Master Thesis in Statistics and Data Mining

Applying Machine Learning to Performance Trend Analysis

Araya Eamrurksiri

Division of Statistics
Department of Computer and Information Science
Linköping University

Supervisor
Prof. Krzysztof Bartoszek

Examiner
Prof. Anders Nordgaard

Nothing endures but change (Heraclitus)

Contents

Abstract

The core idea of this thesis is to reduce workload of manual inspection when the performance analysis of an updated software is required. CPU utilization, which is one of the essential factors for evaluating the performance, is analyzed. The purpose of this work is to apply machine learning techniques that are suitable for detecting the state of the CPU utilization, and also any changes in the test environment that effects the CPU utilization. The detection relies on a Markov switching model to identify structural changes, which are assumed to follow an unobserved Markov chain, in the time series data. A historical behavior of the data can be described by a first-order autoregression. Then, the Markov switching model becomes a Markov switching autoregressive model. Another approach based on a non-parametric analysis, a distribution-free method that requires fewer assumptions, called an E-divisive method is purposed. This method used hierarchical clustering algorithm to detect multiple change point locations in the time series data. As the data used in this analysis does not contain any ground truths, the evaluation of the methods is analyzed by generating simulated datasets with known states. Besides, these simulated datasets are used for studying and comparing between the Markov switching model and the E-divisive method. Results show that the former method is preferable because of its better performance and more efficient in detecting changes. Some information about the state of the CPU utilization are also obtained from performing the Markov switching model. The E-divisive method is proved to have less power in detecting changes and has a higher rate of missed detections. The results from applying the Markov switching autoregressive model to the real data are presented with interpretations and discussions.

Keywords: Markov switching model, Non-parametric analysis, CPU utilization

Acknowledgments

1. Introduction

1.1. Background

In this study, *change point analysis* will be used to identify changes over time in performance of Ericsson's software products. Many test cases are executed for testing software packages in a simulation environment. Before launching the software products to its customers, the company needs to test and determine how each software package performs. The performance of these software packages is evaluated by considering on CPU utilization (percentages of CPU's cycle spent on each process), memory usage, and latency.

Structural changes are often seen in time series data. This observable behavior is highly appealing to statistical modelers who want to develop a model which is well explained. A method to detect changes in time series data when a time instant is unknown is called *change point analysis* (Basseville et al., 1993). The analysis discovers the time point where the changes occur. Change point analysis can be referred to different kinds of name such as breakpoint and turning point. However, *change-point* is the commonly used term for the point in a time series where a change takes place. Another important term used in this area is *regime switch* which refers to persistent changes in time series structure after the occurrence of change point (Weskamp and Hochstotter, 2010). Change point analysis has been studied for several decades as it is a problem of interest in many applications which the characteristic of data is collected over time. A change should be flagged as soon as it occurs in order to be properly dealt with reducing any possible consequences (Sharkey and Killick, 2014). Here are some examples.

- Medical condition monitoring: Evaluate the sleep quality of patients based on their heart rate condition (Staudacher et al., 2005).
- Climate analysis: The temperature or climate variations are detected. This method has gradually become important over the past few decades due to the effects of the global warming and the increases in greenhouse gas emissions (Reeves et al., 2007; Beaulieu et al., 2012).
- Quality control: Since industrial production is a continuous production process, in mass production process, if the product controlled value is not monitored and exceeds the tolerable level undetected, it could lead to the loss of a whole production lot (Page, 1954).

- Other applications: Identifying fraud transaction (Bolton and Hand, 2002), detecting anomalies in the market price (Gu et al., 2013) and detecting signal processing (Basseville et al., 1993) in streaming data as well.

In recent years, a method called hidden Markov model or Markov switching model has become widely used for discovering change points in time series. Both terms are accepted, usage varies with different fields of study. Markov switching model uses a concept of a Markov chain to model an underlying segmentation as different states and then specify a distinct change of location. Hence, the method is able to identify a switch in time series when change point occurs (Luong et al., 2012). This method is generally used in almost all current systems in speech recognition (Rabiner, 1989) and found to be important in climatology such as describing the state in the wind speed time series (Ailliot and Monbet, 2012) and in biology (Stanke and Waack, 2003) where protein coding genes are predicted. Markov switching model has been extensively applied in the field of economics and finance and has a large literature. For example, business cycles can be seen as hidden states with seasonal changes. The growth rate of gross domestic product (GDP) is modeled as a switching process to uncover business cycle phases i.e., expansion and recession. The fitting model can also be used to understand the process where there is a transition between the economic state and the duration of each period (Hamilton, 1989). In finance data, time series of returns is modeled in order to investigate stock market situation i.e., bull or bear market (Kim et al., 1998).

Markov switching model is one of the most well-known non linear time series models. This model can be applied to various time series data with dynamic behavior. The structural changes or regime shifts in data implies that constant parameter settings in a time series model might be insufficient to capture these behaviors and describe their evolution. Markov switching model takes the presence of shifting regime in time series into account and models multiple structures that can explain these characteristics in different states at different time. The shift between states or regimes comes from the switching mechanism which is assumed to follow an unobserved Markov chain. Thus, the model is able to capture more complex dynamic patterns and also identify the change of locations and regime switch in time series.

For the current Ericsson setting, each software package running through the test system is viewed as a time point in time series and the performance of each software package is treated as an observed value. It is proven that the observed values are not completely independent of each other i.e., the performance of the current software package depends on the performance from the prior version of the software package. Therefore, additional dependencies are taken into consideration by a first-order autoregression when modeling. The Markov switching model becomes the Markov switching autoregressive model. This model is applied to the given data in order to discover the changes in the performance.

There are two approaches, a parametric and a non-parametric analysis, for detecting the change point in the time series. The parametric analysis benefits from assuming

some knowledge of data distribution and integrating it to the detection scheme. On the other hand, the non-parametric analysis is more flexible in that there are no assumption made about the distribution. It can, therefore, apply to a wider range of applications and capture various kinds of changes (Sharkey and Killick, 2014). The non-parametric analysis using hierarchical estimation techniques based on a divisive algorithm is implemented. This method, which is called an E-divisive, is designed to perform multiple change point analysis while trying to make a few assumptions as possible. The E-divisive method estimates change points by using a binary bisection approach and a permutation test. The method also capable of estimating not only univariate data but also multivariate data.

In this study, the parametric analysis using the Markov switching autoregressive model and the non-parametric analysis using the E-divisive method are used for identifying change point locations in the time series data.

1.2. Objective

The core idea of this thesis is to reduce workload of manual inspection when the performance analysis of an updated software package is required. With an increased amount of generated data from numerous test cases, the inspection becomes tedious and inefficient to be done manually. The main objective of this thesis is to implement machine learning, an algorithm that has an ability to learn from data, in order to analyze the performance of the software package. The algorithm will help indicate whether the performance of the software package is in a degrading, improving or steady state. It is also worth mentioning that the performance of a particular software package can vary on different test environments. The implemented algorithm should also be able to detect when the test environment is altered. This thesis only focuses on the CPU utilization, which is one of the three essential factors for evaluating the performance of the upgraded software package.

To summarize, this thesis aims to:

- Detect the state of the CPU utilization (degrading, improving, or steady state)
- Detect whether there is any change in the test environment that effects the CPU utilization

The thesis is structured as follow: Chapter 2 provides details and descriptions of datasets used in the analysis. Chapter 3 presents methodology. Results from the analysis along with tables and plots are shown in Chapter 4. Chapter 5 discusses the outcomes and the obtained results. Lastly, Chapter 6 contains conclusion and future work.

2. Data

2.1. Data sources

The data used in this thesis is provided by Ericsson site in Linköping, Sweden. Ericsson[1], founded by Lars Magnus Ericsson in 1876, is one of the world's leaders in the telecommunication industry. The company provides services, software products, and infrastructure related to information and communications technology (ICT). Its head quarter is located in Stockholm, Sweden. Ericsson continuously expands its services and products beyond telecoms sector such as mobile broadband, cloud services, transportation, and network design.

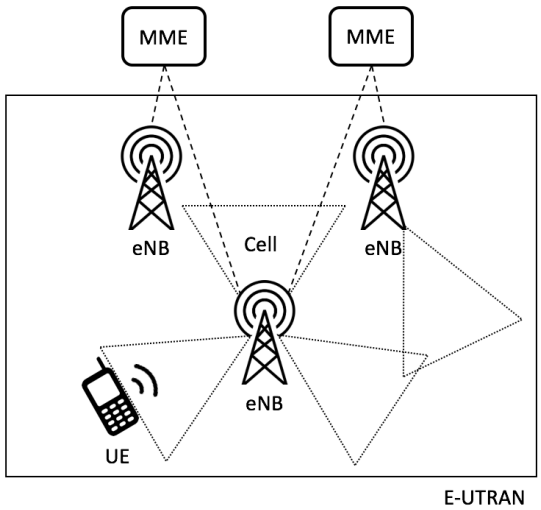


Figure 2.1.: LTE architecture overview

LTE, widely known as 4G, is a radio access technology for wireless cellular communications. The high-level network architecture of LTE is shown in Figure 2.1 and is described as follows (Dahlman et al., 2013). The E-UTRAN, an official standard name for the radio access network of LTE, is the entire radio access network. It handles the radio communication between the User Equipment (UE) or mobile device and the base stations called eNB. Each base station controls and manages radio communications with multiple devices in one or more cells. Several base stations

[1]https://www.ericsson.com/

are connected to a Mobility Management Entity (MME), which is a control-node for the LTE network. MME establishes a connection and runs a security application to ensure that the UE is allowed on the network. In LTE mobile network, multiple UEs are connected to a single base station. A new UE performs a cell search procedure by searching for an available eNB when it first connects to the network. Then, the UE sends information about itself to establish a link between the UE and the eNB.

Network procedures that will be briefly described here are *Paging* and *Handover*. Paging is used for the network setup when UE is in an idle mode. If MME wants to notify UE about incoming connection requests, MME will send paging messages to each eNB with cells belonging to the Tracking Area (TA) where the UE is registered. UE will wake up if it gets the Paging message and will react by triggering a RRC connection request message. The paging process happens in number of paging per second. Handover is a process of changing the serving cells or transferring an ongoing call from one cell to another. For instance, if the UE begins to go outside the range of the cell and enters the area covered by another cell, the call will be transferred to the new cell in order to avoid the call termination. The rate of handover is in second.

Ericsson makes a global *software release* in roughly 6-month cycles or two major releases per year. Each of these releases contains a bundle of features and functionality that is intended for all the customers. The software release is labeled with L followed by a number related to the year of release and a letter either A or B, which generally corresponds to the 1^{st} and 2^{nd} half of that year. Ericsson opens up a track for each software release and begins a code integration track. This track becomes the main track of the work or the focal branch for all code deliveries. There are hundreds of teams producing code, and each team commit the code to this track continuously. In order to create a structure for this contribution, a daily *software package* is built which can be seen as a snapshot or a marker in the continuous delivery timeline. This software package is then run through various automated test loops to ensure that there are no faults in the system. The software packages are named R, and followed by one or more numbers, which is then followed by one or more letters. R stands for Release-state. To summarize, each software package is a snapshot in the code integration timeline. Figure 2.2 presents a relationship between a software release and software packages.

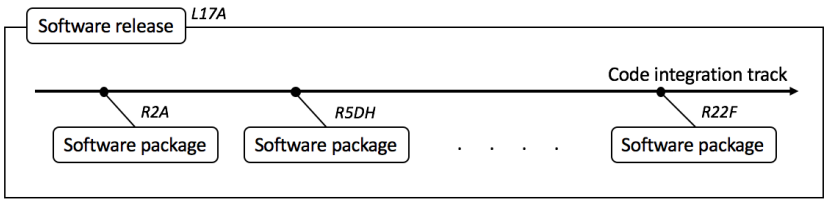


Figure 2.2.: An example of one software release that begins a code integration track. Several software packages are launched in the timeline.

There are thousands of automated tests performed. Each test belongs to a particular suite of tests, which belong to a particular Quality Assurance (QA) area. For this thesis, only a subset of test cases belonging to QA Capacity area, that focus on signaling capacity, is used. The QA Capacity is responsible for testing and tracking test cases related to eNB capacity. Each one of these test cases has a well-defined traffic model that it tries to execute. The traffic model, in this context, means certain intensity (per second) of procedures which can be seen as stimuli in the eNB. Basically simulating the signaling load from a large number of UEs served simultaneously by the eNB. The eNB then increments one or more counters for each one of these procedures or stimuli that it detects. These counters are called local events and represented by *EventsPerSec*.

A logging loop is started during the execution of these test cases of QA Capacity – signaling capacity. The logging loop collects several metrics, and a subset of these metrics is what this thesis is currently studying. Once the logging loop is finished, it is written to a log file. Then, there are cron jobs that slowly scan through this infrastructure once a day to find latest logs and do a post-processing. The final output is either CSV data or JSON encoded charts. The flowchart of this process is illustrated in Figure 2.3.

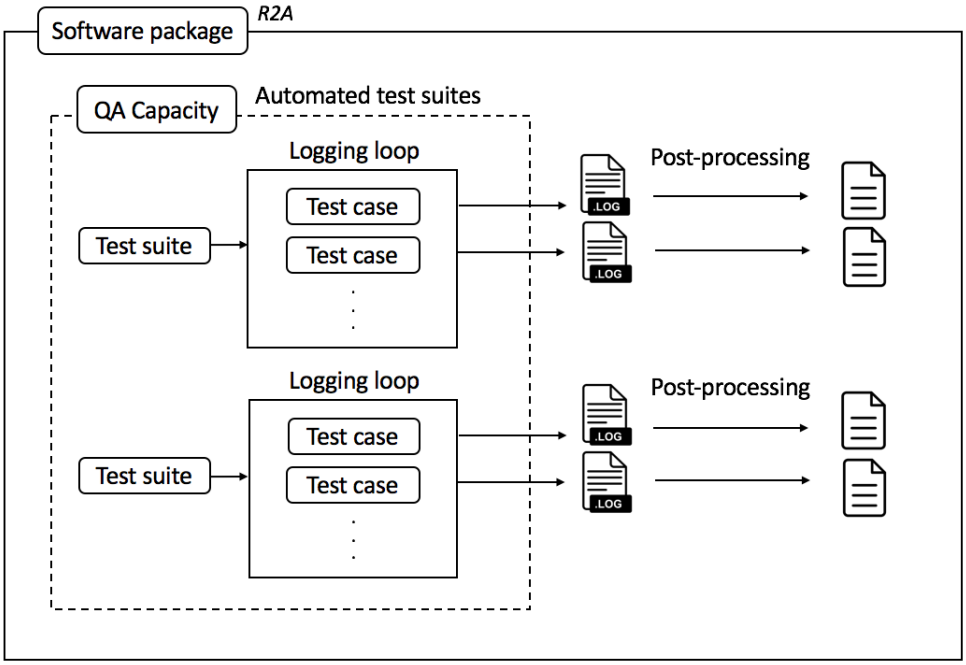


Figure 2.3.: An example of one software package. First, QA Capacity automated test suites is started. For each test suite, a logging loop is started and a log is produced for each test case. The log file is fed to post-processing tools, and the data output is obtained.

2.2. Data description

The data used in the thesis contains 2,781 test cases. It is collected on 20 January 2017 and is extracted from log files produced by test cases. There are different types of test cases which are being executed in the automated test suites. Each test case is viewed as an observation in the data. The following are the variables in the data:

Metadata of test case

- Timestamp: Date and time when a test case is being executed (yy-dd-mm hh:mm:ss)
- NodeName: IP address or the name of a base station
- DuProdName: Product hardware name
- Fdd/Tdd: Different standard of LTE 4G Technology. FDD and TDD stand for Frequency Division Duplex and Time Division Duplex, respectively.
- NumCells: Number of cells in the base station
- Release: Software release
- SW: Software package
- LogFilePath: Path for log file produced by a test case

Observable memory

- MemFreeKiB
- SwapFreeKiB
- BufferCacheKiB
- PageCacheKiB
- RealFreeKiB

CPU

- TotCpu%
- PerCpu%
- PerThread%
- EventsPerSec

  The EventsPerSec variable, or Event intensity, contains several local events that can be used when defining the test cases. Apparently, there is no fixed number of local events in this variable as different test cases involve different testing procedures. The local events along with their values are also varied

depending on which types of test cases are being executed. An example of the local events in test cases is shown in Table 2.1.

Table 2.1.: List of local events in the test cases separated by a tab character

Test case	EventsPerSec
1	ErabDrbRelease=166.11 ErabSetupInfo=166.19 PerBbUeEventTa=167.98 PerBbUetrCellEvent=12.00 ProcInitialCtxtSetup=166.20 RrcConnSetupAttempt=166.21 RrcConnectionRelease=166.11 S1InitialUeMessage=166.20 UplinkNasTransport=32.06 ...
2	ErabDrbAllocated=641.30 EventS1InitialUeMessage=142.20 McRrcConnectionRequest=142.99 McX2HandoverRequest=98.70 Paging=1399.94 PerBbLcgEvent=26.14 ...
...	...

2.3. Data preprocessing

The relevant aspects of the data preprocessing step is describe here. The dataset, which spans three software releases, is split into three datasets according to the software release. In this thesis framework, Ericsson software release will be referred as software release A, B, and C.

The test cases in each dataset are sorted by its software package version, which is named alphabetically. The name of the software package is used as a time point in the time series.

Some test cases are filtered out in the preprocessing step because the test cases are not always executed properly. The problem is either no traffic is generated during the test case or the data is not logged. This usually result in a missing value in the *EventPerSec* field, which causes the test case to be incomplete. The particular test case and all the data related to the test case is ignored. If the value in any other fields is missing, the test case will also be ignored.

In Table 2.1, it can be seen that the *EventsPerSec* stores multiple values separated by a tab character. These tab-separated values in the field are split into columns. A function is implemented to perform this process and is further described in details in sec. A.1. The process is done in order to turn its local events and values, which characterize the test case, into usable parameters. These parameters are later on used as predictor variables when the Markov switching model is applied.

Each software release consists of several software packages. For each specific software package, numerous test cases are executed. Since a software package acts as a time point in the time series, the result is rather difficult to visualize using every executed test case for each software package. Hence, the test case that has the lowest value of the CPU utilization (or minimum value of *TotCpu%*) is selected to represent a performance of the specific software package. Although taking an average of multiple runs for test cases in the software package appears to be a good approach, it does not yield the best outcome in this case. The first reason is that manipulating data can easily be misleading. Another important reason for not using the average value of the CPU utilization is that the essential information in the test case could be lost. Each test case has its own local events in *EventsPerSec* field that is used for identifying the test case. The details of these local events will be absent if the CPU utilization of the test case is averaged. It is, therefore, settled to keep the original data and always use the unmanipulated data to visualize the time series.

After performing all the steps described above, the dataset of the software release A, B, and C consist of 64, 241, and 144 test cases, respectively. Lastly, each dataset with particular software release is divided into two subsets. Ninety percents of the dataset is used for training the model and the remaining ten percents is left out for testing the model

Some variables in the data are chosen to be used for further analysis. In total, there are one response variable and six predictor variables. Table 2.2 shows the name of variables and their descriptions. The first three predictor variables are local events of the test case, which can be found in the *EventsPerSec*, while the last three variables are considered as the test environment. These variables appear to have a high influence to the CPU utilization.

Table 2.2.: List of the selected variables followed by its type and unit measure

Variable	Name	Type	Unit
Response	TotCpu%	Continuous	Percentage
Predictor	RrcConnectionSetupComplete	Continuous	Per second
	Paging	Continuous	Per second
	X2HandoverRequest	Continuous	Per second
	DuProdName	Categorical	
	Fdd/Tdd	Binary	
	NumCells	Categorical	

3. Methods

A chapter first starts by providing a survey of existing methods that address the problem of detecting changes in a system. Later on, general information about Markov chains, the simple Markov switching model feature, and model specification namely Markov switching autoregressive model are discussed. Thereafter, three sections are devoted to methods for estimating the values of parameters, predicting a state for a new observation, and selecting a suitable model for the datasets. Another change-point method in a non-parametric approach is described. Finally, the simulation technique is explained.

3.1. Survey of existing methods

Change point detection, Anomaly detection, Intrusion detection, or *Outlier detection* are terms that closely related to one another. The main idea of these terms is to identify and discover events that are abnormal from the usual behavior. There are several methods to address this types of problem. A survey of existing methods has been done in the thesis, and some methods are presented in this section.

Valdes and Skinner (2000) employed a Bayesian inference technique, specifically a naive Bayesian network, to create an intrusion detection system on traffic bursts. Even though the Bayesian network is effective in detecting anomalies in some applications, there are some limitations that should be considered when using this method. As accuracy of a detection system depends heavily on certain assumptions, the system will have low accuracy if an inaccurate model is implemented (Patcha and Park, 2007).

Support vector machine (SVM) introduced in Cortes and Vapnik (1995) is a supervised learning algorithm to deal with a classification analysis problem by using the idea of separating hyperplanes. The main reason that SVM is used in anomaly detection is because of its speed and scalability (Sung and Mukkamala, 2003). Although this method is effective in identifying new kinds of anomalies, the method often has a higher rate of false alarms due to the fact that the SVM method ignores the relationships and dependencies between the features (Sarasamma et al., 2005).

Self-organizing maps (SOM) developed by Kohonen (1982) is a well-known unsupervised neural network approach for cluster analysis. SOM is efficient in handling large and high dimensional datasets. Nousiainen et al. (2009) used SOM for an

anomaly detection in a server log data. The study presented an ability of the SOM method in detecting anomalies in the data, and also compared the results from the SOM method with a threshold based system. A disadvantage of the SOM is that initial weight vector affects a performance of the SOM, which leads to an unstable clustering result. Besides, if the anomalies in the data tend to form into clusters, this method will not be able to detect these anomalies (Chandola et al., 2009).

Based on previous work, the Hidden Markov model or the Markov switching model has also been used in identifying changes and anomalies. One drawback from the method based on the Markov chain is that the method has a high computational cost, which is not scalable for an online change application (Patcha and Park, 2007). Apart from changes that can be detected in the data, some knowledge about the unobservable condition of the system can also be found such as an uncertainty of change points. This additional information makes the method more appealing than the other methods. Therefore, the Markov switching model is implemented in this thesis framework.

3.2. Markov chains

A Markov chain is a random process which has a property that is given the current value, the future is independent of the past. A random process X contains random variables $X_t : t \in T$ indexed by a set T. When $T = \{0, 1, 2, ...\}$ the process is called a discrete-time process, and when $T = [0, \infty)$ it is called a continuous-time process. Let X_t be a sequence of values from a state space S. The process begins from one of these states and moves to another state. The move between states is called a step. The process of Markov chains is described here.

Definition 3.2.1. (Grimmett and Stirzaker, 2001, p.214) If a process X satisfies the Markov property, the process X is a first order Markov chain

$$P(X_t = s|X_0 = x_0, X_1 = x_1, ..., X_{t-1} = x_{t-1}) = P(X_t = s|X_{t-1} = x_{t-1})$$

where $t \geq 1$ and $s, x_0, ..., x_{t-1} \in S$

If $X_t = i$ then the chain is in state i or the chain is in the ith state at the tth step.

There are transitions between states which describe the distribution of the next state given the current state. The evolution of changing from $X_t = i$ to $X_t = j$ is defined by the transition probability as $P(X_t = j|X_{t-1} = i)$. For Markov chains, it is frequently assumed that these probabilities depend only on i and j and do not depend on t.

Definition 3.2.2. (Grimmett and Stirzaker, 2001, p.214) The chain is time-homogeneous if

$$P(X_{t+1} = j|X_t = i) = P(X_1 = j|X_0 = i)$$

for all t, i, j. The probability of the transition is independent of t. A *transition matrix* $\mathbf{P} = (p_{ij})$ is a matrix of transition probabilities

$$p_{ij} = P(X_t = j|X_{t-1} = i)$$

Theorem. *(Grimmett and Stirzaker, 2001, p.215) The transition matrix* $\mathbf{P}$ *is a matrix that*

- *Each of the entries is a non-negative real number or* $p_{ij} \geq 0$ *for all* i, j
- *The sum of each row equal to one or* $\sum_j p_{ij} = 1$ *for all* i

Definition 3.2.3. (Grimmett and Stirzaker, 2001, p.220) State i is called persistent (or recurrent) if

$$P(X_t = i \text{ for some n} \geq 1|X_0 = i) = 1$$

Let $f_{ij}(n) = P(X_1 \neq j, X_2 \neq j, ..., X_t \neq j|X_0 = i)$ be the probability of visiting state j first by starting from i, takes place at tth step.

Definition 3.2.4. (Grimmett and Stirzaker, 2001, p.222) The mean recurrence time of a state i is defined as

$$\mu_i = E(T_i|X_0 = i) = \sum_n n \cdot f_{ii}(n)$$

State i is a non-null persistent (or positive recurrent) if μ_i is finite. Otherwise, the state i is null persistent.

Definition 3.2.5. (Grimmett and Stirzaker, 2001, p.222) A state i that has a period $d(i)$ is defined as

$$d(i) = gcd\{n : \ p_{ii}(n) > 0\}$$

where *gcd* is the greatest common divisor. If $d(i) = 1$, then the state is said to be aperiodic. Otherwise, the state is said to be periodic.

Definition 3.2.6. (Grimmett and Stirzaker, 2001, p.222) A state is called ergodic if it is non-null persistent and aperiodic.

Definition 3.2.7. A chain is called irreducible if it is possible to go from every state to every other states.

Theorem. *(Manning et al., 2008) If there is a aperiodic finite state space, an irreducible Markov chain is the same thing as ergodic Markov chain.*

3.3. Markov switching model

A Markov switching model is a switching model where the shifting back and forth between the states or regimes is controlled by a latent Markov chain. The model structure consists of two stochastic processes embedded in two levels of hierarchy. One process is an underlying stochastic process that is not normally observable, but possible to be observed through another stochastic process which generates the sequence of observation (Rabiner and Juang, 1986). The time that transition between state occurs is random. In addition, the state transition are assumed to follow the Markov property that the future state depends only on the current state.

The Markov switching model is able to model more complex stochastic processes and describe changes in the dynamic behavior. A general structure of the model can be drawn graphically as shown in Figure 3.1, where S_t and y_t denote the state sequence and observation sequence in the Markov process, respectively. The arrows from one state to another state in the diagram implies a conditional dependency.

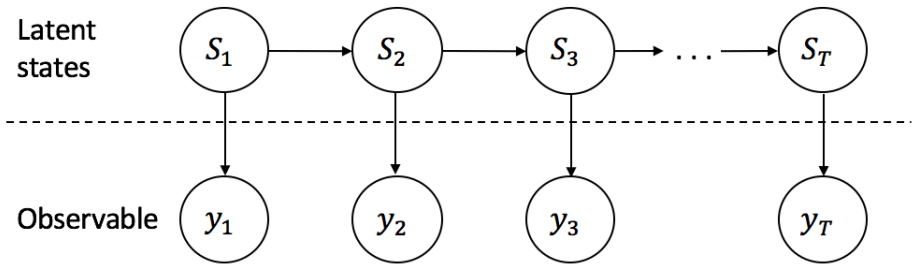


Figure 3.1.: Model structure

The process is given by (Hamilton, 1989)

$$y_t = X_t \beta_{S_t} + \varepsilon_t \tag{3.1}$$

where,

y_t	is an observed value of the time series at time t
X_t	is a design matrix, also known as model matrix, containing values of predictor variables of the time series at time t
β_{S_t}	are a column vector of coefficients in state S_t, where $S_t \in \{1, ..., k\}$
ε_t	follows a Normal distribution with zero mean and variance given by $\sigma^2_{S_t}$

Equation 3.1 is the simplest form for the switching model. To aid understanding, the baseline model is assumed to have only two states ($k = 2$) in this discussion. S_t is a random variable which is assumed that the value $S_t = 1$ for $t = 1, 2, ..., t_0$ and $S_t = 2$ for $t = t_0 + 1, t_0 + 2, ..., T$ where t_0 is a known change point.

The transition matrix $\mathbf{P}$ is an 2x2 matrix where row j column i element is the transition probability p_{ij}. A diagram showing a state-transition is shown in Figure 3.2. Note that these probabilities are independent of t.

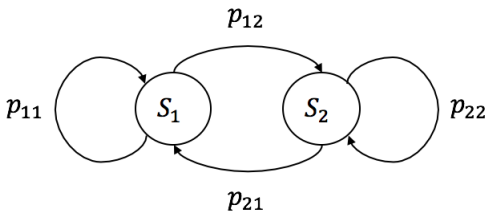


Figure 3.2.: State-transition diagram

Since the whole process S_t is unobserved, the initial state where $t = 0$ also needs to be specified. The probability which describes the starting distribution over states is denoted by

$$\pi_i = P(S_0 = i)$$

There are several options for computing the probability of the initial state. One procedure is to commonly set $P(S_0 = i) = 0.5$. Alternatively, the unconditional probability of S_t

$$\pi_1 = P(S_0 = 1) = \frac{1 - p_{jj}}{2 - p_{ii} - p_{jj}}$$

can be used by presuming an ergodic Markov chain (Hamilton, 2005).

A coefficient of a predictor variable in the Markov switching model can have either different values in different state or a constant value in all state. The variable which have the former behavior is said to have a *switching effect*. Likewise, the variable which have the same coefficient in all states is the variable that does not have a switching effect, or said to have a *non-switching effect*.

A generalized form of Equation 3.1 can be defined as (Perlin, 2015)

$$y_t = X_t^{ns}\alpha_t + X_t^s\beta_{S_t} + \varepsilon_t \tag{3.2}$$

where,

X_t^{ns}	contains all predictor variables that have non-switching effect of the time series at time t
α_t	are non-switching coefficients of the time series at time t
X_t^s	contains all predictor variables that have the switching effect of the time series at time t
β_{S_t}	are switching coefficients in state S_t, where $S_t \in \{1, ..., k\}$
ε_t	follows a Normal distribution with zero mean and variance given by $\sigma^2_{S_t}$

3.3.1. Autoregressive (AR) model

An autoregressive model is one type of time series models used to describe a time-varying process. The model is flexible in handling various kinds of time series patterns. The name autoregressive comes from how the model performs a regression of the variable against its own previous outputs (Cryer and Kellet, 1986). The number of autoregressive lags (i.e., the number of prior values used in the model) is denoted by p.

Definition 3.3.1. An autoregressive model of order p or AR(p) model can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t$$

or

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t$$

where c is a constant, ϕ_i are coefficients in the autoregression and ε_t is a Gaussian white noise with zero mean and variance σ^2.

If p is equal to one, the model AR(1) is called the first order autoregression process.

3.3.2. Markov switching autoregressive model

A Markov switching autoregressive model is an extension of a basic Markov switching model where observations are drawn from an autoregression process. The model relaxes the conditional independent assumption by allowing an observation to depend on both past observation and a current state (Shannon and Byrne, 2009). Basically, this is the combination between the Markov switching model and the autoregressive model.

Definition 3.3.2. The first order Markov switching autoregressive model is

$$y_t = X_t \beta_{S_t} + \phi_{1,S_t} y_{t-1} + \varepsilon_t$$

where ϕ_{1,S_t} is an autoregression coefficient of the observed value at time $t-1$ in state S_t. ε_t follows a Normal distribution with zero mean and variance given by $\sigma^2_{S_t}$.

The structure of the model is shown in Figure 3.3. It can be clearly seen that there is a dependency at the observation level.

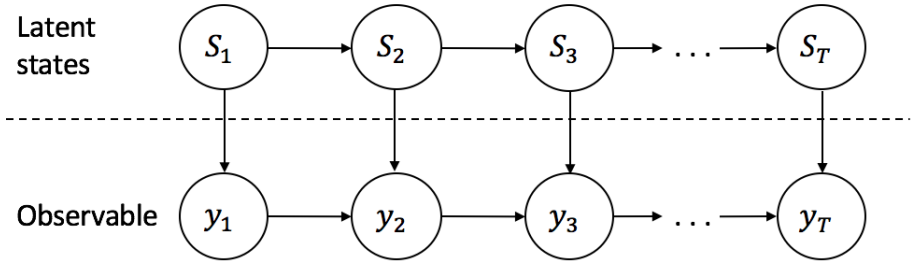


Figure 3.3.: Model structure of Markov switching AR(1)

Assuming two states $S_t = 1$ or 2, the set of parameters that are necessary to describe the law of probability that governs y_t are $\theta = \{\beta_1, \beta_2, \phi_{1,1}, \phi_{1,2}, \sigma_1^2, \sigma_2^2, \pi_1, \pi_2, p_{11}, p_{22}\}$.

3.4. Parameter estimation

There are various ways to estimate parameters of Markov switching model. Methods which have been widely used are as follow: E-M algorithm (Hamilton, 1990; Kim, 1994) used the maximum likelihood criterion, Segmental K-means (Juang and Rabiner, 1990) used K-means algorithm and maximized the state-optimized likelihood criterion, and Gibbs sampling (Kim et al., 1999) used a Markov chain Monte Carlo simulation method based on the Bayesian inference.

In this thesis framework, the E-M algorithm is used in estimating parameters as the algorithm gives an effective results, numerically stable, and easy to implement. Rydén et al. (2008) compared the computational perspective in estimating parameters between the E-M algorithm and the Gibbs sampling. In most cases, the Gibbs sampling tended to have less computational time than the E-M algorithm. However, the study indicated that if the number of states was unknown and only point estimate was sufficient, the E-M algorithm would typically be simpler and quicker solution in computing the estimated parameters. The E-M algorithm is briefly described below.

3.4.1. The Expectation-Maximization algorithm

E-M algorithm is originally designed to deal with the problem of incomplete or missing values in data (Dempster et al., 1977). Nevertheless, it could be implemented in Markov switching model since the unobserved state S_t can be viewed as missing data values.

The set of parameters θ are estimated by an iterative two-step procedure. In the first step, the algorithm starts with arbitrary initial parameters, and then finds the expected values of the state process from the given observations. In the second step of the iterative procedure, a new maximum likelihood from the derived parameters in the previous step is calculated. These two steps are repeated until the maximum value of the likelihood function is reached or converged (Janczura and Weron, 2012). The two steps are known as the E-step and the M-step. Figure 3.4 illustrates the process of the E-M algorithm.

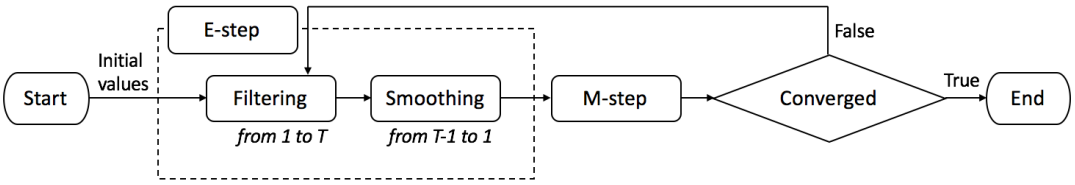


Figure 3.4.: A flowchart showing the process of the Expectation-Maximization algorithm. The algorithm begins with a set of initial values. The E-step is performed by computing a filtering and smoothing algorithm. Then, the M-step is performed. Iterating both steps until convergence.

3.4.1.1. E-step

In this step, $\theta^{(n)}$ is the derived set of parameters in M-step from the previous iteration, and n is a current iteration in the algorithm. The available observations of time $t-1$ is denoted as $\Omega_{t-1} = (y_1, y_2, ..., y_{t-1})$. The general idea of this step is to calculate

the expectation of S_t under the current estimation of the parameters. The obtained result is called smoothed inferences probability, and is denoted by $P(S_t = j|\Omega_T; \theta)$ where T is the number of all observations in the data and $j = 1, 2, ..., k$. The E-step which consists of filtering and smoothing algorithm is described as follows (Kim, 1994):

Filtering A filtered probability is a probability of a non-observable Markov chain being in a given state j at time t, conditional on information up to time t. The algorithm starts from $t = 1$ to $t = T$. A starting point for the first iteration where $t = 1$ is chosen from arbitrary values. The probabilities of each state given available observations up to time $t - 1$ is calculated by

$$P(S_t = j|\Omega_{t-1}; \theta^{(n)}) = \sum_{i=1}^{k} p_{ij}^{(n)} P(S_{t-1} = i|\Omega_{t-1}; \theta^{(n)}) \qquad j = 1, 2, ..., k \tag{3.3}$$

The conditional densities of y_t given Ω_{t-1} are

$$f(y_t|\Omega_{t-1}; \theta^{(n)}) = \sum_{j=1}^{k} f(y_t|S_t = j, \Omega_{t-1}; \theta^{(n)}) P(S_t = j|\Omega_{t-1}; \theta^{(n)}) \tag{3.4}$$

where $f(y_t|S_t, \Omega_{t-1}; \theta) = \frac{1}{\sqrt{2\pi\sigma_{S_t}^2}} exp\left\{-\frac{(y_t - \beta_{S_t})^2}{2\sigma_{S_t}^2}\right\}$ is the likelihood function in each state for time t. This is simply a Gaussian probability density function.

Then, with the new observation at time t, the probabilities of each state are updated by using Bayes' rule as shown below

$$P(S_t = j|\Omega_t; \theta^{(n)}) = \frac{f(y_t|S_t = j, \Omega_{t-1}; \theta^{(n)}) P(S_t = j|\Omega_{t-1}; \theta^{(n)})}{f(y_t|\Omega_{t-1}; \theta^{(n)})} \tag{3.5}$$

The process above is computed iteratively until all the observation is reached i.e., $t = T$.

Smoothing A smoothed probability is a probability of a non-observable Markov chain being in state j at time t, conditional on all available information. The algorithm iterates over $t = T - 1, T - 2, ..., 1$. The starting values are obtained from the final iteration of the filtered probabilities.

By noting that

$$
\begin{aligned}
P(S_t = j|S_{t+1} = i, \Omega_T; \theta^{(n)}) &\approx P(S_t = j|S_{t+1} = i, \Omega_t; \theta^{(n)}) \\
&= \frac{P(S_t = j, S_{t+1} = i|\Omega_t; \theta^{(n)})}{P(S_{t+1} = i|\Omega_t; \theta^{(n)})} \\
&= \frac{P(S_t = j|\Omega_t; \theta^{(n)}) p_{ij}^{(n)}}{P(S_{t+1} = i|\Omega_t; \theta^{(n)})}
\end{aligned} \tag{3.6}
$$

and

$$
P(S_t = j|\Omega_T; \theta^{(n)}) = \sum_{i=1}^{k} P(S_t = j, S_{t+1} = i|\Omega_T; \theta^{(n)}) \tag{3.7}
$$

then, the smoothed probabilities can be expressed as

$$
P(S_t = j|\Omega_T; \theta^{(n)}) = \sum_{i=1}^{k} \frac{P(S_{t+1} = i|\Omega_T; \theta^{(n)}) P(S_t = j|\Omega_t; \theta^{(n)}) p_{ij}^{(n)}}{P(S_{t+1} = i|\Omega_t; \theta^{(n)})} \tag{3.8}
$$

Full log-likelihood Once the filtered probabilities are estimated, there is enough necessary information to compute the full log-likelihood function.

$$
\ln L(\theta) = \sum_{t=1}^{T} \ln(f(y_t|\Omega_{t-1}; \theta^{(n)}) = \sum_{t=1}^{T} \ln \sum_{j=1}^{k} ((f(y_t|S_t = j, \Omega_{t-1}; \theta^{(n)}) P(S_t = j|\Omega_{t-1})) \tag{3.9}
$$

This is simply a weighted average of the likelihood function in each state. The probabilities of states are considered as weights.

3.4.1.2. M-step

The new estimated model parameters $\theta^{(n+1)}$ are obtained by finding a set of parameters that maximizes Equation 3.9. This new set of parameters is more precise and better than the previous estimated value of the maximum likelihood. $\theta^{(n+1)}$ serves as a set of parameters in the next iteration of the E-step.

Each individual parameter in $\theta^{(n+1)}$ are taken from its maximum value, which is determined by taking partial derivative of the log-likelihood function with respect to each parameter. Generally, this process is similar to the standard maximum likelihood estimation. However, it has to be weighted by the smoothed probabilities because each observation y_t contains probability from each k states.

3.4.1.3. Convergence of the E-M algorithm

The E- and M-step are iteratively computed until the algorithm converges. The algorithm will terminate when the different between the previous and current estimate values is less than a specific value. This specific value called a *stopping criteria* needs to be specified beforehand. The convergence is assured since the value of the log-likelihood function will increase in each iteration. However, the E-M algorithm does not guarantee to always converge to a global maximum. The convergence of the algorithm is also possible to be only a local maxima.

3.5. State prediction

A function to predict the most probable state for the new observation is implemented in R as an additional function in the package for this analysis (see sec. A.2).

The probabilities of being in state j at time $T+1$ on a basis of the current information are computed by performing the filtering algorithm in the E-step of E-M algorithm. The filtered probabilities are

$$P(S_{T+1} = j|\Omega_{T+1};\theta) = \frac{f(y_{T+1}|S_{T+1} = j, \Omega_T;\theta)P(S_{T+1} = j|\Omega_T;\theta)}{f(y_{T+1}|\Omega_T;\theta)}$$

This is Equation 3.5 where $t = T + 1$. Then, the new observation at time $T + 1$ is said to be in the state j if it has the highest probability.

3.6. Model selection

In this study, several Markov switching models will be carried out. First, the number of states k for the model will be chosen. Then, the number of switching coefficients in the model will be decided. Models will be selected based on the quality of the model.

Model selection is a task of selecting the best model for a given set of data. The Bayesian Information Criterion (BIC) is widely employed in the applied literature, and proved to be useful in selecting the model among a finite set of models (e.g., Leroux and Puterman (1992) used BIC to select the number of states k). It is also known as Schwarz Information Criterion (Schwarz et al., 1978). Model which has a lower value of BIC is preferred.

$$\text{BIC} = -2\ln(L(\hat{\theta})) + m \cdot \ln(T)$$

where $L(\hat{\theta})$ represents the maximized value of the likelihood function, T is the number of observations, and m is the number of parameters to be estimated in the model. While including more parameters or terms will result in a higher likelihood, it can also leads to an overfitting. BIC attempts to reduce the risk of overfitting by taking into account the number of parameters in the model. BIC can, therefore, heavily penalizes a model complexity.

3.7. Non-parametric analysis

A parametric analysis outperform a non-parametric analysis if the applied data belongs to a known distribution family. However, a parametric test does not perform well in detecting change points of an unknown underlying distribution (Sharkey and Killick, 2014). Applying a non-parametric analysis to a real-world process gives a real advantage to the analysis. Data collected from a real-world process, in general, usually does not have a well-defined structure, which is more suitable to be applied with the non-parametric analysis that is not too restricted (Hawkins and Deng, 2010). For this reason, the non-parametric analysis is implemented in order to get a rough idea of the change point location in this thesis framework. The obtained result is also compared with the result from using the Markov switching autoregressive model.

E-divisive

An *ecp*[1] is an extension package in *R* which mainly focuses on computing a non-parametric test for multiple change point analysis. This change point method is applicable to both univariate and multivariate time series. The fundamental idea of the package is based on the hierarchical clustering approach (James and Matteson, 2013).

An E-divisive method is an algorithm in the *ecp* package. This algorithm performs a divisive clustering in order to estimate change points. The E-divisive recursively partitions a time series and estimates a single change point in each iteration. Consequently, the new change point is located in each iteration, which divides the time series into different segments. The algorithm also uses a permutation test to compute the statistical significance of an estimated change point. The computational time of the E-divisive algorithm is $O(kT^2)$, where k is number of estimated change points and T is number of observations in the time series data. More details about the estimation is described in Matteson and James (2014).

[1] https://cran.r-project.org/web/packages/ecp/index.html

3.8. Simulation study for model evaluation

The state of the CPU utilization in a real data is unknown in the study. As a consequence, an accuracy of the Markov switching model and the E-divisive method cannot be computed, and the comparison between both methods can hardly make. One possible solution to test how effective both methods are, and to verify how well the implemented state prediction function performs is to used a simulation technique. The dataset that consists of two predictor variables and one response variable with already known states is simulated. The actual models of each state are

$$y_t = \begin{cases} 10 + 0.6X_{1,t} - 0.9X_{2,t} + 0.5y_{t-1} + \varepsilon_t^{(1)} & \varepsilon_t^{(1)} \sim N(0,1); \quad \text{Normal} \\ 2 + 0.8X_{1,t} + 0.2y_{t-1} + \varepsilon_t^{(2)} & \varepsilon_t^{(2)} \sim N(2,0.5); \quad \text{Bad} \\ -12 + 0.7X_{1,t} + 0.2X_{2.t} - 0.2y_{t-1} + \varepsilon_t^{(3)} & \varepsilon_t^{(3)} \sim N(1,1); \quad \text{Good} \end{cases}$$

where,

y_t is assumed to be a value of a CPU usage of the time series at time t

$x_{1,t}$ is a predictor variable generated by a uniform distribution on $[50, 200]$ of the time series at time t

$x_{2,t}$ is a predictor variable generated by a uniform distribution on $[0, 50]$ of the time series at time t

There are two simulated datasets – Dataset 1 and Dataset 2 – and each of them contains 500 observations. Both datasets have different time periods where the switches between states occur. The simulated Dataset 1 has a longer duration to remain in its own state before switching to the other states than the simulated Dataset 2. Figure 3.5 and Figure 3.6 present plots of y over a period of time, and the period where observations in the data belong to one of the state for the first and second simulated data, respectively.

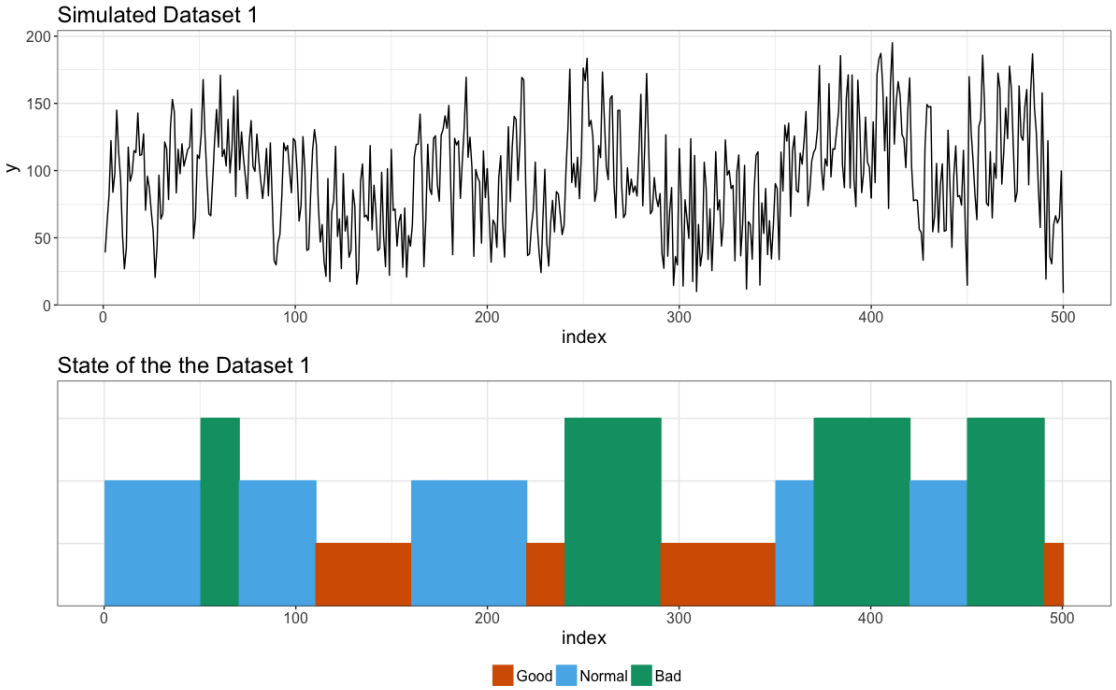


Figure 3.5.: *Top:* A simulated data of Dataset 1 where y variable is the response variable. *Bottom:* The period in the time series when observation is in each state.

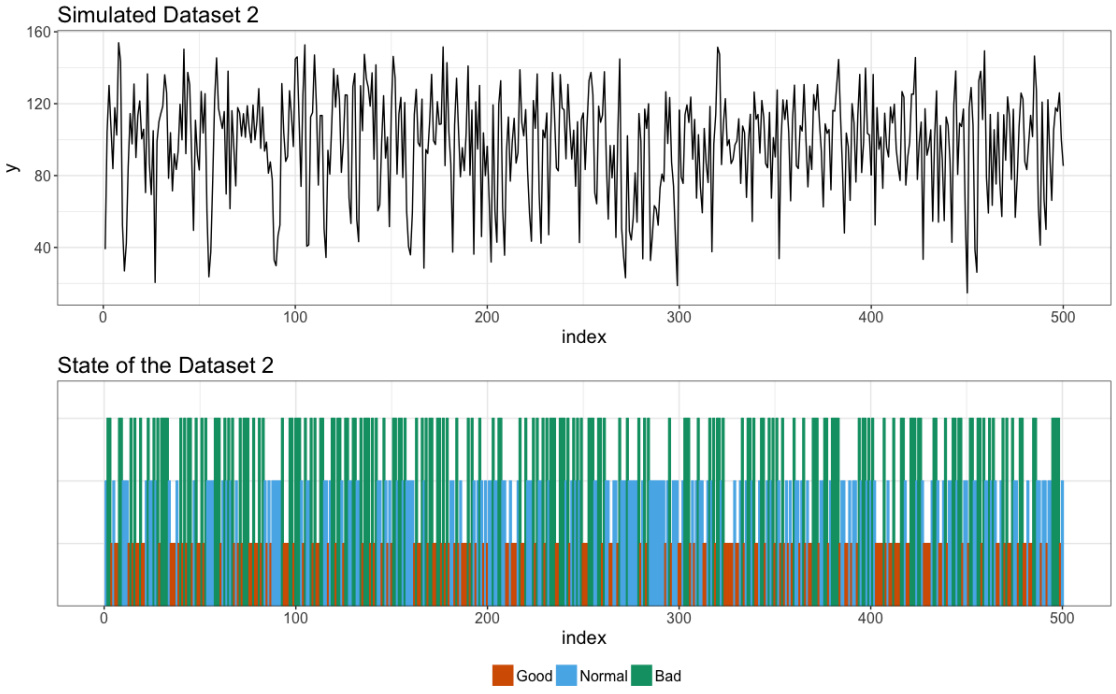


Figure 3.6.: *Top:* A simulated data of Dataset 2 where y variable is the response variable. *Bottom:* The period in the time series when observation is in each state.

3.9. Technical aspects

The thesis work has been carried out using the *R* programming language for the purpose of data cleaning, preprocessing, and analysis. The Markov switching model was performed using the *MSwM* package. Various extensions and modifications were implemented in the package (see sec. A.2). For the E-divisive method, the *ecp* package was used.

4. Results

The most relevant results of the analysis are shown and organized in this chapter. As a first step, the number of states of the model was decided (Analysis I). Then, the number of parameters that have switching effects in the model was determined (Analysis II). A model selection was performed for Analysis I and Analysis II in order to find the most appropriate model for each given dataset. An analysis of residuals was carried out as a means to validate the models. The results are shown in a later section. Next, the results of a non-parametric analysis are presented, and a comparison between the results of Markov switching model analysis and the results of non-parametric analysis are illustrated. The last two sections report the results of a state prediction of the new observations in each dataset, and an evaluation of the predicting function using a simulated data.

4.1. Analysis I: Number of States

To estimate the set of necessary parameters, an *MSwM*[1] package in R was used. More details about the package can be found in sec. A.2. A complete linear Markov switching autoregressive model in this thesis framework is defined as

$$\begin{aligned} y_t =& \beta_{intercept,S_t} + \beta_{RrcConnectionSetupComplete,S_t} X_{RrcConnectionSetupComplete,t} \\ &+ \beta_{Paging,S_t} X_{Paging,t} + \beta_{X2HandoverRequest,S_t} X_{X2HandoverRequest,t} \\ &+ \beta_{DuProdName,S_t} X_{DuProdName,t} + \beta_{Fdd/Tdd,S_t} X_{Fdd/Tdd,t} \\ &+ \beta_{NumCells,S_t} X_{NumCells,t} + \phi_{1,S_t} y_{t-1} + \varepsilon_{S_t} \end{aligned} \tag{4.1}$$

The estimation was made under the assumptions of two or three states $S_t \in S$, where $S = 1, 2, .., k$ and $k = 2$ or 3. These two numbers come from a hypothesis that the state of the CPU utilization might have two states (*Normal* and *Bad*, *Normal* and *Good*, *Bad* and *Good*) or three states (*Normal*, *Bad*, and *Good*). During the estimation, a normality assumption was also applied to the distribution of residuals.

BICs from fitting the Markov switching autoregressive model are shown in Table 4.1. For the software release A, the BIC suggests that the three-state Markov switching

[1] https://cran.r-project.org/web/packages/MSwM/index.html

autoregressive model gives a better fit in comparison to the two-state model. However, the models with two states for the remaining two software releases, B and C, had lower BICs.

Table 4.1.: BIC of the model with two and three states. The left column gives the different datasets.

Software release	BIC	
	$k = 2$	$k = 3$
A	439.677	417.682
B	1,763.507	1,797.259
C	1,189.061	1,199.075

4.1.1. Software release A

Before performing the Markov switching autoregressive model, a standard linear regression model was fitted to the dataset first. It was found that a coefficient of *DuProdName* in the dataset of the software release A was not defined because of singularity i.e., a perfect correlation between predictor variables. Hence, *DuProdName* variable was dropped from Equation 4.1.

Figure 4.1 presents that the Markov chain remained in State1 for an extensive period of time before it switched to State2. When the chain is in State2, it stays there only a short time and then quickly moves back to State1. There are a few switches between these two states in Figure 4.1. On the other hand, it is visible that there are more switches between states in Figure 4.2. Note that State2 in the two-state model seems to be defined as State1 in the three-state model instead. Moreover, the periods of State1, which has a rather long duration in the two-state model, now contains several switches between states in the three-state model.

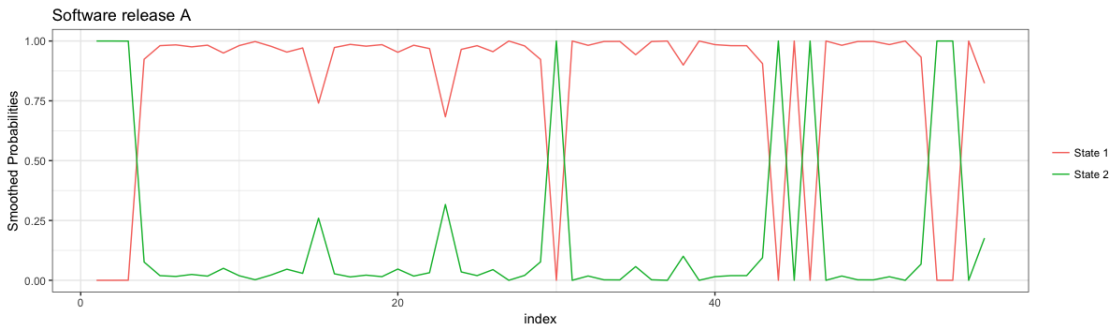


Figure 4.1.: The smoothed probabilities of the software release A with two-state model

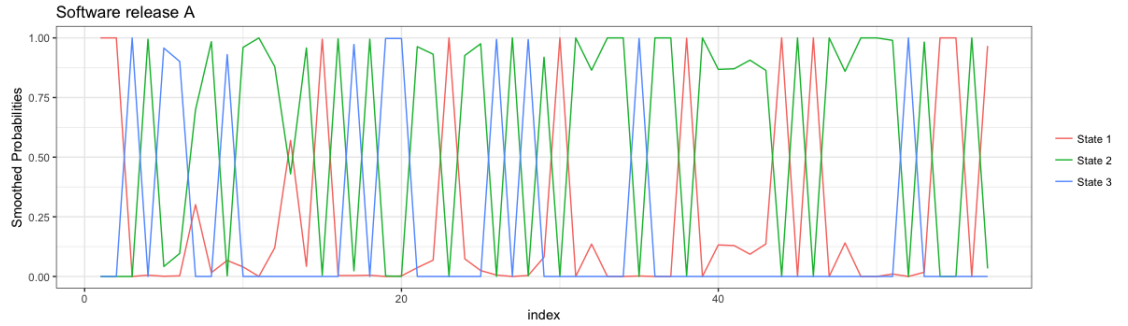


Figure 4.2.: The smoothed probabilities of the software release A with three-state model

4.1.2. Software release B

In Figure 4.3, the Markov chain has several periods where it switches back and forth between two states of the software release B. The durations of the chain being in State2 is longer than the durations of the chain staying in State 1. Although the chain temporarily stays in State1, it remains in this state for a few moments in the middle of the time period (observation 91-99 and 101-114) before returning to State2. Apparently, there are more switches between states in the three-state model, especially in the beginning, middle, and at the end of the period. Figure 4.4 shows that the chain remains in State3 over a considerable period as shown throughout observations 15-39, 42-67, and 140-170.

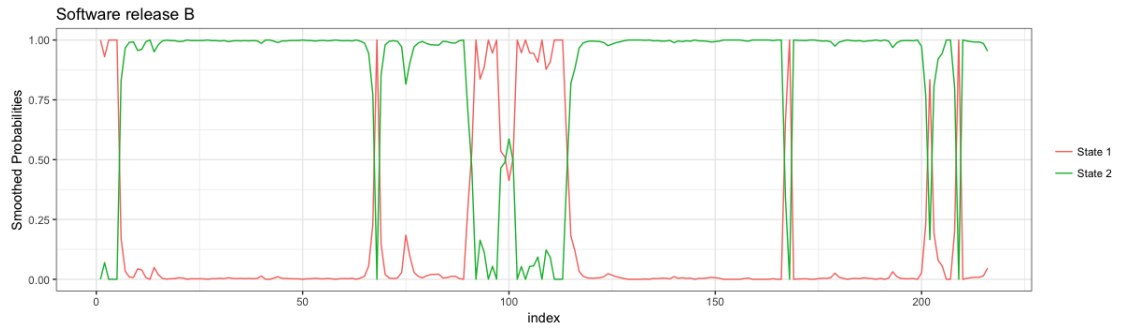


Figure 4.3.: The smoothed probabilities of the software release B with two-state model

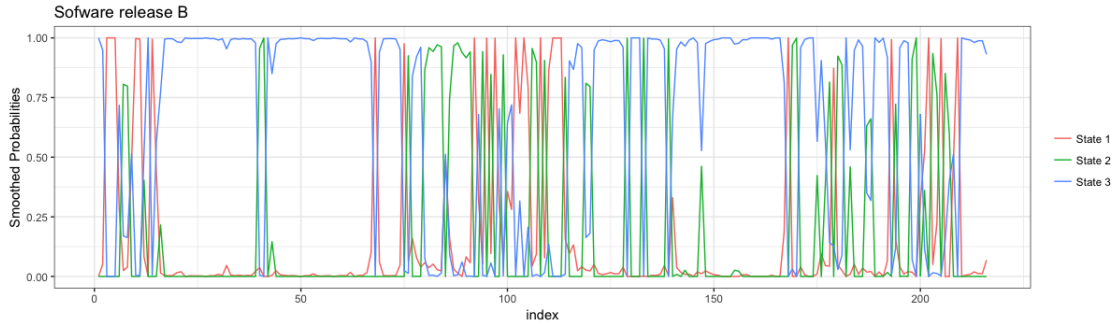
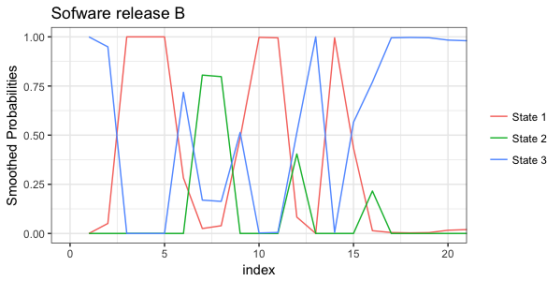
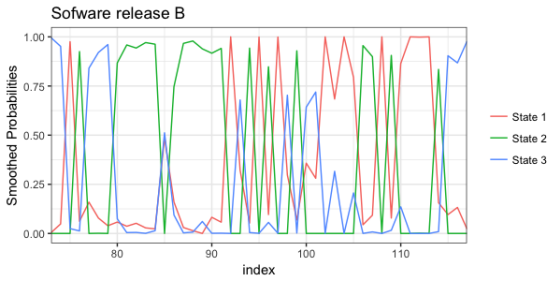


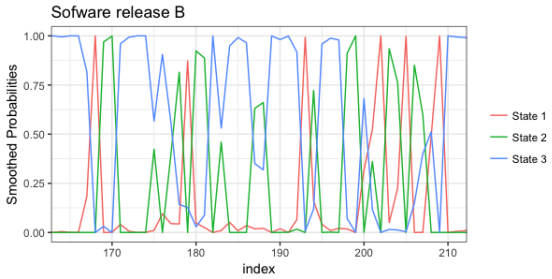
Figure 4.4.: The smoothed probabilities of the software release B with three-state model



(a) Close-up on the observations 0-20



(b) Close-up on the observations 75-115



(c) Close-up on the observations 165-210

4.1.3. Software release C

There are a number of switches between states in the two-state model of the software release C. In Figure 4.5, when the Markov chain is in State1, it continues to stay in its state for a while before leaving to State2. Furthermore, the chain has a fairly short duration of staying in State2. After the chain visits State2, it instantly switches back to State1. Figure 4.6 presents the chain which has many switches between State1 and State2 in the first half of the time period. The chain for the three-state model also stays in State2 significantly long from observation 104 to 129, which is the end of the time series.

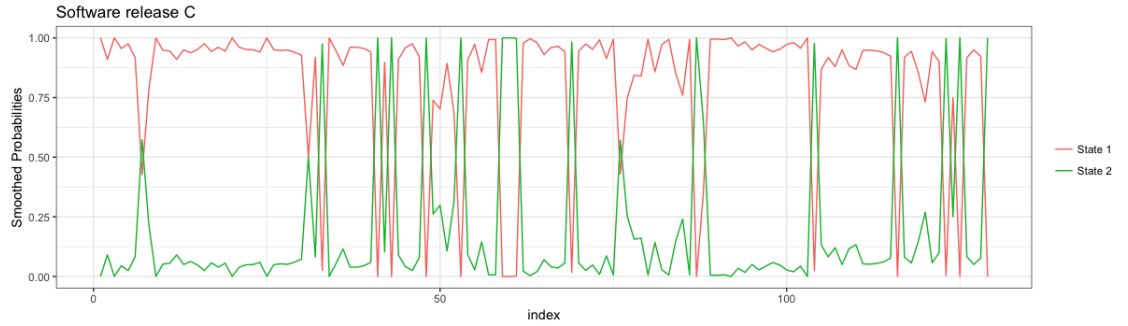


Figure 4.5.: The smoothed probabilities of the software release C with three-state model

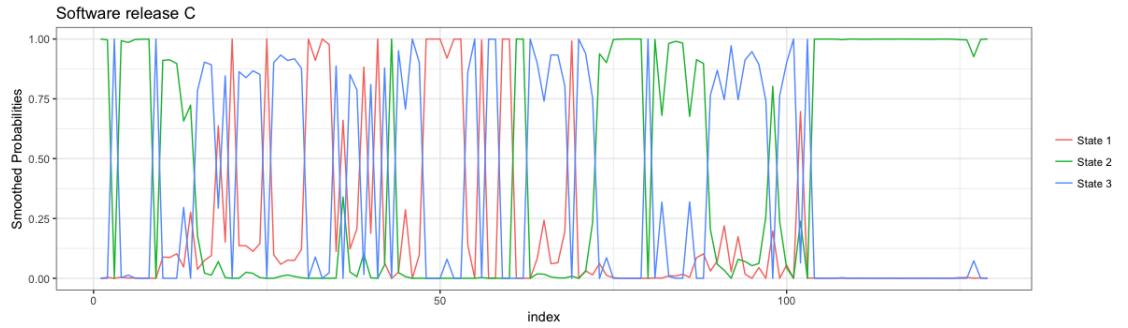
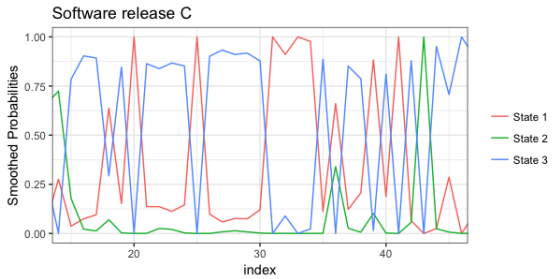


Figure 4.6.: The smoothed probabilities of the software release C with three-state model



(a) Close-up on the observations 15-45

After examining the outputs from the models along with the plots, the three-state models for each software release were further analyzed in the thesis. More details are provided in chapter 5.

4.2. Analysis II: Number of Switching coefficients

The fitted Markov switching autoregressive models in sec. 4.1 were performed by assuming that every parameter in the model had switching effects i.e., coefficients

can have different values in different periods. However, in practice, each coefficient can have either a switching or non-switching effect. Therefore, Markov switching autoregressive models were applied to each dataset again but with a hypothesis that the variables considered as a test environment are possible to have non-switching effects. In this section, the structure of all the models from all three datasets are reported in the tables. The best model is selected for each dataset and its state specification is presented in the plots. Further discussion and details about these chosen models are provided in sec. 5.1. It should be noted that these three chosen models will later be used throughout this thesis, and the model outputs are shown in Appendix B.

4.2.1. Software release A

For the dataset of the software release A, *DuProdName* was not included in the model fitting as explained previously. Only two variables of the test environment were left to try whether they could have non-switching effects or not. The result is shown in Table 4.2. The second model has the highest BIC and even higher than the model which have all switching coefficients. The first model, where both *Fdd/Tdd* and *NumCells* have switching effects, was selected to be used with this dataset.

Table 4.2.: List of the model structure of the software release A along with its BIC. The last line is the result taken from the three-state model in the Analysis I. The line in bold indicates the selected model.

Model	Switching effect		BIC
	Fdd/Tdd	NumCells	
1	**N**	**N**	**413.408**
2	N	Y	438.371
3	Y	N	401.232
	Y	Y	417.682

Figure 4.7 indicates the CPU utilization of the software release A and also shows the periods of the derived state from the model. From the plot, State2 clearly has the longest duration to remain in its own state. When the chain moves to either State1 or State3, it immediately switches to the other states. However, the duration that the chain stays in State1 is longer in the beginning and almost at the end of the period. Another characteristic that could be observed is that State2 have more chance to switch to State3 rather than switch to State1. In the plot, there is a period where there is no significant change in the CPU utilization (observations 15-25) but there are some switches between states. Besides, there are some abrupt changes which are not detected by the model such as observation 11.

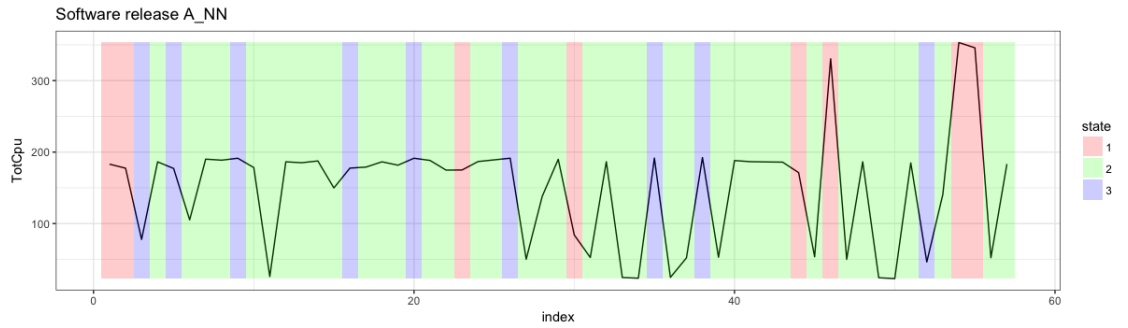


Figure 4.7.: The CPU utilization of the software release A showing the periods where the observation is in the specific state.
Model 1: Fdd/Tdd and Numcells are non-switching coefficients.

4.2.2. Software release B

For the software release B, Table 4.3 presents the results of fitting the model with different combinations of switching coefficients. Models 5 and 7 have higher BICs than the model which have switching effects in all coefficients. The second model, where *DuProdName* and *Fdd/Tdd* are non-switching coefficients, has the smallest BIC. The chosen model for this dataset is the model which has only *DuProdName* as a non-switching coefficient or model 4.

Table 4.3.: List of the model structure of the software release B along with its BIC. The last line is the result taken from the three-state model in the Analysis I. The line in bold indicates the selected model.

Model	Switching effect			BIC
	DuProdName	Fdd/Tdd	NumCells	
1	N	N	N	1787.528
2	N	N	Y	1704.393
3	N	Y	N	1784.384
4	**N**	**Y**	**Y**	**1776.102**
5	Y	N	N	1806.385
6	Y	N	Y	1725.865
7	Y	Y	N	1804.487
	Y	Y	Y	1797.259

Many switches between states can easily be seen in Figure 4.8. However, the state which has the longest duration remaining in its own state is State3. There are three durations where the chain stays in State3 for a long period of time. Another noticeable behavior from this switching mechanism is that there are several switches

between State1 and State2 in the beginning, middle, and at the end of the time period. Both false alarms and missed detections can be seen in the plot. There are periods where the CPU utilization value does not change much but the model identifies some switches. Also, there are periods where the model fails to detect changes which is rather obvious.

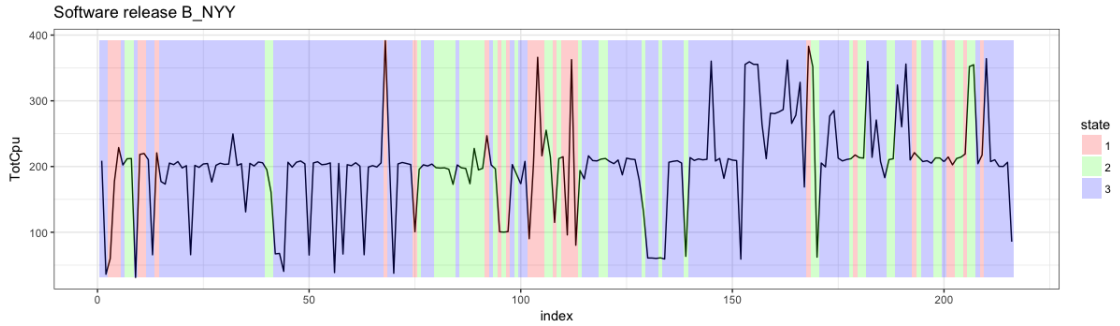


Figure 4.8.: The CPU utilization of the software release B showing the periods where the observation is in the specific state.
Model 4: DuProdName is non-switching coefficient.

4.2.3. Software release C

Table 4.4 presents model structure of the software release C. Only model 2 has higher BIC than the model which have all switching coefficients. The least BIC is from the first model that all three variables in the test environment have non-switching effects. This model was also chosen to be further used for this dataset.

Table 4.4.: List of the model structure of the software release C along with its BIC. The last line is the result taken from the three-state model in the Analysis I. The line in bold indicates the selected model.

Model	Switching effect			BIC
	DuProdName	Fdd/Tdd	NumCells	
1	**N**	**N**	**N**	**1140.474**
2	N	N	Y	1204.280
3	N	Y	N	1152.740
4	N	Y	Y	1184.643
5	Y	N	N	1146.000
6	Y	N	Y	1189.236
7	Y	Y	N	1157.311
	Y	Y	Y	1199.075

Several switches between three states occur in the beginning of the time series as shown in Figure 4.9. Around the end of the time series period, State3 appears to have a longer duration and fewer switches to State1. State2 seems to be the only state which has a fairly short duration for the chain to stay in the state. Furthermore, State2 tends to switch to State1 more often than to switch to State3. The plot indicates some missed detections for this model which is happened mostly in the durations of State3.

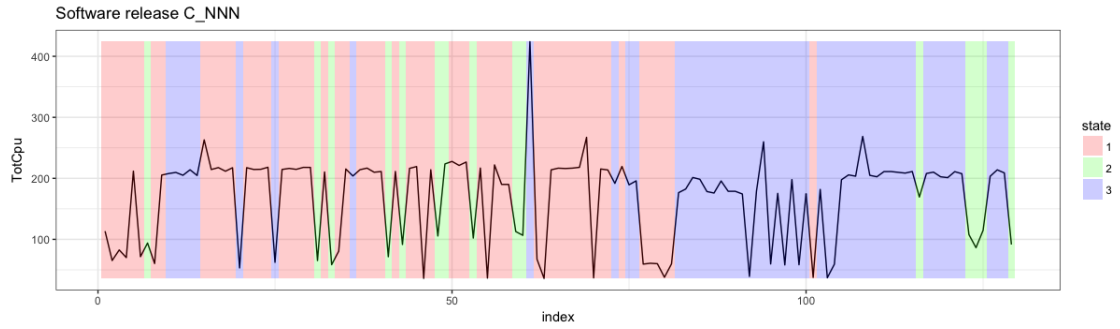


Figure 4.9.: The CPU utilization of the software release C showing the periods where the observation is in the specific state.
Model 1: DuProdName, Fdd/Tdd, and NumCells are non-switching coefficients.

4.3. Residual analysis

Pooled residuals of the selected Markov switching autoregressive model from sec. 4.2 were analyzed to determine how well the model fitted with an assumption of a normal distribution. A Quantile-Quantile (Q-Q) plot is an effective tool for assessing normality. Moreover, an Autocorrelation function (ACF) and a Partial Autocorrelation Function (PACF) of residuals are a useful technique to check on the independence of noise terms in the model. The Q-Q plot and the ACF/PACF plots play a significant role in the residual diagnostics. These plots of each dataset are shown in Figure 4.10, Figure 4.11, and Figure 4.12.

4.3.1. Software release A

In Figure 4.10, the pooled residuals appear to fall in a straight line with some deviations in its tails. There is an evidence of autocorrelation in the residuals of this model, which can be seen in both ACF and PACF plot, at lag 8.

4.3.2. Software release B

Figure 4.11 presents points that form a straight line in the middle of the plot, but curve off at both ends. This is a characteristic of a heavy-tailed distribution. The

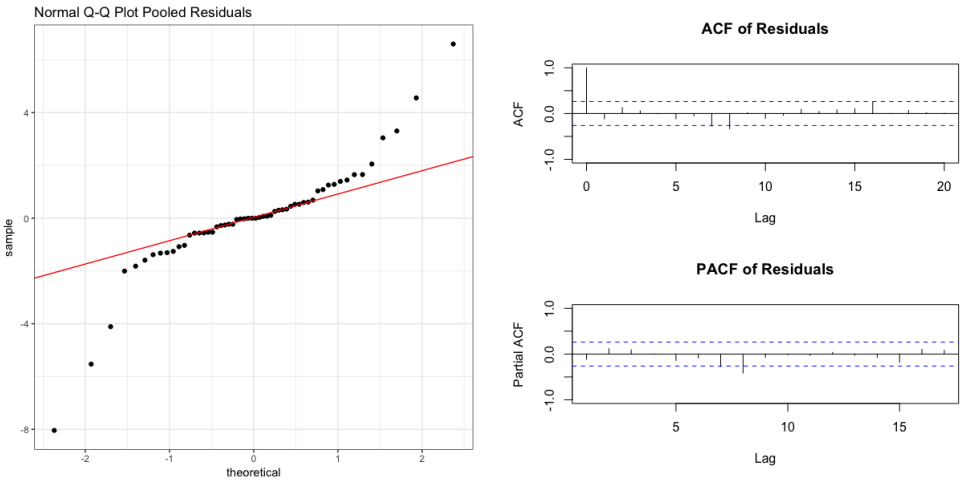


Figure 4.10.: The normal Q-Q plot and the ACF/PACF of pooled residuals of the software release A

data has more extreme values than it should if the data truly comes from a normal distribution. In addition, both the ACF and PACF plots show that there is a small amount of autocorrelation remain in the residuals. The statistically significant correlation of this model are at lags 6 and 10. The significance at lag 4 both in the ACF and PACF plots is slightly higher than two standard errors.

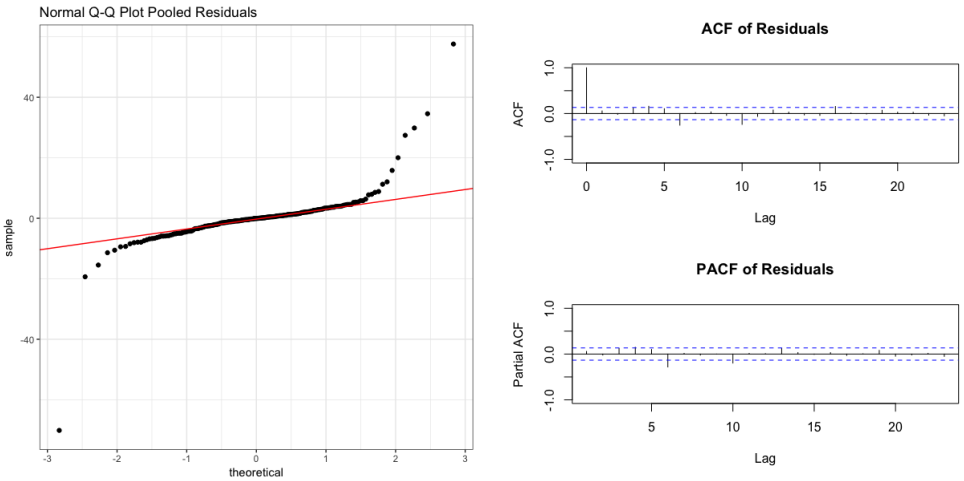


Figure 4.11.: The normal Q-Q plot and the ACF/PACF of pooled residuals of the software release B

4.3.3. Software release C

A Q-Q plot in Figure 4.12 suggests that a distribution of the pooled residuals may have a tail thicker than that of a normal distribution. It is visible that there are

many extreme positive and negative residuals in the plot. Furthermore, the ACF plot of pooled residuals are significant for the first two lags, whereas the PACF plot is significant only at lag 2.

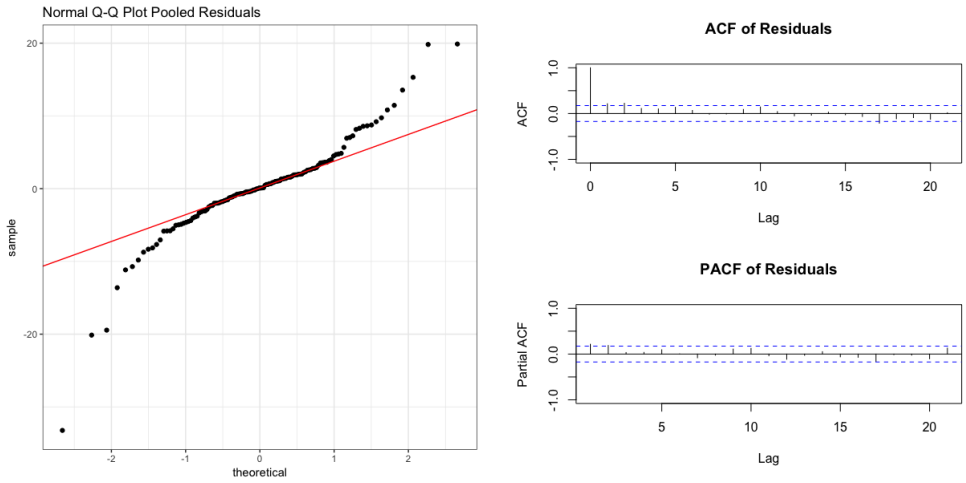


Figure 4.12.: The normal Q-Q plot and the ACF/PACF of pooled residuals of the software release C

4.4. Non-parametric analysis

An E-divisive method was applied to all three datasets. The method reported one cluster for the dataset of the software release A and five clusters for the dataset of each software release B and C. Table 4.5 shows places in the time series data where the E-divisive method detected the significant changes.

Table 4.5.: The locations of the statistically significant change points from applying the E-divisive algorithm in each dataset

Software release	Change-point location
A	-
B	130, 135, 153, 170
C	9, 77, 82, 105

The CPU utilization of the software release A, B and C along with its estimated change points in the time series are plotted and shown later in sec. 4.5.3.

4.5. Comparison between the Markov switching model and the E-divisive method

A comparison between the Markov switching model and the E-divisive method was made in this section. Two methods were applied to two simulated datasets, where the actual changes are already known beforehand, and then also applied to a real data.

4.5.1. Simulated Dataset 1

Figure 4.13 illustrates that the simulated data contains nine estimated change point locations. For the Markov switching model, the model reported two extra locations. The plot illustrates that, besides the extra detected, the model performs rather well, as the model discovers all the changes accurately. Only one change point location was indicated to occur later than its actual occurrence time. In contrast, the E-divisive method detected changes fewer than the actual changes in the data. Furthermore, most of the detection are also not quite accurate as three out of six detections are delayed and out one of six detections is indicated to happen prior to its actual time. The E-divisive method is unable to detect any changes in the data at the beginning of the time period.

When these two methods detect changes after the actual changes, most of their estimated change point locations are only behind by one or two time index. To sum up, from this dataset, the Markov switching model had more false alarms while the E-divisive had more missed detections.

4.5.2. Simulated Dataset 2

Figure 4.14 presents estimated change point locations of the Markov switching model, the E-divisive method, and the actual change points in the data. The simulated dataset contains numerous switches between states. Generally, the Markov switching model is able to identify the changes considerably well despite a few false alarms and missed detections. On the other hand, the E-divisive method can detect only estimated change points. Both two detections were correctly located. The method has quite poor performance as it failed to detect most of the change points in the data. The difference can be seen in the plot.

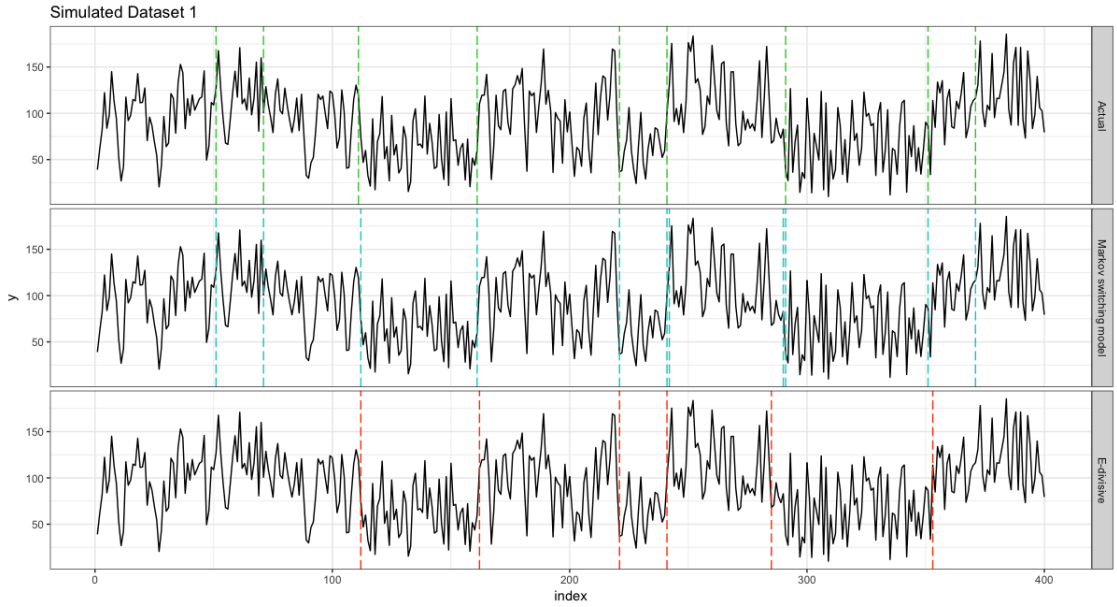


Figure 4.13.: The simulated Dataset 1 showing the estimated change point locations indicated by dashed vertical lines from the Markov switching model (Middle) and the E-divisive method (Bottom). The actual change point locations in the data are indicated in the top plot.

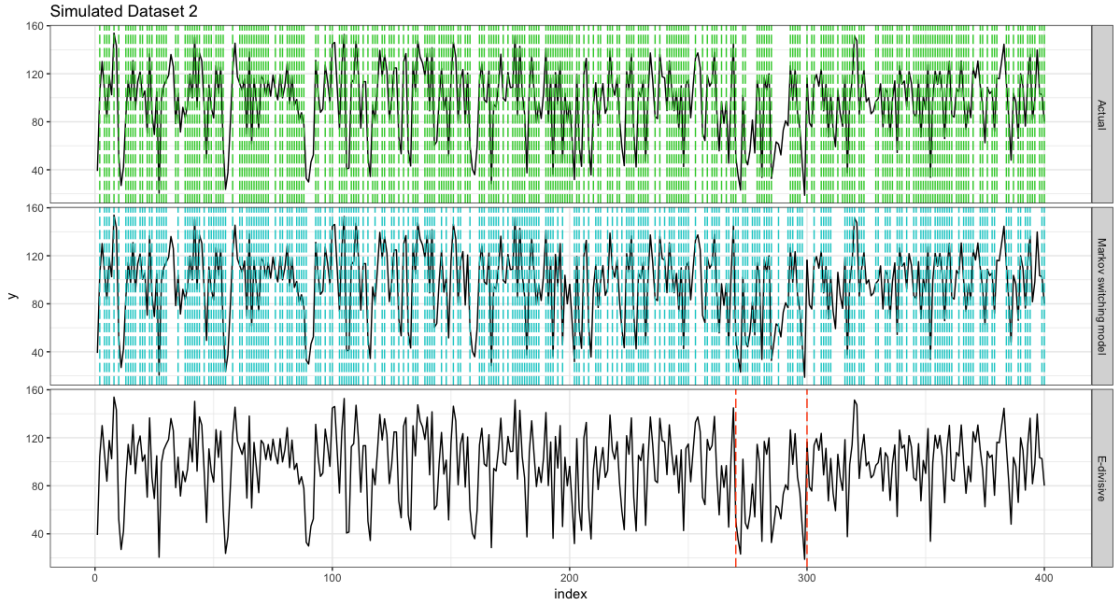


Figure 4.14.: The simulated Dataset 2 showing the estimated change point locations indicated by dashed vertical lines from the Markov switching model (Middle) and the E-divisive method (Bottom). The actual change point locations in the data are indicated in the top plot.

4.5.3. Real data

4.5.3.1. Software release A

According to Table 4.5, the E-divisive method could not identify any changes in the dataset of the software release A. Thus, a comparison between two methods could not be made for this dataset.

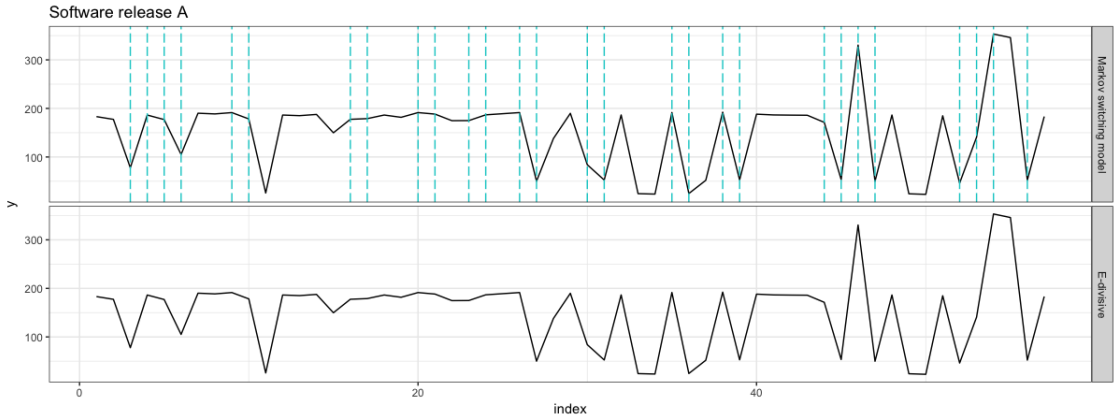


Figure 4.15.: The CPU utilization of the software release A. The dashed vertical lines indicate the locations of estimated change points from the Markov switching model (Top).

4.5.3.2. Software release B

Figure 4.16 presents results of estimated change points from the Markov switching model and the E-divisive method for the software release B. Sixty three estimated change point locations are found by the Markov switching model. On the contrary, only four estimated change point locations are identified from the E-divisive method. They are likely to occur around the same period of time. Most of the locations of the change point detected by the E-divisive method are at peaks and negative peaks. Apparently, only one change point location is determined at the exact same time from both methods, and two change points are closely located. The rest detected locations are completely different for each method.

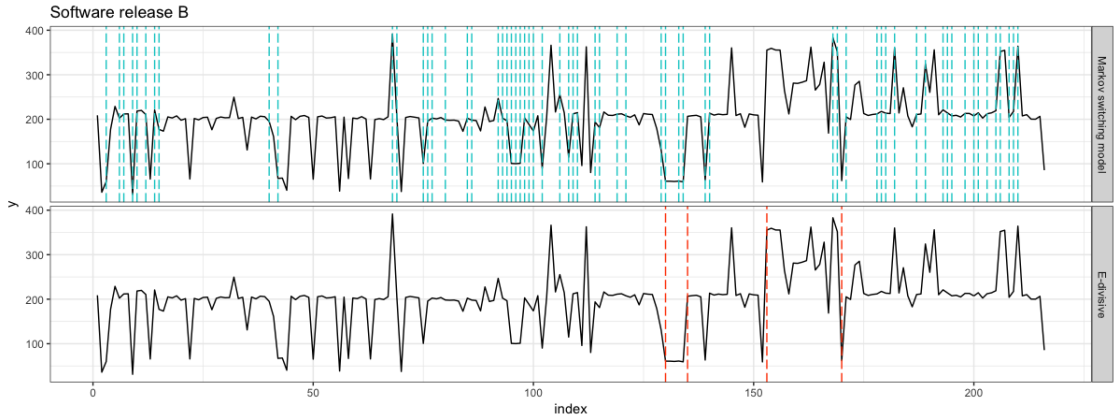


Figure 4.16.: The CPU utilization of the software release B. The dashed vertical lines indicate the locations of estimated change points from the Markov switching model (Top) and the E-divisive method (Bottom).

4.5.3.3. Software release C

For the software release C, thirty-seven changes were discovered by the Markov switching model as can be seen in Figure 4.17. However, only four change point locations were identified from the E-divisive method. These locations are rather spread out if compare to the results from the software release B. The E-divisive method discovered changes when the CPU utilization value was about to decrease or increase. From both methods, two change points are determined at exactly the same locations, and another one which is located in the beginning is close to each other. Other detect locations are totally different as shown in the plot.

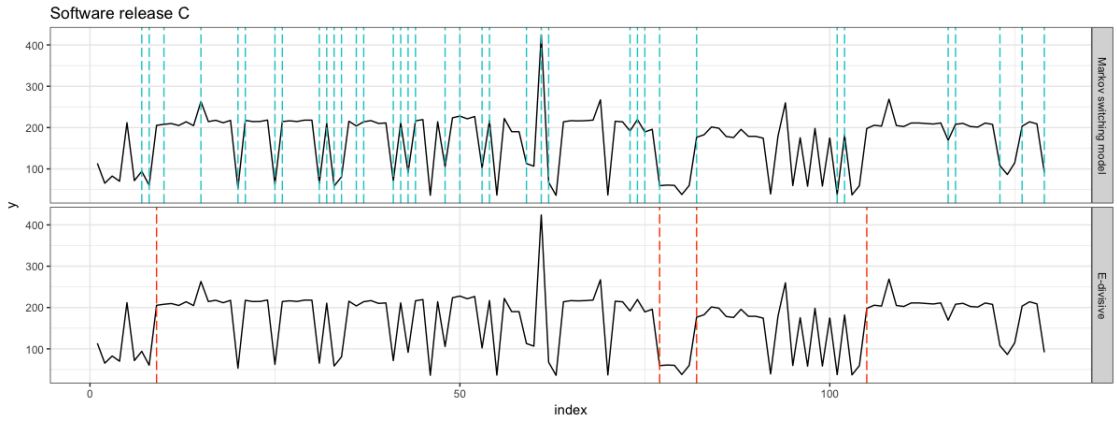


Figure 4.17.: The CPU utilization of the software release C. The dashed vertical lines indicate the locations of estimated change points from the Markov switching model (Top) and the E-divisive method (Bottom).

4.6. Predicting the state of the CPU utilization

In this section, the state prediction function was implemented to the test set in order to predict the most probable state for new observations.

4.6.1. Software release A

For the software release A, there are 7 observations in total. In Figure 4.18, only two states, State1 and State2, are assigned for these observations. The first three observations are in State2. Afterwards, observation tends to switch back and forth between both states until the end of the time period. Note that the most likely state for the last observation of the test set is unable to be predicted, and so it does not belong to any state.

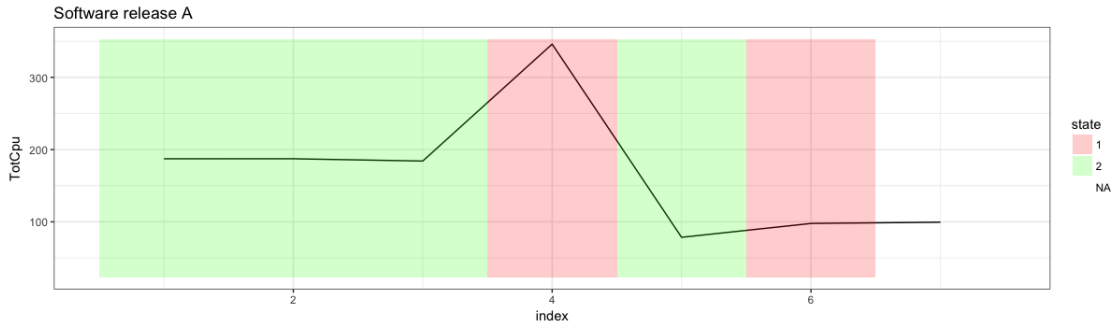


Figure 4.18.: The predicted state of the test set in the software release A

4.6.2. Software release B

In total, there are 25 observations in the test set of the software release B. The result after applying the predict function to the test set is shown in Figure 4.19. Observation 15 is the only observation which is in State2. Many switches between State1 and State2 can be seen from the plot. In addition, observation appears to stay in State1 only a short time before moving to State3, except for the first five observations.

4.6.3. Software release C

The test set of the software release C consists of 15 observations in total. Figure 4.20 illustrates that the observations stay in State2 for a considerably long period of time (observation 10-15). There are several switches between states shown in the plot. The time period between the observations between 4 and 7 switch between states rapidly. The observations visit a particular state for one time, and then immediately move to the other states.

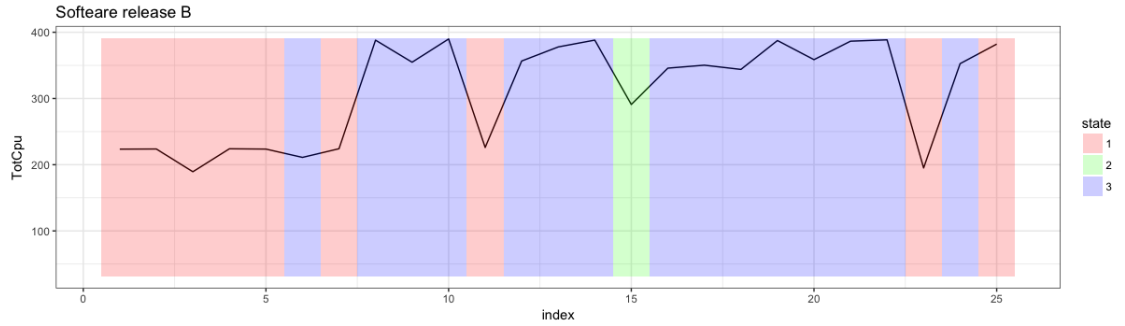


Figure 4.19.: The predicted state of the test set in the software release B

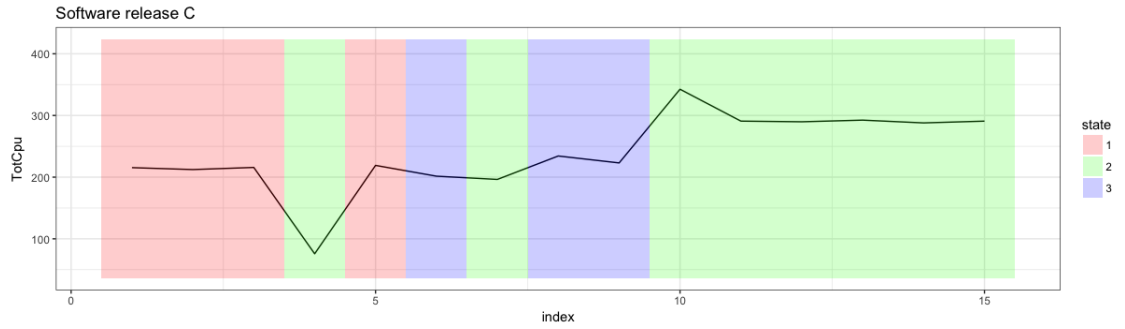


Figure 4.20.: The predicted state of the test set in the software release C

4.7. Model evaluation

Eighty percents of the observations from the simulated data were fitted with Markov switching autoregressive model, and the remaining was used as a test set to evaluate a performance of the model.

4.7.1. Simulated Dataset 1

Table 4.6 presents outputs from applying the Markov switching model with the simulated Dataset 1. Each state has a considerably high r-squared value which is greater than 0.9. From the result, State2 has the highest standard error. When inferring back to the actual models that was used to generate these three states in sec. 3.8, it could be concluded that the result State1, State2, and State3 in the table are a *Bad*, *Normal*, and *Good* state, respectively.

Table 4.6.: Outputs from applying the Markov switching model with the simulated Dataset 1. The table shows estimated coefficients, residual standard error, and r-squared for each state. A switching coefficient is followed by (S), and a significant coefficient is highlighted in bold.

Estimated coefficient	State 1	State 2	State 3
(Intercept)(S)	**1.3024**	**5.5339**	**-14.3116**
x1(S)	**0.8005**	**0.5900**	**0.7005**
x2(S)	0.0003	**-0.8610**	**0.1941**
y_1(S)	**0.1902**	**0.3843**	**-0.1927**
Residual standard error	1.4900	7.3013	1.8255
r^2	0.9980	0.9467	0.9962

The result of the model performance using Dataset 1 is shown in Table 4.7. There are two observations from a *Bad* state which were incorrectly predicted to be in a Normal state. Moreover, two more observations from a *Good* state were predicted to be in a *Normal* state. The overall accuracy of the model is 0.96, and the misclassification rate is 0.04. One can see that the model was able to perfectly predict the state of the observations that are in a *Normal* state.

Table 4.7.: The confusion matrix of the result from the test set of the simulated Dataset 1

		Predicted state		
		Bad	Normal	Good
Actual state	Bad	58	2	0
	Normal	0	30	0
	Good	0	2	8

4.7.2. Simulated Dataset 2

Outputs after applying the Markov switching model with the simulated Dataset 2 are presented in Table 4.8. The r-squared values of State1 and State3 are slightly lower than the obtained result from the simulated Dataset 1, but the standard errors are significantly higher. However, State2 has close output value to the simulated Dataset1. It could be said that the model still performs rather well as the r-squared value is very high. State1 is said to be a *Bad* state although an intercept is insignificant. State2 and State3 are a *Good* and *Normal* state, respectively.

Table 4.8.: Outputs from applying the Markov switching model with the simulated Dataset 2. The table shows estimated coefficients, residual standard error, and r-squared for each state. A switching coefficient is followed by (S), and a significant coefficient is highlighted in bold.

Estimated coefficient	State 1	State 2	State 3
(Intercept)(S)	5.0716	**9.0219**	**23.1711**
x1(S)	**0.8672**	**0.5401**	**0.5546**
x2(S)	-0.0195	**0.2599**	**-1.1239**
y_1(S)	**0.1232**	**-0.0882**	**0.2272**
Residual standard error	6.6367	5.3211	7.5079
r^2	0.9089	0.9480	0.9337

Table 4.9 presents a confusion matrix for a test set from a second simulated dataset. The model was able to correctly predict all the observations in a *Bad* state. On the contrary, the model did not perform well in predicting observations which had *Good* and *Normal* state. Nine observations from a *Good* state were predicted to be in a *Bad* state, and another five observations from a *Good* state were inaccurately predicted to be in a *Normal* state. Six observations from a *Normal* state were incorrectly predicted to be in a *Good* state. The overall accuracy of the model and the misclassification rate are 0.8 and 0.2, respectively.

Table 4.9.: The confusion matrix of the result from the test set of the simulated Dataset 2

		Predicted state		
		Bad	Normal	Good
Actual state	Bad	35	0	0
	Normal	0	29	6
	Good	9	5	16

5. Discussion

In this chapter, a discussion of the model selection for each given dataset is explained. Then, a state inference from the model is made and effects of the test environments are provided. Lastly, a results discussion of this study is discussed.

5.1. Model selection

In this analysis, a three-state model which had all switching coefficients from Analysis I (see sec. 4.1) acted as a baseline model for each dataset. This model was used to compare with other models which had different combination of switching coefficients. If the compared model has higher BIC than the baseline model, its performance is concluded to be inferior and should not be considered for any further analysis. However, only examining the BIC when choosing a model for the data is insufficient. Remark that BIC might be unreliable for a small or moderate dataset as BIC is derived under the assumption of a normal distributed data (Rydén et al., 2008). Therefore, other aspects should also be taken into account along with the BIC such as model outputs and plots.

Software release A Even though model 3 had the lowest BIC, a coefficient of *Paging* in one state had a zero standard error which led t-value to infinity. This zero value can be interpreted as an actual zero or an extremely small value that a computer treated as zero because significant digit is lost. Nevertheless, either way suggests that this model is not a good model to be used with this dataset as the model might be overfitting with the training data. The standard error of zero means that there is no variation in the data i.e., every data value is equal to the mean value. Therefore, model 1 which had the second lowest BIC was chosen for this given dataset instead.

Software release B Models 2 and 6 had the lowest BIC among the other models. Nonetheless, their plots are similar to each other and provided a difficulty in interpreting results (see Figure B.1). Observations stay in State3 in the first half of the period but stay in State2 in the second half of the time period. There are continuous fluctuations of the CPU utilization value through out the whole time period. Therefore, for observations (or software packages) to remain in the same

state for a long duration without switching to other states seem unrealistic. The selected model for this dataset was model 4 where its BIC was in the third lowest place. Even though model 4 had slightly higher BIC than models 2 and 6, the model produced more sensible result.

Software release C The first four models with lowest BIC (Models 1, 5, 3, and 7, respectively) had similar results both in model outputs and plots. Thus, Model 1 which had the lowest BIC was chosen for this dataset.

5.2. State inference

The model output after applying the Markov switching model with the dataset does not provide any definition of the derived states. Therefore, an interpretation and inference of these states need to be specified. The state inferences for each dataset are shown below.

Software release A (see Figure 4.7) Despite periods where there is a slightly decrease in CPU utilization value, State1 contains two peaks which have the value of the CPU utilization higher than 300. This clearly implies that test cases in State1 perform badly and should be defined as a *Degradation* state. As for State2 and State3, both states can be viewed as an Improvement state. However, State2 seems to have many periods where test cases drop to a lower value in the CPU utilization. State3 also has two periods of abrupt changes to higher CPU utilization values. Therefore, State2 appears to be an *Improvement* state while State3 is said to be a *Steady* state.

Software release B (see Figure 4.8) There are many high peaks occurred in State1, and test cases tend to increase the values of the CPU utilization when they enter this state. Hence, State1 is said to be a *Degradation* state. State2 could be characterized as an *Improvement* state. The reason is that a decreasing pattern of the CPU utilization value for the test cases in State2 can be seen in most of the period. State3 contains the period of positive and negative peaks, and does not appear to have a specific characteristic for the state of the CPU utilization. The difficulty of making an interpretation for State3 infers that the state may be defined as a *Steady* state.

Software release C (see Figure 4.9) Abrupt changes in the CPU utilization values when test cases enter State1 could be seen in most of the period. Despite low values in some of the test cases, State1 appeared to have more high value in the CPU utilization. Hence, State1 could be defined as a *Degradation* state. The CPU

utilization values clearly decrease when test cases enter State2. These test cases often switch from State1, which is thought to be the state where the performance of the test cases perform badly. As a consequence, this behavior indicates an improving in the test cases. Thus, State2 is defined as an *Improvement* state. The test cases which stay in State3 have an unclear pattern in the CPU utilization. The values are quite fluctuate and because of this behavior, the state is labeled as a *Steady* state. The only peak in the time series data belonged to State3, and will be viewed as an anomaly in this case.

5.3. Test environment

This section contains a discussion about the effects of the test environments - *DuProdName*, *Fdd/Tdd*, and *NumCells* - to the CPU utilization in each dataset.

Software release A (see Table B.1) There are only two variables, *Fdd/Tdd* and *NumCells*, in the test environments that were included in the model. The output indicates that both variables are statistically significant. Therefore, changes in the test environments will have an impact on the CPU utilization.

Software release B (seeTable B.2) All test environment variables are statistically significant, except for the *Fdd/Tdd* in State1. Hence, it can be concluded that changes in *DuProdName* and *NumCells* have significant effects on the CPU utilization. *Fdd/Tdd* will affect the CPU utilization value when a test case is in State2 or State3.

Software release C (see Table B.3) According to the output, *DuProdName* obviously has a significant effect on the CPU utilization whereas *Fdd/Tdd* does not have any. It could also be said that *NumCells* has somewhat an impact on the CPU utilization; however, this is not always the case and should have been taken with care.

5.4. Results discussion

A thorough search of relevant literature yielded that this thesis work might be the first time that Markov switching model has been applied to the data of CPU utilization. In previous works, this model was mainly applied with financial data or signal processing which were used as an inspiration for this study. However, because of differences in the characteristic of data, the procedure was slightly adjusted.

In this study, *NodeName*, which is used to execute test cases, was assumed to be indifferent i.e., performance of test cases are the same regardless of the machine. Therefore, selecting a minimum value of the CPU utilization of the test case for each software package in data preprocessing step is reasonable.

There is no fixed rules in deciding the number of states. Therefore, difference number of states were tried with the method in order to determine which number would be best for this analysis. The results from the model with two states offered less details and did not cover all happening situations. Furthermore, all the graphs of the two-state model are unrealistic and also difficult in interpret. Despite higher BICs for the dataset of the software release B and C, three-state model provides more interpretable plots and better fit to the time series data. The four-state Markov switching model for each software release was also briefly tested in the analysis. Apparently, defining four states caused more difficulty when fitting the model and making an inference on the states. The results were rather poor when compare to the model with two and three states and was not include in the report. Higher number of states are more likely to give worse results and were not considered. Thus, the number of chosen states after applying Markov switching model to each dataset was three.

Among all the predictor variables, the local events variables in *EventPerSec* is more essential than the test environment variables. Having a constant value for the local events variables would be inadequate to characterize the whole time series data and its evolution. However, the test environment variables do not necessarily require a switching mechanism. This is why in this study, the test environment variable was possible to not have a switching effect.

Another thing worth mentioning is that when modeling the Markov switching model, it is better to try estimating a simpler model first. The size of the model grows exponentially according to the number of states k, and the number of predictor variables n. In addition, if all coefficients have switching effects, the model will have more parameters to be estimated. The function in the package will try to fit the model and estimate the value of the coefficients, but there is no guarantee that the obtained results of the likelihood will be a global maximum. As a consequence, the estimated values of the coefficients cannot be completely trusted. Perlin (2015) suggested any model with $k > 3$ and $n \geq 4$ is not recommended. This is also the main reason why not all the local events were included, and the model was not made to have all the switching coefficients.

As mentioned previously in sec. 4.1, a variable called *DuProdName* in the dataset of the software release A was excluded from the regression model as it caused a singularity problem. This problem could be reduced by increasing the size of the data. However, with a limited available data, fifty-seven observations in this dataset, the variable *DuProdName* has to be dropped from the regression model before doing a further analysis.

Note that in this study, the unfiltered data has been used for the analysis. This data

contains all the QA capacity test case types which are executed in the system. Each test case type in the data established different value in its local events (e.g., one test case type has *RrcConnectionSetupComplete* value very high while another type has value of this local events very low). In data preprocressing step, the minimum value of the CPU utilization of each software package is selected, regardless the type of test case. As a consequence, many type of test cases are presented in the final datasets. This leads to the sudden drop in the CPU utilization as can be seen in the plot of each software release.

For each software release, each state in the model has a significantly high r-squared value which is greater than 0.9 (see Appendix B). R-squared value is an intuitive measurement to determine how well the model fits with the data. In general, the higher r-squared value is more preferable. Nevertheless, high r-squared value does not necessarily indicate that the model is a good model because the model with high r-squared value could be overfitting due to too many predictor variables were used in the model. Using only r-squared value to determine adequacy of the model can be misleading. A residual analysis should be considered when evaluating a model.

A Q-Q plot is a visual check for an assumption of normality. The plot provides a rough idea whether the assumption is plausible or not. From the residual analysis in sec. 4.3, it revealed that the residuals of all three datasets did not completely follow the normal distribution. This is not surprising as real data rarely have a well-defined structure. Since the Markov switching model assumed normal distributed residuals, the chosen models might have less credible results. In addition, the dataset used in this thesis is not sufficiently large. As a result, the Q-Q plot is often unclear and difficult to spot its basic feature due to more noises. ACF and PACF plots are commonly used to check a randomness in a data and also to identify a possible structure of a time series data. The models of all three datasets seemed to capture the dependent structure of the error terms in the data, even though a small amount of autocorrelation were left in the residuals. Moreover, these plots indicate that an AR(1) is already justified for these datasets.

More importantly, the results of the Markov switching model could be affected by various types of bias. Due to a small size of the training data, the training model will not be very effective and unable to capture the behaviors of new observations that lie outside the range of the training data. This could be seen in a state prediction of the software release A (see sec. 4.6). Secondly, only the predictor variables that have been analyzed to have significant impacts on the CPU utilization were included in the model. However, many other variables that is possible to have minor impact on the CPU utilization but were overlooked. Therefore, there might be some bias by not including these variables into the analysis. Finally, the criteria that has been used to select the number of states and the number of switching coefficients in the model are rather subjective. Hence, these chosen number might not be the best for different perspectives and so could be counted as one of the bias as well.

A benefit of using a non-parametric analysis is that no assumption of the data

distribution is required, and all type of distributional change within a time series can be detected. In general, the non-parametric analysis performs well when applying to the data of no known distributional form (Sharkey and Killick, 2014). However, the E-divisive method, based on the non-parametric approach, was proved to have less power in detecting changes in the data in this analysis. The main reason that the E-divisive method is not as powerful as the Markov switching model is because the Markov switching model included variables that have influences on the CPU utilization into the analysis but the E-divisive method only considered the values of the CPU utilization.

One behavior found from the Figure 4.13 and Figure 4.14 is that if a pattern of a state in the data is not obvious, the E-divisive method will be unable to detect locations of the change points. For instance, the method could not identify any changes in the first hundred observations, and also at the end of the time series in the simulated Dataset 1. This is more clear when examining the results in the simulated Dataset 2 where the E-divisive method could only detect two change point locations. If the response variable value fluctuates but the average value does not change drastically, the E-divisive method will not be able to identify the state change.

Despite some false alarms and missed detections as seen when the methods were tested with the simulated datasets, the detections are prone to be accurate when both the Markov switching model and the E-divisive method indicate the change at the exactly same location. When applying the methods to the real data, it could be concluded that the state change actually happens if both methods locate the change at the exact location. However, the location of the occurrence is possible to be slightly missed located. Furthermore, if both methods could detect the change at the close but not exact location, there are small chances that the detections could be false alarms.

In sec. 4.7, the evaluation for each model was performed. The Markov switching model was able to predict the state for the test set of the simulated Dataset 1 considerably well. The estimated coefficients in each state were similar to the coefficients in the actual models as well, except for an intercept of State2. State2 was considered to be a *Normal* state. The deviation between the actual and estimated coefficient of the intercept in this state is probably a reason why the model made four out of hundred wrong state predictions (i.e., the model indicated observations from the *Bad* and *Good* state as the *Normal* state). For the simulated Dataset 2, most of the estimated coefficients from the model were not close to the actual coefficients, especially the values of intercept in every state. State2, a *Good* state, had a totally different value between the estimated and actual coefficients. Therefore, the performance of the model was the worst when predicting observations that came from the *Good* state. However, the model was able to predict observations from a *Bad* state perfectly. The overall accuracy of the model was still rather high despite numerous switches between states in the data. This indicated that the method performed well in identifying the switches between states.

From the results obtained in this section, it can also be implied that a state prediction function, an additional implemented function in the package, is able to work properly.

6. Conclusions

This thesis assesses an ability of detecting any changes in the performance of the Ericsson's software products by applying the Markov switching autoregressive model and the E-divisive method. The simulated datasets with known states were used to make a comparison between both methods. The results from applying the Markov switching model to the real data were presented with interpretations and discussions.

For the Markov switching model, the number of states and the number of switching coefficients in the model were determined and chosen by examining the BIC, along with model output and plot. The findings from the simulated datasets revealed that the Markov switching model were able to discover switches between states rather well, despite some false alarms and missed detections. This method works well even though numerous switches between states were occurred in the data. The E-divisive method is less powerful compare to the Markov switching model. The method could identify fewer change point locations, and failed to detect many changes that were occurred in the simulated datasets. The E-divisive method will perform better and will be more efficient if the data have an obvious pattern of shifting in the time series. Based on the results from the simulated datasets and the real data, the Markov switching model was considered to be the representative method for the analysis. The E-divisive method was rather used as a guideline for any changes that could happen in the data. The method could also be used together with the Markov switching model for a confirmation of the changes in the data when the actual state is unknown. After applying the Markov switching model to both simulated datasets, the accuracy of the test sets implied that an implementation of a state prediction function appears to functionally work well.

Evaluating the obtained results is rather difficult and complicated, mostly due to a lack of annotations or label of the state of the CPU utilization. This is a common situation to an unsupervised learning problem where the ground truths are not often available. To conclude, this work has provided knowledge to understand more about the properties of the state of the CPU utilization which will, in turn, pave the way for further analysis.

6.1. Future work

The Markov switching model gave quite a promising result but several improvements could also be done to increase a robustness of the analysis. For future work, more

extensive data is recommended. Obtained results will be more reliable as additional information will decrease an uncertainty in the data.

Furthermore, there are still two more performance metrics in QA Capacity area which have not been taken into account in the thesis (e.g., a memory usage and a latency). The analysis could also be extended to analyze these metrics in a further work.

Another future extension to get the best state prediction is to consider applying the Markov switching autoregressive model on the different QA Capacity test case types i.e., one model for each type of test case. Results are expected to be more accurate as value of the CPU utilization will be slightly more stable and not so fluctuate.

It would also be interesting to use normalize value by introducing weight parameters instead of using the real value of the local events in the *EventsPerSec* and the CPU utilization. Different weights can be given to the local events depending on how much effects it has on the CPU utilization. These values are sum up and then divided by the CPU utilization value to get a final value that represents a value with respect to a specific type of test case and the CPU utilization.

Finally, in the future if some test cases have been labeled by an area expert, a semi-supervised learning algorithm, a technique that falls between a supervised learning and an unsupervised learning, could also be implemented. Training a model with a large amount of unlabeled data and a small amount of labeled data could considerably improving an accuracy of the model. The semi-supervised learning could be of good practical use, especially to an application where labeling all the data is very expensive and time-consuming.

A. Implementation in *R*

A.1. EventsPerSec

The *EventsPerSec* variable contains several local events in the test case which are separated by a tab character. A function to split tab-separated values in the field into columns is described by the following steps.

Algorithm A.1 Extract local events in EventsPerSec

```
1: for each test case do
2:     Split the vector of local events with its value into elements
3:     for each element do
4:         Split the element into the local event and the value
5:         if the column of the local event exists in the data then
6:             Insert value
7:         else
8:             Create new column for this local event and insert its value
9:         end if
10:    end for
11: end for
12: Replace missing value with zero
```

A.2. MSwM Package

A *MSwM* package in R (Josep A. Sanchez-Espigares, 2014) is mainly used to perform an univariate autoregressive Markov switching model for linear and generalized models. The package implements an Expectation-Maximization (E-M) algorithm to fit the Markov switching model.

Code was further implemented, and some modifications were also made in the function to handle warnings and errors produced when fitting the model. These modifications are described in more details here.

Non-switching effect When setting variance to have a non-switching effect, a function generates a warning. This issue is caused by a minor mistake in the code.

Non-invertible Hessian The package uses a Hessian, a matrix of second order partial derivatives with respect to parameters, for a numerical optimization. In some cases, the Hessian matrix will not be invertible as the matrix is singular. Consequently, the function cannot compute the standard error of estimated parameters. It is assumed that the singularity is coming from numerics issue i.e., the matrix is not singular but computationally it is. This non-invertible Hessian is solved by using generalized inverse (or pseudoinverse) procedure (Gill and King, 2004).

Plots Two more plots for visualizing the results have been added to the package. Name of functions and descriptions are provided below.

- `plotSmo(object)`

  Plot of smoothed probabilities with different regimes

  Description: Creates a plot that shows the smoothed probabilities for all the regimes.

- `plotArea(object)`

  Plot of response variable with regimes specifications

  Description: Create a plot that shows the periods where the variable is in a specific regime.

Categorical variable The package does not appear to work well with categorical predictor variables. Hence, a further implementation in the code for handling with categorical variables is done. *R* refers to categorical variables as factors. Most of the time, an order of the factor does not matter in the study. However, sometimes the order of the factor is of interest in the analysis and needs to be specified for the purpose of comparison or obtaining a meaningful interpretation.

NA coefficients When performing the Markov switching model, a function in the package first randomly divides the data into different subsets, and then separately fit the model to each subset to get initial coefficients in each state. These coefficients are later used for further analysis e.g., computing condition means, conditional residuals of the model, and likelihood of the parameters for each state.

It is worth noting that the function computes conditional means for each state by using matrix multiplication $\hat{y} = X\hat{\beta}$. Therefore, if NAs exist in the coefficient matrix $\hat{\beta}$, these conditional means $\hat{y}$ will become NAs, and so does conditional residuals and likelihood of the parameters.

The main reason that generated this NA coefficient issue is due to the randomness in partitioning the data. The issue frequently occurs with a categorical variable because each categorical variable has its own number of levels. The following situations listed below are what generated an error and NA coefficient:

- A variable in a subset contains the same value in all observations. Then, NA coefficient is generated for this specific variable because of singularity.
- Containing all levels of the variable in a subset is rather difficult, especially if the variable consists of several number of levels or many states are specified. For this reason, the missing level of the variable in any subset will have an NA coefficient.
- A categorical variable in a subset contains only one factor level which causes an error in fitting a regression model.

One approach to resolve this issue is by removing the process of partitioning the data in the beginning, and fitting the model with the whole data to get the initial coefficients instead. It proved that dividing data into subsets in the original function is only a method to get the starting values of coefficients in each state.

State prediction function An implementation for the state prediction is added in the package. This function predicts the most probable state that a new observation will belong to. The input of the state prediction function is the training model and a new set of the data. The function computes filtered probabilities or Equation 3.5. The output from the function is the most likely state of the new observation. The function is defined as

```
statePredict(object, newdata)
```

Algorithm A.2 State prediction function

1: **for** each observation in the test set **do**
2: Compute the probabilities of each state given that the available observation is up to time T
$$P(S_{T+1} = j|\Omega_T;\theta) = \sum_{i=1}^{k} p_{ij} P(S_T = i|\Omega_T;\theta)$$
3: Compute the conditional densities of y_{T+1}
$$f(y_{T+1}|\Omega_T;\theta) = \sum_{j=1}^{k} f(y_{T+1}|S_{T+1} = j, \Omega_T;\theta) P(S_{T+1} = j|\Omega_T;\theta)$$
4: Compute the filter probabilites of each state
$$P(S_{T+1} = j|\Omega_{T+1};\theta) = \frac{f(y_{T+1}|S_{T+1}=j,\Omega_T;\theta)P(S_{T+1}=j|\Omega_T;\theta)}{f(y_{T+1}|\Omega_T;\theta)}$$
5: Find a state which has the maximum value in the filter probabilites
6: **end for**

B. Output

Outputs from applying Markov switching autoregressive models with each dataset are shown here.

Table B.1.: Outputs from applying the selected model (Model 1) with the software release A. The table shows estimated coefficients, residual standard error, and r-squared for each state. A switching coefficient is followed by (S), and a significant coefficient is highlighted in bold.

Estimated coefficient	State 1	State 2	State 3
(Intercept)(S)	**-46.6802**	**24.0909**	**38.1426**
RrcConnectionSetupComplete(S)	**1.3429**	**0.9821**	**0.9273**
Paging(S)	**0.4105**	**0.0198**	**0.0063**
X2HandoverRequest(S)	**-55.5901**	**-44.3637**	**-44.1603**
Fdd.TddTDD	**-2045.5990**	**-2045.5990**	**-2045.5990**
NumCells6	**14.8933**	**14.8933**	**14.8933**
NumCells9	**6.5972**	**6.5972**	**6.5972**
TotCpu_1(S)	0.0143	0.0033	**-0.0033**
Residual standard error	5.2028	1.4562	0.2018
r^2	0.9970	0.9995	1.0000

Table B.2.: Outputs from applying the selected model (Model 4) with the software release B. The table shows estimated coefficients, residual standard error, and r-squared for each state. A switching coefficient is followed by (S), and a significant coefficient is highlighted in bold.

Estimated coefficient	State 1	State 2	State 3
(Intercept)(S)	**52.1714**	**10.0589**	**33.7823**
RrcConnectionSetupComplete(S)	**1.0370**	**1.2122**	**1.0199**
Paging(S)	**0.2988**	**0.0460**	**0.0204**
X2HandoverRequest(S)	0.2073	**0.5048**	**0.5462**
DuProdName	**-22.6507**	**-22.6507**	**-22.6507**
Fdd.TddTDD(S)	-19.2896	**-36.5123**	**-13.1966**
NumCells18(S)	**-358.4238**	**-14.3739**	**21.7627**
NumCells4(S)	**-353.4862**	**61.3992**	**49.8430**
NumCells6(S)	**-354.5987**	**18.5767**	**33.9792**
NumCells9(S)	**57.2922**	**35.9999**	**57.4627**
TotCpu_1(S)	0.0241	**0.0080**	**0.0121**
Residual standard error	21.1321	1.1277	3.7431
r^2	0.9373	0.9996	0.9972

Table B.3.: Outputs from applying the selected model (Model 1) with the software release C. The table shows estimated coefficients, residual standard error, and r-squared for each state. A switching coefficient is followed by (S), and a significant coefficient is highlighted in bold.

Estimated coefficient	State 1	State 2	State 3
(Intercept)(S)	**82.1406**	**153.4261**	**22.2871**
RrcConnectionSetupComplete(S)	**1.0955**	0.0070	**1.3451**
Paging(S)	**-0.0156**	**-0.0583**	**0.0154**
X2HandoverRequest(S)	**1.0467**	**3.2512**	**0.4970**
DuProdName	**8.2626**	**8.2626**	**8.2626**
Fdd.TddTDD	0.7422	0.7422	0.7422
NumCells12	**-48.4617**	**-48.4617**	**-48.4617**
NumCells18	**19.0542**	**19.0542**	**19.0542**
NumCells9	6.1590	6.1590	6.1590
TotCpu_1(S)	**0.0146**	-0.0629	**0.0513**
Residual standard error	3.8316	14.8584	5.6141
r^2	0.9973	0.9138	0.9927

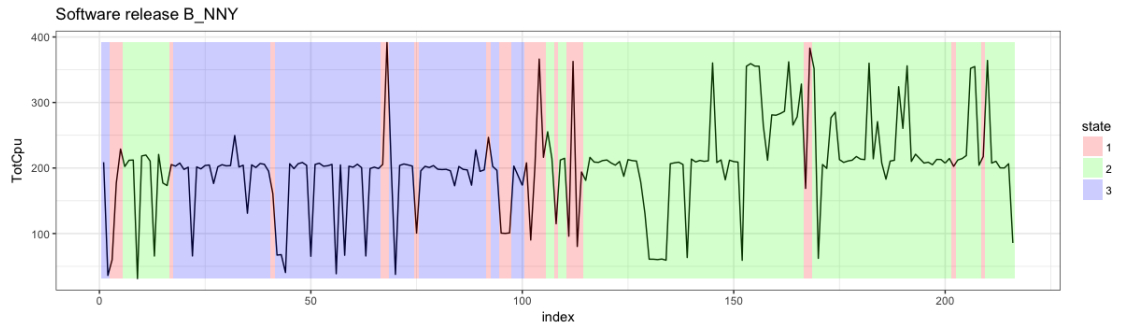


Figure B.1.: The CPU utilization of the software release B showing the periods where the observation is in the specific state.
Model 2: DuProdName and Fdd/Tdd are non-switching coefficients.

Bibliography

Ailliot, P. and Monbet, V. (2012). Markov-switching autoregressive models for wind time series. *Environmental Modelling & Software*, 30:92–101.

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

Beaulieu, C., Chen, J., and Sarmiento, J. L. (2012). Change-point analysis as a tool to detect abrupt climate variations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 370(1962):1228–1249.

Bolton, R. J. and Hand, D. J. (2002). Statistical fraud detection: A review. *Statistical science*, pages 235–249.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cryer, J. D. and Kellet, N. (1986). *Time series analysis*, volume 101. Springer.

Dahlman, E., Parkvall, S., and Skold, J. (2013). *4G: LTE/LTE-advanced for mobile broadband*. Academic press.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Gill, J. and King, G. (2004). What to do when your hessian is not invertible: Alternatives to model respecification in nonlinear estimation. *Sociological methods & research*, 33(1):54–87.

Grimmett, G. and Stirzaker, D. (2001). *Probability and random processes*. Oxford university press.

Gu, W., Choi, J., Gu, M., Simon, H., and Wu, K. (2013). Fast change point detection for electricity market analysis. In *Big Data, 2013 IEEE International Conference on*, pages 50–57. IEEE.

Hamilton, J. D. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the Econometric Society*, pages 357–384.

Hamilton, J. D. (1990). Analysis of time series subject to changes in regime. *Journal of econometrics*, 45(1-2):39–70.

Hamilton, J. D. (2005). Regime-switching models.

Hawkins, D. M. and Deng, Q. (2010). A nonparametric change-point control chart. *Journal of Quality Technology*, 42(2):165.

James, N. A. and Matteson, D. S. (2013). ecp: An r package for nonparametric multiple change point analysis of multivariate data. *arXiv preprint arXiv:1309.3295*.

Janczura, J. and Weron, R. (2012). Efficient estimation of markov regime-switching models: An application to electricity spot prices. *AStA Advances in Statistical Analysis*, 96(3):385–407.

Josep A. Sanchez-Espigares, A. L.-M. (2014). *MSwM: Fitting Markov Switching Models*. R package version 1.2.

Juang, B.-H. and Rabiner, L. R. (1990). The segmental k-means algorithm for estimating parameters of hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9):1639–1641.

Kim, C.-J. (1994). Dynamic linear models with markov-switching. *Journal of Econometrics*, 60(1-2):1–22.

Kim, C.-J., Nelson, C. R., et al. (1999). State-space models with regime switching: classical and gibbs-sampling approaches with applications. *MIT Press Books*, 1.

Kim, C.-J., Nelson, C. R., and Startz, R. (1998). Testing for mean reversion in heteroskedastic data based on gibbs-sampling-augmented randomization. *Journal of Empirical finance*, 5(2):131–154.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69.

Leroux, B. G. and Puterman, M. L. (1992). Maximum-penalized-likelihood estimation for independent and markov-dependent mixture models. *Biometrics*, pages 545–558.

Luong, T. M., Perduca, V., and Nuel, G. (2012). Hidden markov model applications in change-point analysis. *arXiv preprint arXiv:1212.1778*.

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

Matteson, D. S. and James, N. A. (2014). A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505):334–345.

Nousiainen, S., Kilpi, J., Silvonen, P., and Hiirsalmi, M. (2009). Anomaly detection from server log data.

Page, E. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

Patcha, A. and Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470.

Perlin, M. (2015). Ms_regress-the matlab package for markov regime switching models.

Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Reeves, J., Chen, J., Wang, X. L., Lund, R., and Lu, Q. Q. (2007). A review and comparison of changepoint detection techniques for climate data. *Journal of Applied Meteorology and Climatology*, 46(6):900–915.

Rydén, T. et al. (2008). Em versus markov chain monte carlo for estimation of hidden markov models: A computational perspective. *Bayesian Analysis*, 3(4):659–688.

Sarasamma, S. T., Zhu, Q. A., and Huff, J. (2005). Hierarchical kohonenen net for anomaly detection in network security. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(2):302–312.

Schwarz, G. et al. (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464.

Shannon, M. and Byrne, W. (2009). A formulation of the autoregressive hmm for speech synthesis.

Sharkey, P. and Killick, R. (2014). Nonparametric methods for online changepoint detection. *Ýëåêòðîííûé ðåñóðñ.–[Ðåæèì äîñòóïà]: http://www. lancaster. ac. uk/pg/sharkeyp/PaulRT2. pdf–23 p. Reference*, 1:3.

Stanke, M. and Waack, S. (2003). Gene prediction with a hidden markov model and a new intron submodel. *Bioinformatics*, 19(suppl 2):ii215–ii225.

Staudacher, M., Telser, S., Amann, A., Hinterhuber, H., and Ritsch-Marte, M. (2005). A new method for change-point detection developed for on-line analysis of the heart beat variability during sleep. *Physica A: Statistical Mechanics and its Applications*, 349(3):582–596.

Sung, A. H. and Mukkamala, S. (2003). Identifying important features for intrusion detection using support vector machines and neural networks. In *Applications and the Internet, 2003. Proceedings. 2003 Symposium on*, pages 209–216. IEEE.

Valdes, A. and Skinner, K. (2000). Adaptive, model-based monitoring for cyber attack detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 80–93. Springer.

Weskamp, P. and Hochstotter, M. (2010). Change point analysis and regime switching models. Technical report, Working paper. https://statistik. econ. kit. edu/download/Change_Point_Analysis. pdf.