloudy	59	955	***2***	***2***
Cloudy	64	955	***1***	***1***
Sunny	72	970	***1***	***1***
Cloudy	65	960	***1***	***2***

The presumable state numbers are mapped from predictors (Temperature and Atmospheric Pressure) based on the state splitting rules in Table 8, whereas the predicted state numbers are generated by the Viterbi algorithm given the weather conditions (observation sequence with length = 6 in Table 16). Apparently, we can evaluate the accuracy of prediction by calculating the percentage of matched state numbers, which I call *Hit Ratio* in this dissertation. Also, as the hit ratio is a simple measure that may overestimate the accuracy with the chance of hits, I also consider another measure called *Longest Matched Run Length Ratio* (LMRL Ratio), which denotes the ratio of longest matched run length of state numbers to the Viterbi path length. Figure 37 illustrates how to compute the hit ratio and LMRL ratio for the data shown in Table 16 given that the predicted length is 3 states.

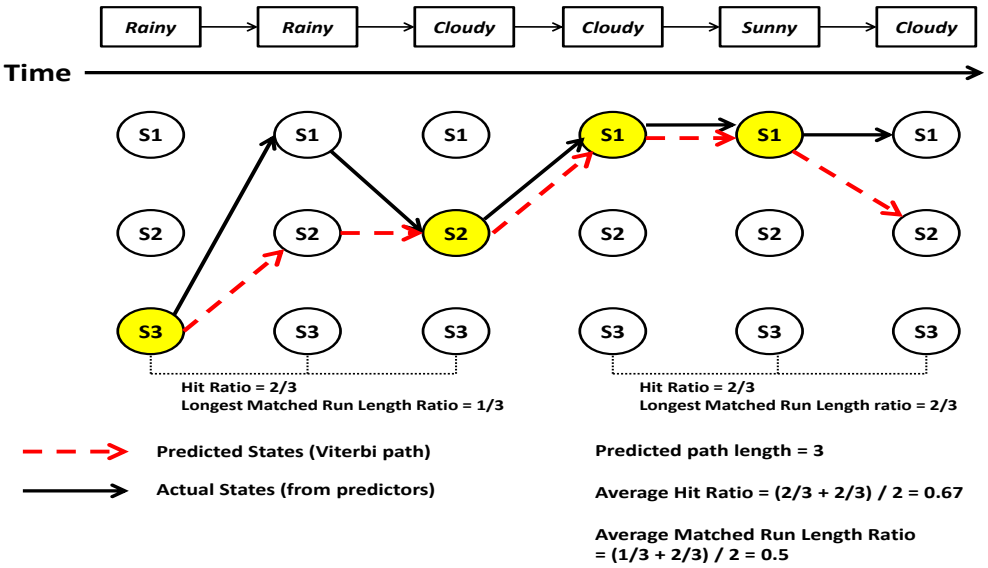


Figure 37: Actual and Predicted States with Average Hit Ratio and LMRL Ratio for Table 16

Note that the most probable state sequences, Viterbi path, are computed with 2 observation sequences with length = 3, i.e. (Rainy → Rainy → Cloudy) and (Cloudy → Sunny → Cloudy). In Figure 37, there are 2 matched state numbers in both 2 sequences (both with 3 states), but the longest run length of the matches are 1 and 2 respectively. I then calculate the averages of the hit ratio and LMRL ratio, which will be considered as the accuracy of prediction and used as evaluation criteria in later experiments. Therefore, given an observation sequence (Rainy → Rainy → Cloudy), for example, we are 66.67% sure that the most probable state sequence is (S3 → S2 → S2), i.e. the Temperature might stay "< 60 F", but the Atmospheric Pressure might rise from "< 950 hPa" to ">= 950 hPa".

Again, we assess the accuracy of state splitting by using the measures introduced in previous section to choose the best CTHMM models that could give us more information, i.e. the number of states and the state definition rules. On the other hand, we evaluate selected CTHMM models by checking their accuracy of Viterbi Path predictions to see how well they perform in

terms of average hit ratio and LMRL ratio. It is certain that we should always select CTHMMs with high accuracy of prediction. However, these models may not always provide us more information about state transitions of the dynamic system we monitor. For example, an CTHMM with 2 possible states could result in highest accuracy of prediction, whereas an CTHMM with 10 possible states often lead to lower accuracy of prediction. Models with fewer states have too simple state definition rules that provide less information, as they involve fewer variables of predictors. On the other hand, models with many states and many state definitions may provide more information but complicated and incomprehensible rules. We should consider the model selection process a trade-off to find the balance between the accuracy of prediction and the complexity of state definition rules based on our applications. To conclude, I provide Figure 38 that summarizes the proposed CTHMM/CTHSMM model selection and evaluation process.

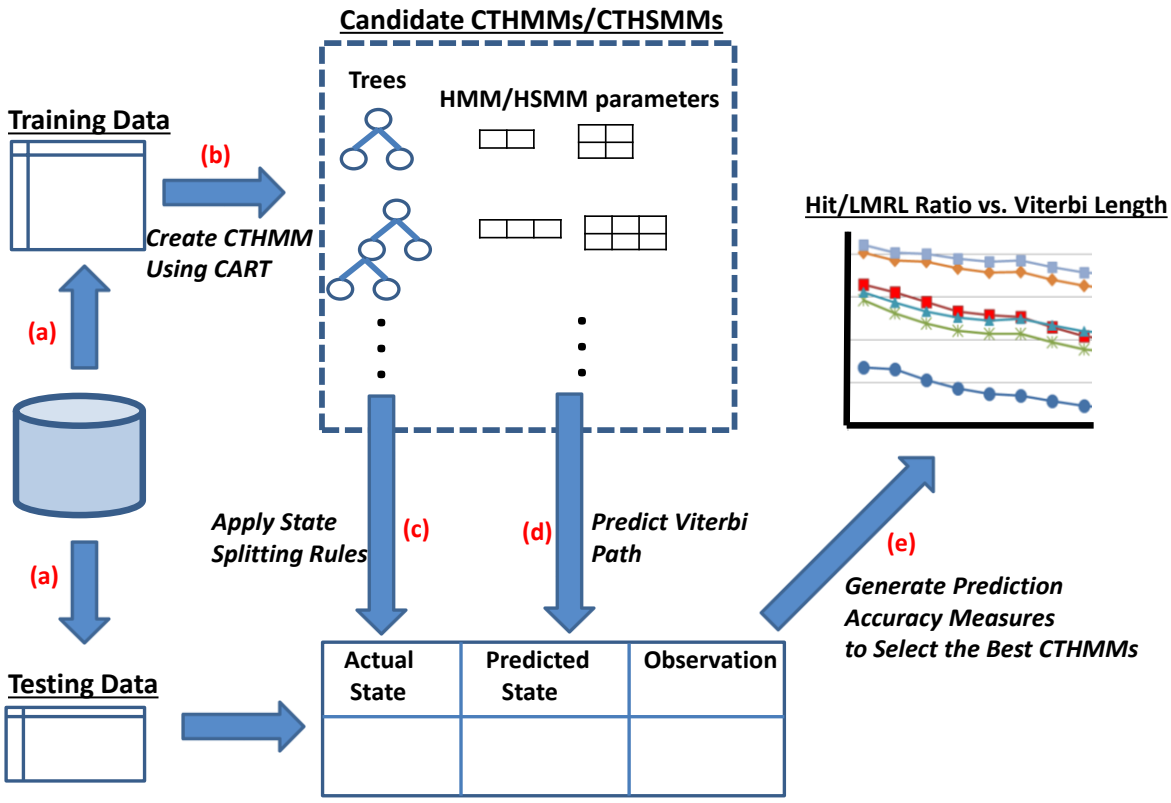


Figure 38: Proposed CTHMM model selection and evaluation process

The process in Figure 38 consists of the following steps:

(a) Divide data into two different parts as training and testing datasets.

(b) Use CART with adapted post-pruning (cost-complexity) and pre-pruning (MMIE) methods to learn candidate CTHMMs from the training dataset.

(c) Apply state splitting/definition rules to the predictors of the testing dataset to obtain presumable state sequences of testing dataset.

(d) Generate Viterbi paths (predicted state sequences) given the observation sequences from testing dataset.

(e) Calculate the averages of hit and LMRL ratios, create plots for model evaluation, and then select the best CTHMM that balances information richness/complexity and prediction accuracy based on real applications.

4.3 EXTENDING CTHMM WITH VARIABLE STATE DURATIONS-CLASSIFICATION TREE HIDDEN SEMI-MARKOV MODEL (CTHSMM)

In the recent decades, literature has indicated that modeling with HMM may be unrealistic and inaccurate when HMM is used in the applications that the state duration distributions (sojourning times) are different (Barbu & Limnios, 2008; Sansom & Thomson, 2001). In Section 4.1, I introduced my approach, CTHMM, which combines Classification Tree and typical HMM without taking into account the time/duration each state has spent. That is, I accepted the assumption of HMM that the state durations are all identical, which implies that the state durations are geometrically distributed (Barbu & Limnios, 2008). It is because, in this case, we consider the probability of spending continuous *m* times/steps in *i* state as:

$$d_i(m) = p_{ii}^{m-1}(1 - p_{ii}) \qquad \text{(Eq. 27)}$$

where $d_i(m)$ is state duration/sojourning time density and p_{ii} is the probability that state *i* transits to itself. We are modeling the probability that how many time (*m* steps) a system will take to "leave" state *i*. That is, for example, Figure 39 shows a simulation of state duration distribution given p_{ii}= 0.8.

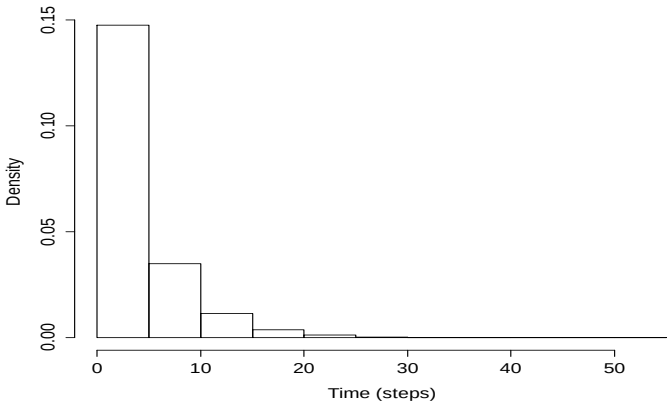


Figure 39: State duration distribution with $\boldsymbol{p_{ii}}$= 0.8

We can see that, most likely, the system will stay in the state for less than 20 steps (time units). Apparently, for most HMM applications, the distributions for states durations are not all in this particular "shape" (geometrically-distributed). Consider previous example of weather system data in Table 9 but with one more variable about state duration in hours as shown in Table 17.

Table 17: Weather data with state numbers and durations in hour

Weather Condition	Temperature (F)	Atmospheric Pressure (hPa)	State Number	Duration (Hour)
Cloudy	62	982	*1*	**2**
Rainy	50	950	*2*	**1**
Rainy	48	930	*3*	**1**
Cloudy	55	950	*2*	**2**
Sunny	65	980	*1*	**6**

It is obvious that the weather state dynamics may not vary that often (within a couple of hours). In other words, the changes of states may take longer than a few hours. The duration distributions of actual hidden weather states identified by state-splitting rules do not necessarily follow geometric

distribution. Therefore, in such cases, we need to extend proposed CTHMM so that it can take state durations into consideration and explicitly estimate the duration density $d_i(m)$ for each state.

Many researchers have proposed generalized versions of HMM that model with state durations, such as Hidden Semi-Markov Model (HSMM) (Barbu & Limnios, 2008) and Variable Duration HMM (VDHMM) (Chen, Kundu, & Srihari, 1995). Here, I consider extending my proposed CTHMM with HSMM to create a Classification Tree Hidden Semi-Markov Model (CTHSMM) by modifying the way to estimate the state transition probabilities. As discussed in Section 4.1, instead of using *Baum-Welch algorithm* (Barbu & Limnios, 2008) that iteratively re-estimates state transition probability matrix, the proposed CTHMM created the matrix by calculating the relative frequencies of state transitions, as those hidden states are clearly-identified by CART with IF-THEN state splitting rules. To build CTHSMM, however, I propose to estimate "in-state" (absorbing state) transition probabilities (p_{ii}) and "out-state" transition probabilities (p_{ij}, where $i \neq j$) separately. Consider a hidden semi-Markov model with 3 hidden states as an example, Figure 40 shows an example of the estimation of HSMM state transition probabilities.

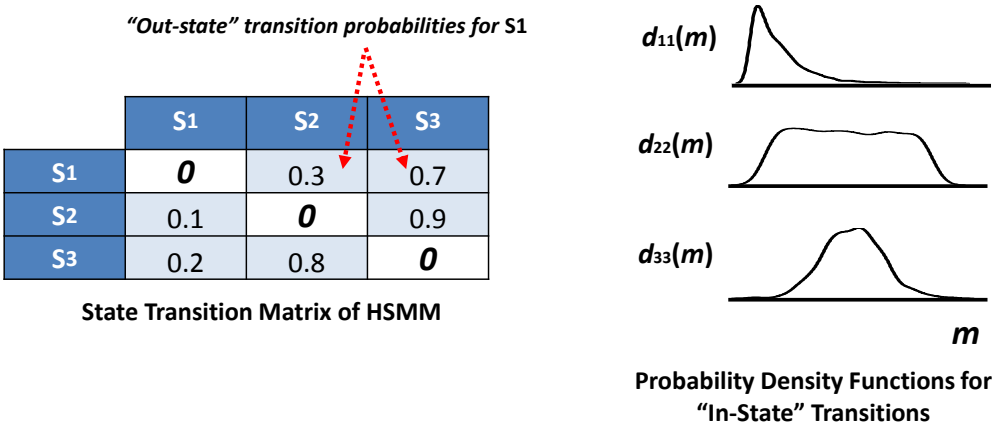


Figure 40: The estimation of state transition probability of HSMM

The "out-state" transition probabilities can be computed from the relative frequencies of those "out-state" transitions from the training dataset with identified presumable hidden states. Note that diagonal cells of the left matrix in Figure 40 must be zero, as we model/estimate the "in-state" transition probabilities separately and do not consider these absorbing/sojourning states. On the other hand, we can see the state durations (sojourning time) could vary from different states and are not necessarily geometrically-distributed. Also, as the hidden states are identified by the Classification Tree of CART, we may not have the information about the duration distribution for each state. Therefore, instead of making assumptions about the duration distributions, I propose to explicitly estimate the probability density function of each state duration from the training data by using Kernel Density Estimation (KDE) with Gaussian kernel smoother (Silverman, 1986). KDE is a non-parametric estimation technique widely used to estimate probability density function of a random variable. Let $\boldsymbol{x} = (x_1, x_1, \ldots x_n)$ and be i.i.d., KDE is here defined as:

$$\widehat{f_h}(x) = \frac{1}{n} K_h(x - x_i) \qquad \text{(Eq. 28)}$$

where K_h is a symmetric kernel function that integrates to 1. Also note that h is a free smoothing parameter to determine the bandwidth. Here in later experiments, I empirically use the rule-of-thumb estimation of the bandwidth (Silverman, 1986) as defined:

$$h = \left(\frac{4\widehat{\sigma}^5}{3n}\right)^{\frac{1}{5}} \qquad \text{(Eq. 29)}$$

where n is the sample size and $\widehat{\sigma}$ is the sample standard deviation. The estimated density functions for state durations are then used in the predictions of the Viterbi path (Forney, 1973) for different lengths (m) of sojourning/absorbing states.

With the transition probability estimation method shown in Figure 40 that can clearly addresses the problem of variable state duration distributions, the proposed CTHSMM is more

flexible and able to predict the most probable state transitions with variable time units given different lengths of observation sequences. Figure 41 shows an example of the aforementioned weather system dynamics with 3 hidden states. We can see that, given a sequence of weather conditions for total 18 hours, we can find the most probable hidden weather state transitions with state duration in hours (i.e. $S_1 \rightarrow S_2 \rightarrow S_3$ for 5, 4, and 9 hours respectively).

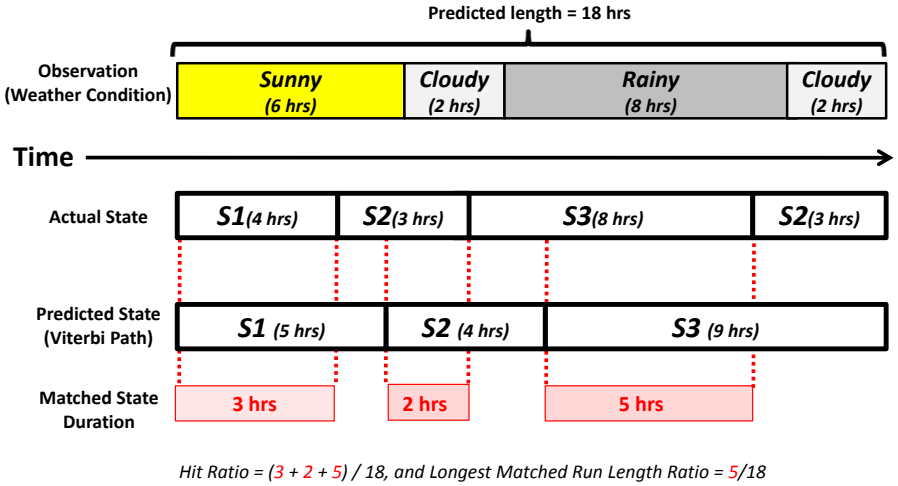


Figure 41: Hit and LMRL Ratios for Variable Lengths of Actual and Predicted States

Also note that the way to compute the aforementioned hit ratio and LMRL ratio discussed in Section 4.2.2 is now with the time units (hours). In Figure 41, the hit ratio is the percentage of total matched state durations, whereas the LMRL ratio is the percentage of longest matched state duration. Evidently, unlike CTHMM, CTHSMM is a more flexible and realistic probabilistic process model that learns system hidden states with variable state sojourning times to infer most probable/relevant system dynamics in temporal-probabilistic manners.

4.4 EXPERIMENTS

To demonstrate how the proposed approach works, I applied CTHMM/CTHSMM modeling process to a real-word dataset from Bedside Monitoring Systems that records patients' vital sign conditions. The dataset consists of 1 categorical dependent variable as a patient's current location (ICU or Floor), 1 continuous variable as the duration in hour, and 5 continuous independent variables as 5 vital signs for a patient: Diastolic Blood Pressure (DBP), Systolic Blood Pressure (SBP), Respiratory Rate per minute (RR), SpO2 Bedside Monitor (SPO2), and Temperature (Temp). There are 13,006 data point for total 359 patients who are children between 1 and 6 years old and were hospitalized in Children's Hospital of Pittsburgh (CHP) in 2008. Table 18 and Table 19 shows an excerpt of CHP data for a patient and the reference normal vital sign ranges for children aged between 1 and 6 years old [59 - 61] respectively.

Table 18: An excerpt of CHP dataset

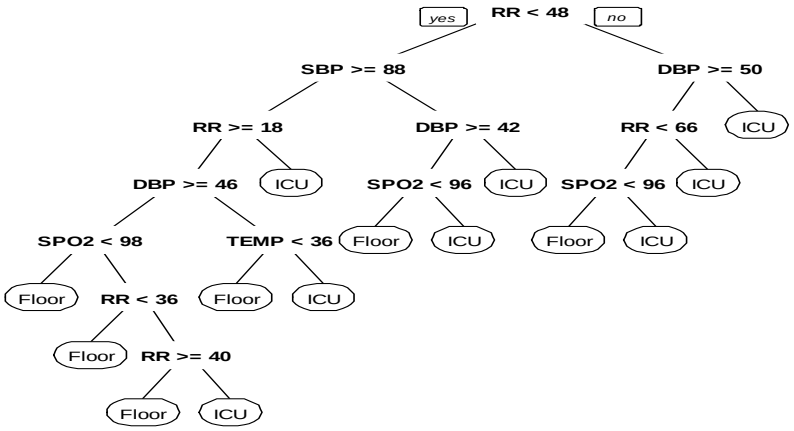
Diastolic Blood Pressure (mm Hg)	Systolic Blood Pressure (mm Hg)	Respiratory Rate (bpm)	SpO2 Bedside Monitor (%)	Temperature (C)	Location	Duration (Hour)
64	117	29	100	37.5	ICU	1
65	110	21	99	37.5	ICU	1
65	110	21	99	37.5	ICU	1
65	110	21	98	37.5	ICU	1
66	90	26	96	36.7	Floor	2
67	97	27	98	36.7	Floor	1
67	97	27	96	36.7	Floor	2
65	94	26	98	37.1	Floor	2
65	94	26	98	37.1	Floor	3
68	87	15	98	37.3	Floor	1

Table 19: Reference normal vital sign ranges for children aged 1-6

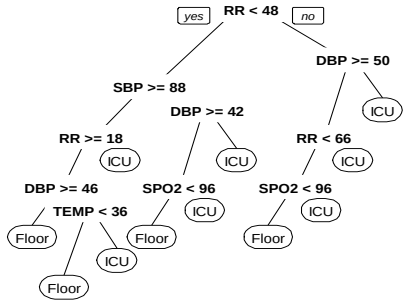
Diastolic Blood Pressure (mm Hg)	Systolic Blood Pressure (mm Hg)	Respiratory Rate (bpm)	SpO2 Bedside Monitor (%)	Temperature (C)
55 - 75	90 - 100	15 - 30	≥ 95%	36.33 - 37.56

These patients have previously reported with respiratory problems. The goal of using this dataset is to explore whether proposed approach can help doctors understand patients' vital sign pattern dynamics, evaluate patient respiratory complaint risk, and further improve hospital bed utilization rate. I first divided CHP data by randomly sampling approximately 70% patients from CHP data as a training dataset and the rest of 30% data are considered as a testing dataset. There are total 8,734 and 4,272 rows for training and testing datasets respectively.

The proposed CTHMM learning process is then applied to the training dataset to construct candidate CTHMMs. I first use the cost-complexity post-pruning algorithm to continuously prune a fully-growth tree model (with 1% of training data records as the threshold of the minimum number of data points in a leaf note). There are 1 fully-grown tree and 6 pruned tree models created after the pruning process as shown in Figure 42 . We can see that the state definition rules of the fully-grown tree (Figure 42a) are not too hard to understand, but the CTHMM with the tree might not result in high accuracy of Viterbi path prediction because it has 14 states/leaves. Also, the tree may have overfitted the training data (CV-MR = 0.2004, which is slightly higher than the tree with 1 pruning in Figure 42b), but those pruned trees (Figure 42b-g) could solve this problem and create simpler trees.



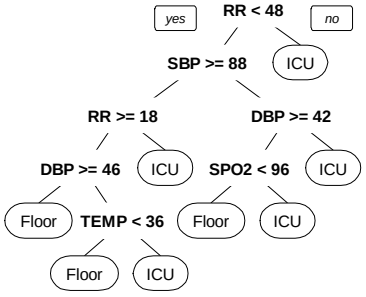
a) Fully-grown Tree (14 states, MI_O = 0.1981 bits, MI_S = 1.5105 bits, CV-MR= 0.2004)



b) Tree with 1 pruning (11 states)

MI_O = 0.2098 bits, MI_S = 1.4448 bits

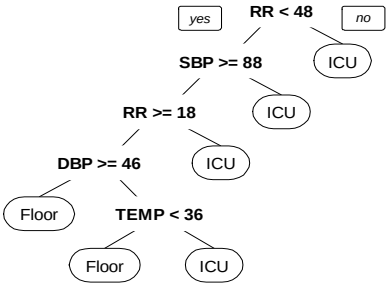
CV-MR= 0.1975



c) Tree with 2 pruning (8 states)

MI_O = 0.1960 bits, MI_S = 1.2203 bits

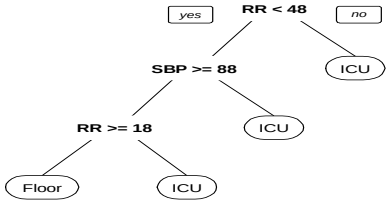
CV-MR= 0.2029



d) Tree with 3 pruning (6 states)

MI_O = 0.1834 bits, MI_S = 1.1001 bits

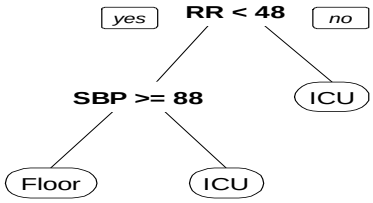
CV-MR= 0.2135



e) Tree with 4 pruning (4 states)

MI_O = 0.2017 bits, MI_S = 0.8960 bits

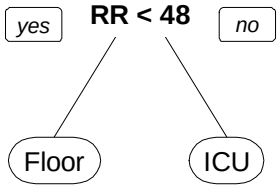
CV-MR= 0.2139



f) Tree with 5 pruning (3 states)

MI_O = 0.1475 bits, MI_S = 0.7085 bits

CV-MR= 0.2266



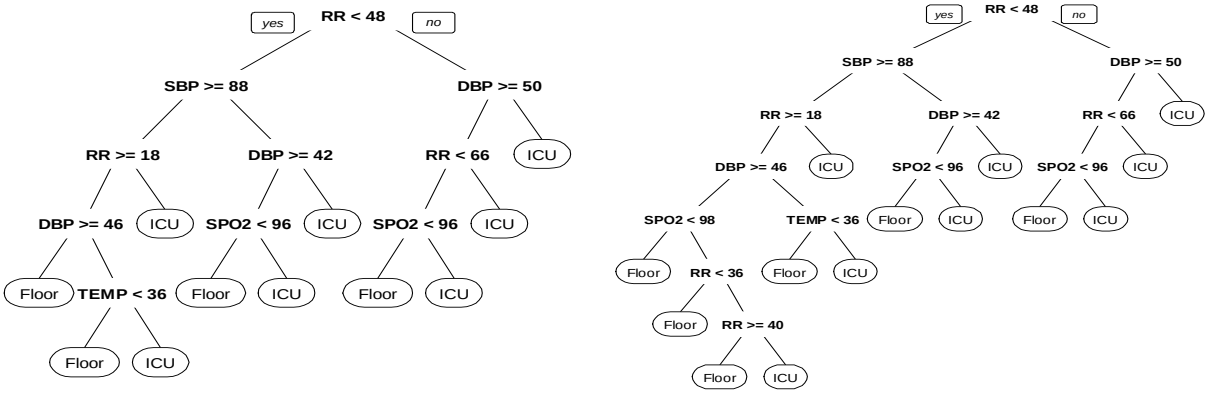
g) Tree with 6 pruning (2 states)

MI_O = 0.1638 bits, MI_S = 0.4395 bits

CV-MR= 0.2416

Figure 42: Tree models generated after post-pruning process

The next step is to create 2 additional candidate CTHMMs with two different $minbucket_{maxMI}$ that maximize the Mutual Informations computed from state transition and observation matrices of CTHMMs respectively. Both are simple one dimensional optimization problems introduced in Eq. 26. I found that, for the training dataset with 8,734 rows from CHP data, tree models with $minbucket$ = 101 and $minbucket$ = 89 data points, can maximize MI_O and MI_S (0.2098 and 1.5105 bits respectively). Given these 2 different $minbucket$, we can then create 2 $CTHMM_{\max\ MI}$ with tree models as shown in Figure 43 .



(a) Tree with maximum MI_O (11 states)
MI_O = 0.2098 bits, MI_S = 1.4448 bits, CV-MR= 0.1975

(b) Tree with maximum MI_S (14 states)
MI_O = 0.1981 bits, MI_S = 1.5105 bits, CV-MR= 0.2004

Figure 43: Maximum Mutual Information tree models

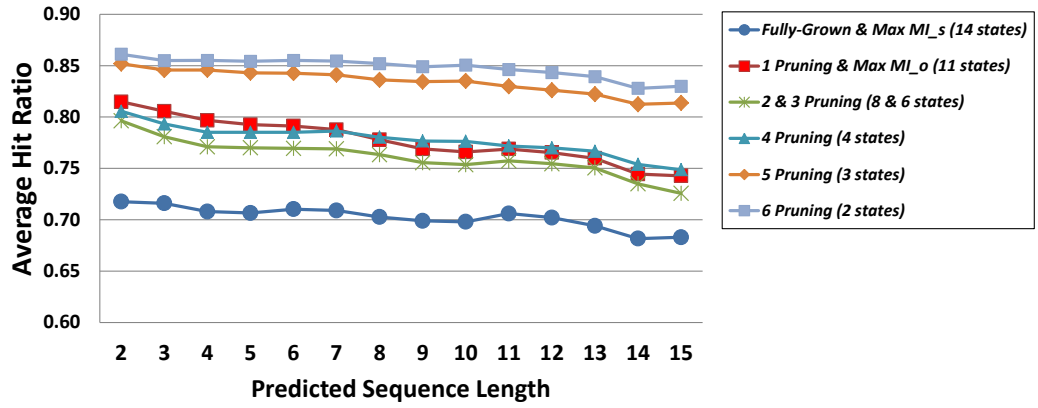
Note that the selected trees in Figure 43a and Figure 43b are identical to the trees in Figure 42b and Figure 42a respectively after performing MMIE (Eq. 26) with 2 different $minbucket$.

Now, we have 9 candidate tree models. I then consider two different types of state transition probability matrices for each tree model to create 9 CTHMMs and 9 CTHSMMs. To create CTHMMs, I assume that the duration of each record in both training and testing dataset are identical. In this case, however, these CTHMMs may be unrealistic and inaccurate because they

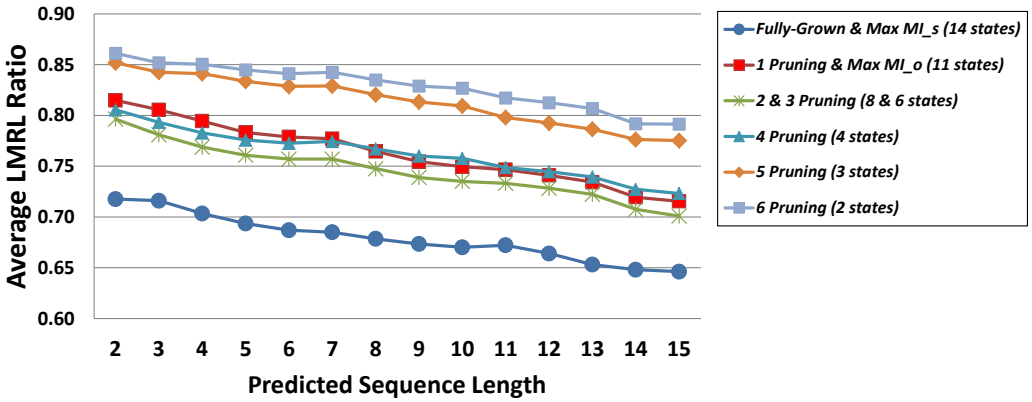
ignore the actual state duration distribution as discussed in Section 4.3. Therefore, in Section 4.4.1 and Section 4.4.2, I present two different model evaluation results and discuss the advantages of using CTHSMM instead of CTHMM.

4.4.1 Prediction of Most Probable Patient's Vital Sign State Transitions using CTHMM

The testing dataset of CHP dataset are used to evaluate these 9 candidate CTHMMs built from 9 aforementioned tree models. I apply state splitting/definition rules from each candidate CTHMM to mapping predictors (5 vital signs) into presumable state sequences (vital sign pattern dynamics). On the other hand, each patient's location sequences (observation sequence) of the testing dataset are used to infer most probable state sequences (predicted Viterbi path) with different lengths. Comparing the actual and predict state sequences, we can compute the aforementioned average hit and LMRL ratio. Figure 44 show both measures given each CTHMMs with different length of predicted sequences.



(a)



(b)

Figure 44: Average Hit and LMRL Ratio with different lengths of prediction

Figure 44 suggests that predicting longer sequence would result in lower accuracy in terms of average hit and LMRL ratios. Also, we expect that CTHMMs with fewer states would inevitably perform better. For example, the CTHMM with 6-pruning tree has highest hit and LMRL ratios, because there are only 2 states and the chance of hits are relatively higher. However, the CTHMM selected based on maximum Mutual Information of state-observation matrix (MI_O) perform comparatively well even with more states compared to CTHMMs with fully-grown or 2 to 4 pruning trees, which suggest that using $\text{CTHMM}_{\max MIo}$ provide us more information and relatively higher accuracy of Viterbi path prediction. On the other hand, although the CTHMM

with 6-pruning tree outperform others, it provide us much less information—we only know that patient's respiratory rate (RR) could switch between "RR < 48" and "RR >= 48".

Consider possible real-world applications of the CTHMMs we have built. Here, I choose the CTHMM in Figure 43a to infer most probable a patient's vital sign pattern dynamics given his/her location changes, as the CTHMM with maximum MI_O has shown that it could provide us relatively much information and better accuracy of prediction. Table 20 shows the observation/emission matrix with the state definition rules of the CTHMM. Note that those vital signs out of reference ranges in Table 19 are colored red.

Table 20: Observation matrix with state definition rules for CTHMM$_{max\ MIo}$

State	Floor	ICU	State Definition Rule
S1	0.0573	0.9427	RR>=48 & DBP< 50
S2	0.1103	0.8897	RR< 18 & SBP>=88
S3	0.1161	0.8839	RR< 48 & SBP< 88 & DBP< 42
S4	0.1608	0.8392	RR>=66 & DBP>=50
S5	0.8271	0.1729	RR< 48 & RR>=18 & SBP>=88 & DBP>=46
S6	0.6230	0.3770	RR< 48 & SBP< 88 & DBP>=42 & SPO2< 96
S7	0.3393	0.6607	RR< 48 & SBP< 88 & DBP>=42 & SPO2>=96
S8	0.5965	0.4035	RR>=48 & RR< 66 & DBP>=50 & SPO2< 96
S9	0.3918	0.6082	RR>=48 & RR< 66 & DBP>=50 & SPO2>=96
S10	0.6903	0.3097	RR< 48 & RR>=18 & SBP>=88 & DBP< 46 & TEMP< 36
S11	0.3898	0.6102	RR< 48 & RR>=18 & SBP>=88 & DBP< 46 & TEMP>=36

Assuming that we do not have the information about the patient's past vital sign state changes, the prior probabilities of states are thus set as equally-probable (i.e. 1/11 for each state for CTHMM$_{max\ MIo}$). Here, I consider 3 possible application scenarios:

Scenario 1: *Suppose that a patient stayed in ICU for a while. Then he was moved to Floor. Based on* $CTHMM_{max\ MIo}$ *, what would be the most probable the patient's vital sign state dynamics given different lengths of times (sequences) he stayed in ICU and Floor?*

Figure 45 shows 3 possible state-observation sequences. According to Figure 44b, we know that the prediction accuracy of most probable state sequence with length 3 and 6 are approximately 81% and 78% respectively.

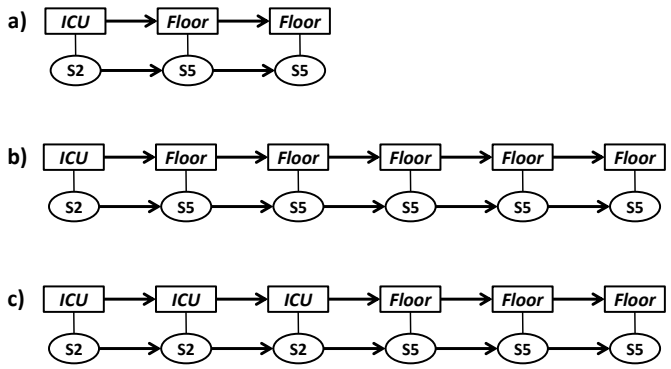


Figure 45: Most probable state sequences given different lengths and observations for Scenario 1

Figure 45 suggests that the patient's vital sign pattern would switch from State 2 to State 5, i.e. from "RR< 18 & SBP>=88" to "RR< 48 & RR>=18 & SBP>=88 & DBP>=46", if the patient was moved from ICU to Floor. Also, this pattern would most likely remains, no matter how long the patient stayed in ICU.

Scenario 2: *Another situation is that the patient first stayed in Floor. Then his condition became worse. Doctors decided to move him to ICU and keep monitoring him. Given that we have the process model ($CTHMM_{max\ MIo}$) built from the dataset with hundreds of patients' vital sign records, again, what would be the most probable the patient's vital sign patterns dynamics?*

Figure 46a suggests that the patient's vital sign pattern would remain in State 7, i.e. "RR< 48 & SBP< 88 & DBP>=42 & SPO2>=96", given the location sequence (Floor → ICU → ICU). However, the pattern may vary if we predict longer sequences. Figure 46b shows that, for those

patients who stayed in ICU longer, their vital sign patterns may most likely just remain in State 1, i.e." RR>=48 & DBP< 50". Also, the vital sign patterns may switch from State 8 to State 1 for those patients who stayed longer in Floor before being moved to ICU.

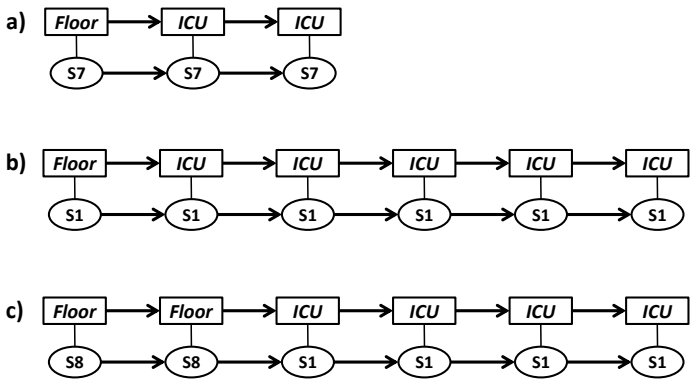


Figure 46: Most probable state sequences given different lengths and observations for Scenario 2

Scenario 3: *A patient was admitted and moved to ICU. He was then moved between Floor to ICU back and forth several times, as his situation was never stable. We would like to know, based on the CTHMM we choose, whether his vital sign conditions would become more stable.*

Figure 47 shows the predicted vital sign state sequences when lengths are from 3 to 6. Again, we expect that the accuracies of prediction are approximately 81%, 79% 78%, and 78% respectively. From Figure 47, we can see that the vital sign patterns vary in different lengths of prediction for those patients who were in such situation. However, the state definition rules provide us valuable information—these patients' vital signs would be in better ranges (State 5) when they were eventually moved to Floor.

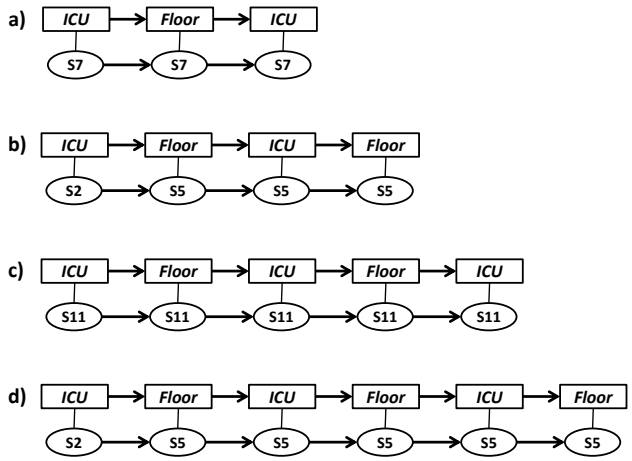
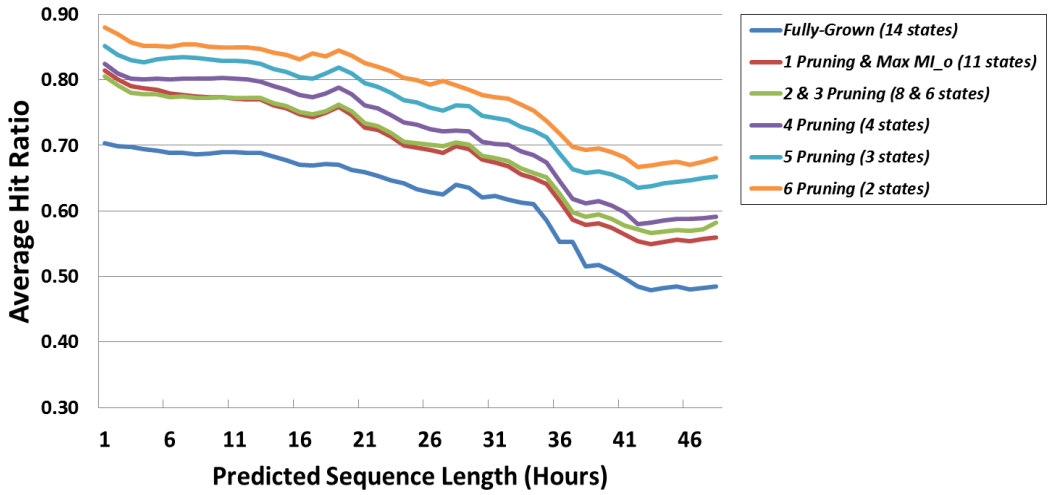


Figure 47: Most probable state sequences given different lengths and observations for Scenario 3

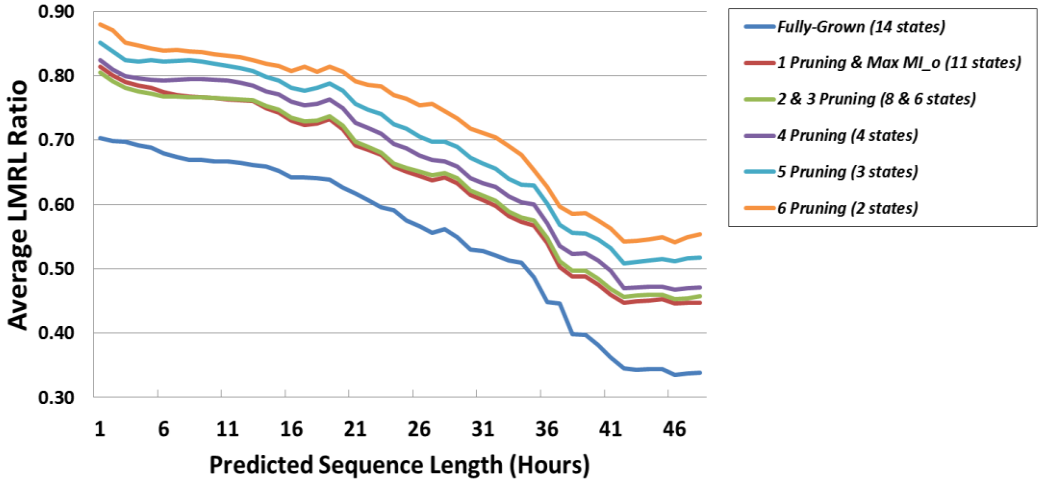
You may notice that there is a limitation of the CTHMM when it is applied to the applications like the aforementioned predictions of patients' vital sign dynamics—the most probable state sequences we have discovered do not refer to any state duration (sojourning time) of each state. As discussed previously, CTHMM is derived from HMM and therefore should be used in the applications when state duration are all supposed to be identical/fixed (or there is no information about state duration). In the next section, we consider the evaluation of more generalized version of proposed approach—CTHSMM introduced in Section 4.3, with the examples of previous 3 scenarios.

4.4.2 Prediction with Variable State Duration using CTHSMM

From aforementioned CHP dataset and tree models, we can build candidate CTHSMMs that explicitly model with state durations. As discussed in Section 4.3, the hit and LMRL ratio with different lengths of time units can be computed by comparing presumable state and predicted state sequences generated from testing dataset with states durations. Figure 48 shows the average hit and LMRL ratios of these CTHSMMs given different predicted length of times up to 48 hours.



(a)



(b)

Figure 48: Average Hit and LMRL Ratio with different hours of prediction

Again, we can see similar results close to those in Figure 44. Longer sequence of prediction would result in lower accuracy. Here, I would suggest using CTHSMM$_{max\ Mio}$ in real-world applications, as it balances the information richness and accuracy of prediction. Figure 49 shows the estimated duration density function for each state up to 72 hours using KDE introduced in Section 4.3

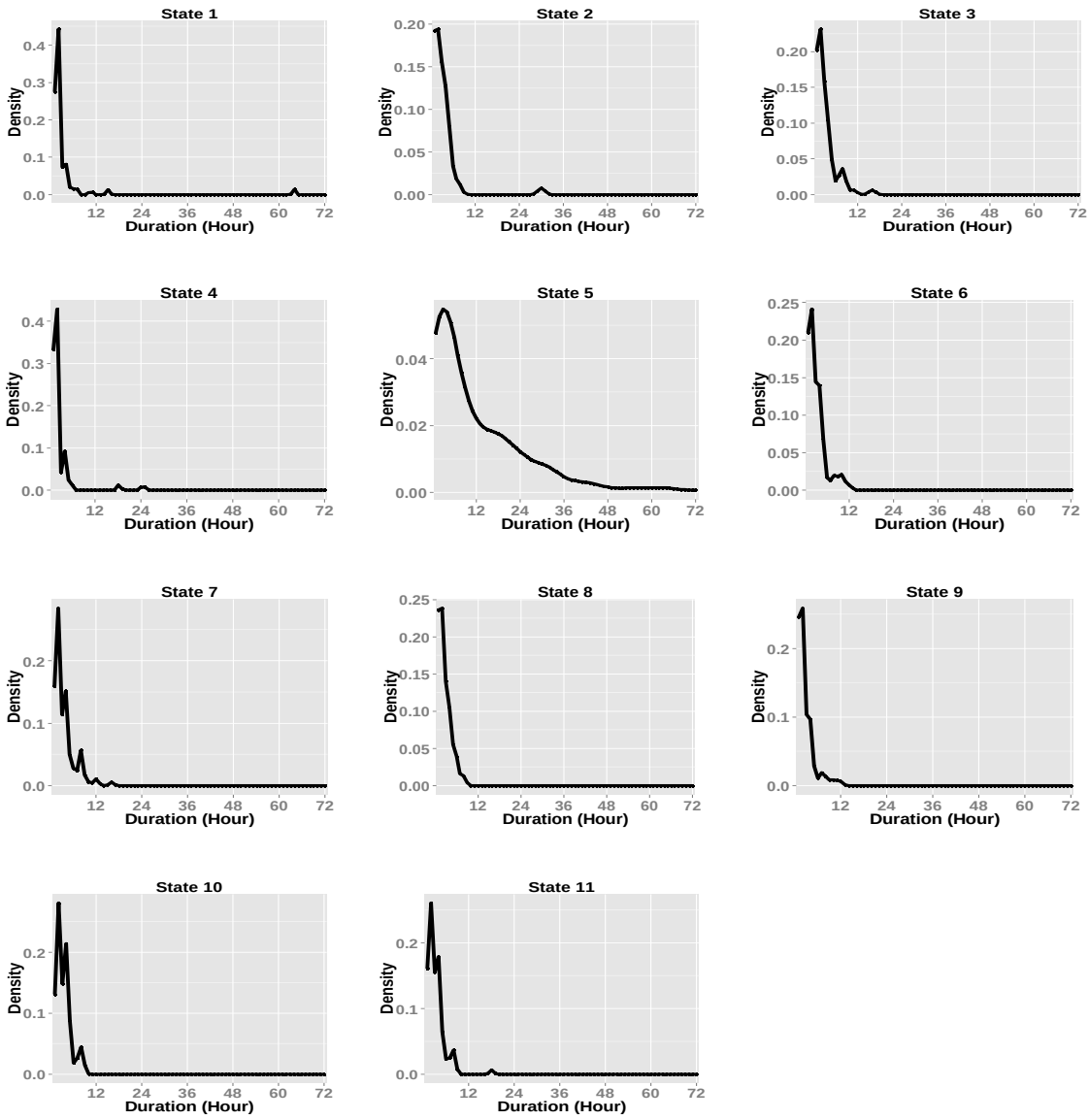


Figure 49: State Duration Distributions of CTHSMM$_{max\ Mio}$

We can see that most of the state duration (sojourning time) are within 12 hours. One interesting finding is that patients' vital sign conditions tend to stay in State 5 longer, most likely because patients' known vital signs are all in normal ranges as indicated in Table 20 and patients were required to keep staying in Floor longer.

To demonstrate the advantage of using CTHSMM, here I consider the same application scenarios discussed in Section 4.4.1 but with predicted times in hours. As shown in Figure 48b, if we predict state sequences longer than 21 hours, the accuracy of prediction are lower than 70% in terms of average LMRL ratio. So, I only discuss the cases that we predict state sequence within 21 hours in the following examples. In Figure 50, we consider the case in Scenario 1 that the patient was first in ICU and then was moved to Floor. We can learn that there is a "transition state" (e.g. S10 in Figure 50a) before the patient was moved to Floor, which is more realistic, as patients would most likely be moved from ICU to Floor only when their conditions get better.

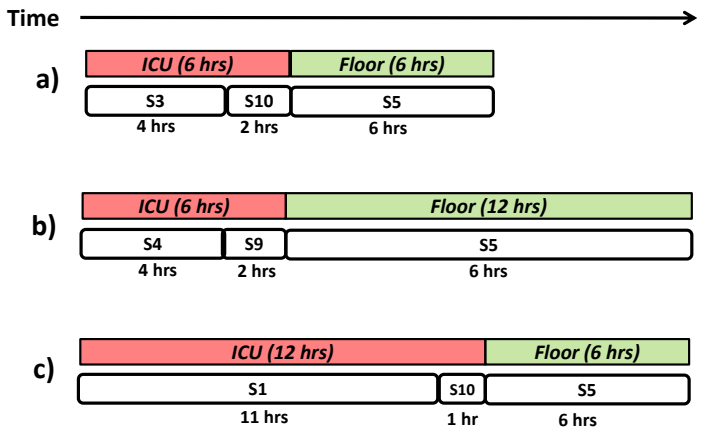


Figure 50: Most probable state sequences given different patients' location within 18 hours for Scenario 1

Consider the reverse cases in Scenario 2 that the patient was first admitted, and then was moved from Floor to ICU. Figure 51 shows a similar situation that there is a transition before a patient is moved to different location. It also prove that CTHSMM is more sensitive model compared to CTHMM and it could capture this kind of state transitions with the durations.

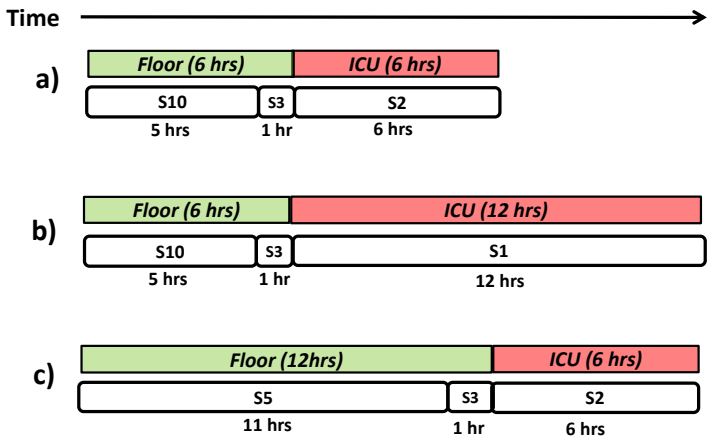


Figure 51: Most probable state sequences given different patients' location within 18 hours for Scenario 2

Let's consider more dramatic cases as discussed in Scenario 3 that the patient's condition was very unstable and was moved between Floor and ICU several times. In Figure 52, I present such cases with different lengths of predictions. We can see that CTHSMM could not only discover most relevant state sequences but also characterize more detail about state transitions with state durations.

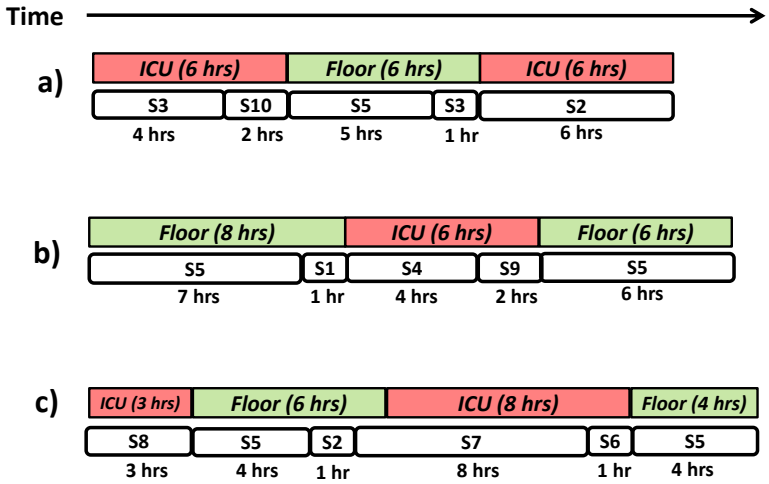


Figure 52: Most probable state sequences given different patients' location and times for Scenario 3

4.5 CONCLUSION

In Section 4, I introduced a process modeling technique based on tree-based Hidden (Semi-) Markov models to cope with state-observation sequence data stream from various kinds of information applications and systems. I proposed using the result of classification tree induction to transform data into state-observation sequences. This approach produces simple IF-THEN HMM/HSMM state splitting rules and suggests the best model based on maximizing mutual information of State-to-State and State-to-Observation matrices. The experiment results show that the proposed method could help discover most relevant state sequence patterns in temporal-probabilistic manner. The contributions of the proposed approach in this section can be summarized as:

1. *Provide a new process modeling technique that build a probabilistic process model and identify patterns simultaneously.*

2. *Help understand the evolution of a system with comprehensible IF-THEN system state definition rules.*

3. *Support decision making by predicting most probable pattern dynamics in temporal-probabilistic manner.*

I proved that the proposed CTHMM/CTHSMM is particularly useful when we are interested in understanding the changes of system patterns, which are the IF-THEN rule combinations of the predictors we use to identify states. Here, we consider CTHMM when there is no information about the duration of the data or the duration are assumed to be identical. On the other hand, we can build CTHSMM to take advantage of using the duration information from the data. The proposed CTHSMM is more advanced technique that could capture more detail about state transitions with times, and then infer most relevant state transition with the state durations.

5.0 SUMMARY

In Section 3 and Section 4, I present my proposed process monitoring and modeling approaches that address the research questions in Section 1.2. I assume that we cope with discrete dynamic systems and consider system state sequences as the process defined in Section 1. Two different applications, the detection of system anomalies and the discovery of most relevant system dynamics, are discussed in this dissertation. Both approaches are proved to be effective under specific research settings and assumptions. Here, I summarize the limitations and applicability of both approaches and review how I address the aforementioned research questions.

5.1 LIMITATIONS

The proposed approaches are based on probabilistic models and assumed to be used to cope with the log data generated from discrete dynamic systems, which means both approaches may not be applied to the anomaly detections and system development inferences for those systems that generate deterministic event logs. That is, further data pre-processing methods are required to convert these log data into sequences of system states so that these data can be used in proposed approaches.

Also, in Section 3, I presented my process anomaly detection approach that monitors the sequences of system states. These states can be identified by time-series representation techniques or by domain experts. However, how to choose the best techniques to discretize event log data into system states and how to determine the appropriate number of states remain open questions. On the other hand, the Classification Tree Hidden (Semi-) Markov Model, CTHMM/CTHSMM introduced in Section 4 is based on tree-based pattern classification methods, which means it also

has one major drawback of typical tree-based methods—variable selection bias (Hothorn, Hornik, & Zeileis, 2006). Typical classification tree learning algorithm tends to choose variables with more levels and to overfit the training dataset, which may easily result in creating an over-complex tree with many leaves. Although CTHMM/CTHSMM selects the best tree model by finding a tree with maximum Mutual Information (instead of using typical tree pruning methods), the large search space for optimization methods we use may require significant computation time when learning tree models from big training datasets.

5.2 DISCUSSIONS

In this dissertation, I discussed process anomaly detection and relevant process dynamics discovery. The detection of process anomaly is about finding short-term change points or long-term system deviation, provided that we only have an observable temporal state sequences. On the other hand, the discovery of relevant process dynamics is concerned with identifying hidden system states and then predicting/inferring most probable/relevant system hidden state dynamics given observable state sequences (observations). I presented 5 research question in Section 1.2 and demonstrated how I address them in Section 3 and Section 4 respectively. Here, I summarize how I answer these questions.

Question 1: Given a simple univariate temporal sequence as process log generated from an information system, how to detect the anomalies of the system from this temporal sequences? By monitoring the evolutionary information divergence/distance measures from the changes of stationary state probabilities (SV), we can detect whether the system deviates. In Section 3, I showed that comparing SVs generated by Google's PageRank algorithm can maximize the information divergence. I proved that General Jensen-Shannon Divergence D_{GJS} with SV

outperforms other measures, and the significant threshold of the D_{GJS} can be used as a criterion to detect the system short-term abrupt changes and long-term anomalies.

Question 2: Stationary state probability vector generated from first-order Markov chain is used in proposed process monitoring technique. Would using stationary sequence probability vector generated from higher-order Markov chain improve the accuracy of sequence anomaly detection?

In Section 3, I suggested using stationary state probability vector (SV) with those distribution-based divergence/distance measures to evaluate the deviation of the system we monitor. However, if we would like to detect a specific case that the temporal sequences show changes of higher-order state transition patterns, using the stationary sequence probability vector (SSeqV) generated from higher-order Markov chain instead of SV could improve the accuracy of detection. In Section 3.2.4, I proved it with examples that shows SSeqV outperforms SV only in this particular case.

Question 3: Most temporal sequence data generated from information systems are used to create deterministic process models. How to make use of these sequences to predict the dynamics of systems in probabilistic manners?

The proposed Classification Tree Hidden (Semi-) Markov Model (CTHMM/CTHSMM) introduced in Section 4 can automatically identify appropriate hidden states using tree-based supervised learning method, generate concurrent hidden state and observable state (observation) sequences that represent the dynamics of the system we monitor, and further build process models used to predict/discover relevant state changes in temporal-probabilistic manner.

Question 4: The hidden Markov model (HMM) is used here to create probabilistic process models. Most applications of HMM require well-defined state definitions or use pattern (state)

identification techniques to create models with vague state definitions. How to define human-comprehensible and appropriate number of states?

Most applications of HMM have pre-defined hidden states and estimate parameters using Expectation-Maximization-like learning algorithm. However, in my proposed CTHMM/CTHSMM discussed in Section 4, I used Classification Tree to estimate HMM parameters with appropriate number of hidden states. A new form of Maximum Mutual Information Estimation method is proposed to determine the best tree model. Besides, the hidden states discovered by CTHMM/CTHSMM are IF-THEN combinations of variables and their ranges. These state definitions are easy to understand by human without further interpretations.

Question 5: Typical HMM assume hidden state sojourning times are fixed, which implies that the hidden state durations are geometrically distributed and thus impractical. Can we extend proposed approach, Classification Tree Hidden Markov Model (CTHMM), by taking different hidden state durations into consideration? How does the extended model improve the prediction?

The proposed CTHMM can be extended by employing hidden semi-Markov model (HSMM) instead of typical HMM. The major difference between HMM and HSMM is that HSMM assumes the state duration/sojourning time distributions are variable. In addition, HSMM explicitly estimates model parameters (e.g. state transition probability matrix) and takes different state duration distributions into account so that the model can be used to infer different lengths/durations of system hidden state dynamics. In Section 4.3, an extended version of CTHMM, CTHSMM, with a modified state transition matrix is introduced. I showed a real-world application of CTHSMM and proved that proposed CTHSMM is very useful when we have the time/duration information about the system changes. In this case, CTHSMM can capture more detail about state transitions and infer most relevant state sequences with respective time units.

BIBLIOGRAPHY

Barbu, V. S., & Limnios, N. (2008). *Semi-Markov chains and hidden semi-Markov models toward applications: their use in reliability and DNA analysis* (Vol. 191).

Basseville, M., & Nikiforov, I. V. (1993). *Detection of abrupt changes: theory and application* (Vol. 15). Citeseer.

Ben-David, A. (2008). Comparison of classification accuracy using Cohen's Weighted Kappa. *Expert Systems with Applications: An International Journal*, 825–832.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees* (Vol. 1). Chapman & Hall/CRC.

Budalakoti, S., Budalakoti, S., Srivastava, A. N., Otey, M. E., & Otey, M. E. (2009). Anomaly Detection and Diagnosis Algorithms for Discrete Symbol Sequences with Applications to Airline Safety. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 39*(1), 101 –113. doi:10.1109/TSMCC.2008.2007248

Cassandras, C. G., & Lafortune, S. (2008). *Introduction to discrete event systems* (2nd ed.). Springer.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, *41*(3), 15:1–15:58. doi:10.1145/1541880.1541882

Chen, M.-Y., Kundu, A., & Srihari, S. N. (1995). Variable duration hidden Markov model and morphological segmentation for handwritten word recognition. *Image Processing, IEEE Transactions on*, *4*(12), 1675–1688.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20 No.1, pp.37–46.

Daw, C. S., Finney, C. E. A., & Tracy, E. R. (2003a). A review of symbolic analysis of experimental data. *Review of Scientific Instruments*, *74*(2), 915–930. doi:doi:10.1063/1.1531823

Daw, C. S., Finney, C. E. A., & Tracy, E. R. (2003b). A review of symbolic analysis of experimental data. *Review of Scientific Instruments*, *74*, 915. doi:10.1063/1.1531823

Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A., & Vapnik, V. (1996). Support Vector Regression Machines. *Advances in Neural Information Processing Systems 9, NIPS*, 155–161.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. New York: Wiley.

Dumas, M., Van der Aalst, W. M., & Ter Hofstede, A. H. (2005). *Process-aware information systems: bridging people and software through process technology*. Wiley-Interscience.

Durbin, R. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. *SIGMOD Rec.*, *23*(2), 419–429. doi:10.1145/191843.191925

Fodor, I. K. (2002). A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*.

Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268–278.

Grosse, I., Bernaola-Galván, P., Carpena, P., Román-Roldán, R., Oliver, J., & Stanley, H. E. (2002). Analysis of symbolic sequences using the Jensen-Shannon divergence. *Physical Review E*, *65*(4), 041905. doi:10.1103/PhysRevE.65.041905

Haas, P. J., & Haas, P. J. (2002). *Stochastic petri nets: Modelling, stability, simulation*. Springer.
Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques: Concepts and Techniques*. Elsevier.

Hanke, J. E., & Wichern, D. W. (2005). *Business Forecasting, Printice-Hall*. Inc.

Herzel, H., & Große, I. (1997). Correlations in DNA sequences: The role of protein coding segments. *Physical Review E*, *55*(1), 800–810. doi:10.1103/PhysRevE.55.800

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, *15*(3), 651–674.

Jiang, T., & Li, M. (1995). On the Approximation of Shortest Common Supersequences and Longest Common Subsequences. *SIAM Journal on Computing*, *24*(5), 1122–1139. doi:10.1137/S009753979223842X

Jolliffe, I. T. (2002). *Principal component analysis*. Springer verlag.

Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems*, *3*(3), 263–286. doi:10.1007/PL00011669

Keogh, E. J., & Pazzani, M. J. (2000). Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 285–289). Boston, Massachusetts, United States: ACM. doi:10.1145/347090.347153

Keogh, E., Lin, J., & Fu, A. (2005). HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence. In *Data Mining, IEEE International Conference on* (Vol. 0, pp. 226–233). Los Alamitos, CA, USA: IEEE Computer Society. doi:http://doi.ieeecomputersociety.org/10.1109/ICDM.2005.79

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models*. MIT press.

Lake, J. A. (1994). Reconstructing Evolutionary Trees from DNA and Protein Sequences: Paralinear Distances. *Proceedings of the National Academy of Sciences of the United States of America*, *91*(4), 1455–1459.

Lamberti, P. W., Majtey, A. P., Borras, A., Casas, M., & Plastino, A. (2008). Metric character of the quantum Jensen-Shannon divergence. *Physical Review A*, *77*(5), 052311. doi:10.1103/PhysRevA.77.052311

Landis, R., & Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *International Biometric Society*, 159.

Langville, A. N., & Meyer, C. D. (2006). *Google page rank and beyond*. Princeton Univ Pr.

Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE TRANSACTIONS ON INFORMATION THEORY*, *37*, 145–151.

Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery* (p. 11).

Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, *15*(2), 107–144.

Littman, M. L. (1996). *Algorithms for sequential decision making*. Brown University. Retrieved from http://www.cs.rutgers.edu/~mlittman/papers/thesis-mini.ps

Montgomery, D. C. (2008). *Introduction to Statistical Quality Control* (6th ed.). Wiley.

Núñez, H., Angulo, C., & Català, A. (2002). Rule extraction from support vector machines. In *European Symposium on Artificial Neural Networks Bruges* (pp. 24–26). Belgium.

Oxygen saturation in medicine. (2014, May 7). In *Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/w/index.php?title=Oxygen_saturation_in_medicine&oldid=605356703

Patil, G. P. (2006). Weighted distributions. *Encyclopedia of Environmetrics*. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/9780470057339.vaw009/full

Patil, G. P., & Rao, C. R. (1978). Weighted Distributions and Size-Biased Sampling with Applications to Wildlife Populations and Human Families. *Biometrics*, *34*(2), 179–189. doi:10.2307/2530008

Psaila, R. A. G., & Wimmers Mohamed &It, E. L. (n.d.). Querying shapes of histories.

Quinlan, J. R. (1993). C4.5: Programs for Machine Learning. *Machine Learning, Morgan Kaufmann Publishers, Inc.*, 235–240.

Rabiner, L., & Juang, B. (1986). An introduction to hidden Markov models. *ASSP Magazine, IEEE*, *3*(1), 4 –16. doi:10.1109/MASSP.1986.1165342

Redman, T. C. (1998). The impact of poor data quality on the typical enterprise. *Communications of the ACM*, *41*(2 (February 1998)), 79 – 82.

Robin, S., Rodolphe, F., & Schbath, S. (2005). *DNA, words and models*. Cambridge Univ Pr.

Sansom, J., & Thomson, P. (2001). Fitting hidden semi-Markov models to breakpoint rainfall data. *Journal of Applied Probability*, *38*, 142–157.

Setiono, R., & Liu, H. (1995). Understanding neural networks via rule extraction. In *14th International Joint Conference on Artificial Intelligence* (pp. 480–485). Montreal, Canada.

Shieh, J., & Keogh, E. (2008). iSAX: indexing and mining terabyte sized time series. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 623–631). Las Vegas, Nevada, USA: ACM. doi:10.1145/1401890.1401966

Silverman, B. W. (1986). *Density estimation for statistics and data analysis* (Vol. 26). CRC press.

Somervuo, P., & Kohonen, T. (1999). Self-Organizing Maps and Learning Vector Quantization for Feature Sequences. *Neural Processing Letters*, *10*(2), 151–159.

Srivastava, A., & Sahami, M. (2009). *Text mining: Classification, clustering, and applications* (Vol. 10). Chapman & Hall/CRC.

Van Der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag New York Inc.

Van Hee, K. M. (2004). *Workflow management: models, methods, and systems*. The MIT press.

Vital signs. (2014, May 13). In *Wikipedia, the free encyclopedia*. Retrieved from http://en.wikipedia.org/w/index.php?title=Vital_signs&oldid=608147769

Vital Signs in Children. (n.d.). *UW Health*. Retrieved May 18, 2014, from http://www.uwhealth.org/health/topic/special/vital-signs/abo2987.html

Weske, M. (2012). *Business process management: concepts, languages, architectures*. Springer.

Willsky, A., & Jones, H. (1976). A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems. *Automatic Control, IEEE Transactions on*, *21*(1), 108–112.

Yamanishi, K., & Takeuchi, J. (2002). A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 676–681).