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

This is the summary of the thesis.

Keywords: Markov switching model

Acknowledgments

My warmest appreciation goes to the following people:

- Professor Krzysztof Bartoszek for his constant wise guidance and support in every stage of the thesis project.
- Ericsson for giving me an opportunity to work with them and providing the data for this thesis. In particular, Armin Catovic and Jonas Eriksson for defining and interesting problem as well as advising me in various matters.
- Linköping university for granting me a scholarship and giving me a chance to be a part of Master's program in Statistics and Data Mining.
- My family, friends and boyfriend for their continuous support and encouragement.

1. Introduction

1.1. Background

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). It has a bottom line for discovering the time point where the changes in time series 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 when the point in time series takes place. Another important term which will be used in this thesis 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 over decades as it is a problem of interest in many applications which the characteristic of data is collected over time. The change should be flagged as soon as it occurs in order to be properly dealt with such changes in time and reduced 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 is detected. This method gradually becomes 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 exceed the tolerant undetected, it could lead to the lost 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.

For this analysis, change point analysis will be used to identify changes in performance of Ericsson's software products. Many test cases have been 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.

Recently, a method called hidden Markov model or Markov switching model is widely used for discovering a change point in time series. Both terms are acceptable but being called differently regarding to the different fields of study. Markov switching model uses a concept of 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 the prediction of gene is being made. Markov switching model has been extensively applied in the field of economics and finance and has a large literature. For example, business cycle 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 mean that constant parameter in 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.

In this study, each software package in the 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 value is 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, with the first order autoregression, is taken into consideration when modeling the Markov switching model and becomes 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 that there is no assumption made about the distribution. It can, therefore, apply to a wider range of application and capture various kinds of changes (Sharkey and Killick, 2014). In this study, the non-parametric analysis using hierarchical estimation techniques is also implemented for identifying the change of location in the time series.

1.2. Objective

The core idea of this thesis is to reduce workload of manual inspection when the performance analysis of an update software package is required. With an increase 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. There is also worth to mention that the performance of a particular software package can vary on different test environment. The implemented algorithm should also be able to detect when the test environment is altered. This thesis only focuses on a CPU utilization which is one of the three essential factors for evaluating the performance of the upgrade 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.

2. Data

2.1. Data sources

Data used in this thesis is provided by Ericsson site in Linköping, Sweden. Ericsson[1] founded by Lars Magnus Ericsson since 1876 is one of the world's leading in 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.

LTE, widely known as 4G, is the Long-Term Evolution of the 3G network. 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 network. It handles the radio communication between the User Equipment, UE, or mobile device and the evolved base stations called eNB. Each eNB is a base station which controls and manages radio communications with multiple devices in one or more cells. Several base stations are connected to a Mobility Management Entity, MME, which handles the high-level operation of the mobile for different UEs. 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 connect 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.

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 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 code is committed to this track continuously. In order to create a structure for this contribution, a daily *software package* is built. Every day a new software package is built which can be seen as a snapshot or a marker in

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

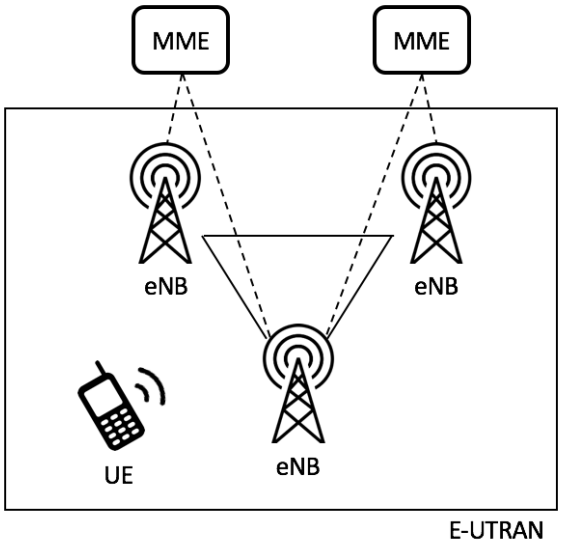

Figure 2.1.: LTE architecture overview

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* 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 the snapshot in the code integration timeline. Figure 2.2 presents the relation between software release and software package.

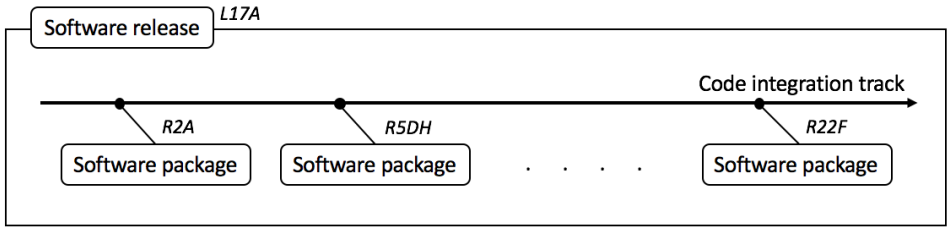

Figure 2.2.: Example of one software release that begins a code integration track. Several software packages are launched in the timeline.

There are thousands of automated tests performed. Each test belongs to a particular suite of tests, which belongs to a particular quality assurance (QA) area. For this thesis framework, one suite of test cases that test the signaling capacity within QA Capacity is focused. The QA Capacity has the responsibility of testing and tracking test cases related to LTE node capacity. Each one of these test cases has a well-

defined traffic model 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 LTE node. The LTE node then increments one or more counters for each one of these procedures or stimuli it detects. These counters are called local events and they are represented by EventsPerSec.

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

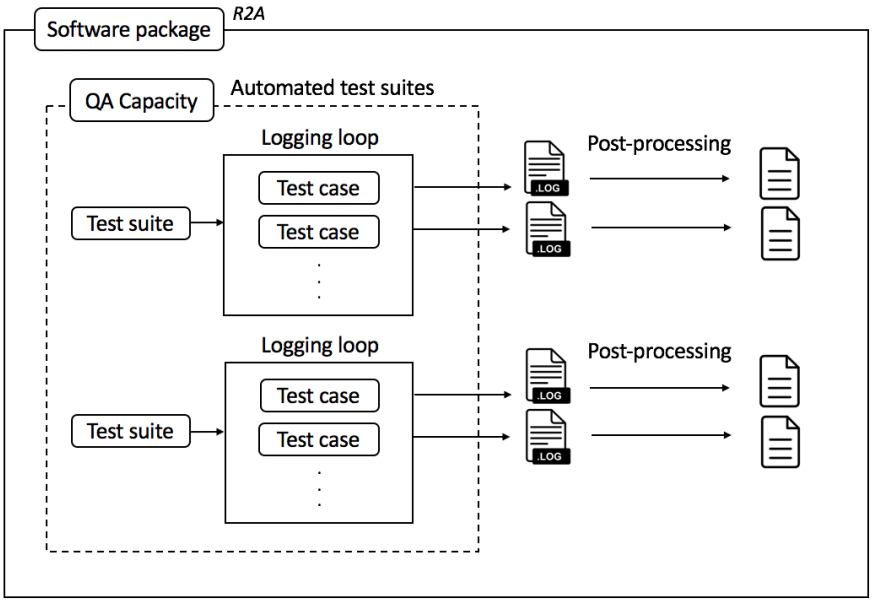

Figure 2.3.: Example of one software package. First, the QA Capacity automated test suites is started. For each test suite, a logging loop is started and a log is produced for each test case. The log file is fed to post-processing tools and the data output is obtained.

2.2. Data description

This is the data for the second generation (G2) product which contain 2,781 test cases. The data is collected on 20 January 2017 and is extracted from log file

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. 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 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 Antenna
- 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%: Performance of a test case in terms of CPU utilization
- PerCpu%
- PerThread%
- EventsPerSec: Event intensity

  This variable contains several local events that can be used when defining the test case. Apparently, there is no fixed number of local events in this variable as different test cases involve additional procedure compared to other test cases. 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 data consists of three software releases – L16A, L16B and L17A – and it is split into a dataset according to the software release. The dataset is sorted by the version of the software package which is named alphabetically. Even though the dataset has a timestamp when the test case is run, the timestamp is unsuitable to represent itself as a time point in time series. The software package is an important variable and more of an interest to this analysis. Therefore, it is viewed as the data points indexed in time order in the time series.

Test cases are not always being executed properly. It is either no traffic is generated during the test case or that data is not logged. If the EventsPerSec field is missing or has no value in it, the test case is being treated as incomplete and all the data related to that particular test case is ignored. This also applied for other cases when any fields in the dataset is missing.

In Table 2.1, it can be seen that EventsPerSec variable stores multiple values separated by a tab character. These tab-separated values in the field are split to columns. The process is done in order to get its local events and values which characterized the test case. These are later on used as predictor variables for fitting the Markov switching model.

Each software release consists of several software packages. Also, numerous test cases are executed in one software package. Since the software package acts as a point in time in the time series, it appears to be rather difficult to visualize all results from every executed test case for each software package. Hence, the test case which has the lowest value of the CPU utilization (or minimum value of TotCpu%) is selected to represent the performance of a specific software package. Although

taking an average of the 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. It is, therefore, settled to remain the data as it is and always visualize the data exactly as it is recorded. Another important reason for not using the average value of the CPU utilization is because the essential information in the test case will be lost. Each test case has its own local events in EventsPerSec field that used for identifying the test case. The details of these local events are absent when averaging over the CPU utilization of the test cases in the software package.

After performing all the steps describing above, the dataset of software release L16A, L16B and L17A consist of 64, 241, and 144 test cases, respectively. Lastly, each dataset with particular software release is divided into two subsets. Ninety percent of the dataset is used for training the model and the remaining ten percent is left out for testing.

Some variables in the data are chosen to be used for further analysis. There are in total one response variable and six predictor variables. Table 2.2 shows the name of variables and their description. 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 contribution to the CPU utilization.

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

In this chapter, methods used for performing the change point analysis in this thesis are explained. It first starts by providing general details about Markov chains. Later on, the simple Markov switching model feature and more general model specifications namely Markov switching autoregressive model are discussed. Next three sections are devoted to some 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, a simulation technique is explained.

3.1. Markov chains

A Markov chain is a random process which has a property that is given on 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 of T, where $T = \{0, 1, 2, ...\}$ is called a discrete-time process, and $T = [0, \infty)$ is called a continuous-time process. Let X_t be a sequence which has 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 3.1.1. (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 being 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. This evolution of changing from $X_t = i$ to $X_t = j$ is defined by the transition probability as $P(X_t = j|X_{t-1} = i)$. Markov chains are frequently assumed that these probabilities depend only on i and j and do not depend on t.

Definition 3.1.2. (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 stochastic 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 column equal to one or* $\sum_i p_{ij} = 1$ *for all* j

Definition 3.1.3. (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 positive recurrent or non-null persistent if μ_i is finite. Otherwise, the state i is null persistent.

Definition 3.1.4. (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 3.1.5. (Grimmett and Stirzaker, 2001, p.222) A state is called ergodic if it is non-null persistent and aperiodic.

Definition 3.1.6. A chain is called irreducible if it is possible to go from every state to every other states.

Definition 3.1.7. If there is a finite state space, an irreducible Markov chain is the same thing as ergodic Markov chain.

3.2. 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 assumes 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 imply a conditional dependency.

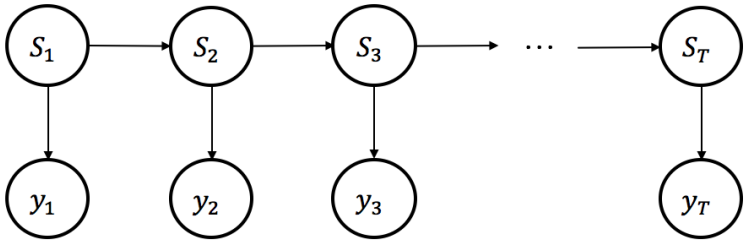

Figure 3.1.: Model structure

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 are predictor variables of the time series at time t

β_{S_t} are coefficients in state S_t, where $S_t = i$, $1 \leqslant i \leqslant k$

ε_t follows a Normal distribution with zero mean and variance given by $\sigma^2_{S_t}$

The Equation 3.1 is the simplest form for the switching model where there are $k-1$ structural breaks in the model parameters. 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}. Since

the whole process S_t is unobserved, the initial state where $t = 0$ of each state 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).

3.3. Autoregressive (AR) model

An autoregressive model is one type of time series models that is used to describe the 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 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.

3.4. 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).

Definition 3.4.1. 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 3.2. It can be clearly seen that there is a dependency at the observation level and the observation is not independent from one another.

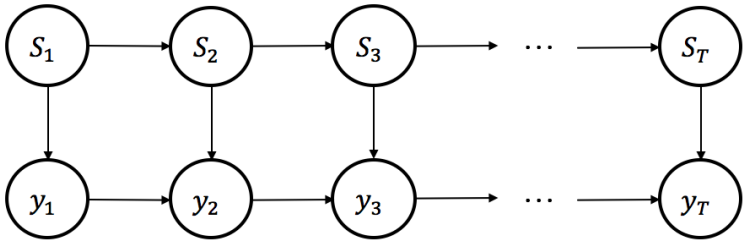

Figure 3.2.: Model structure of Markov switching AR(1)

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 $\theta = \{\beta_1, \beta_2, \phi_{1,1}, \phi_{1,2}, \sigma_1^2, \sigma_2^2, \pi_1, \pi_2, p_{11}, p_{22}\}$.

3.5. 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, E-M algorithm is used in estimating parameters and its general procedure is briefly described below.

3.5.1. The Expectation-Maximization algorithm

E-M algorithm is originally designed to deal with 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 values. The set of parameters θ is estimated by 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 (Janczura and Weron, 2012).

3.5.1.1. E-step

In this step, $\theta^{(n)}$ is the derived set of parameters in M-step from the previous iteration, and 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 $P(S_t = j|\Omega_T; \theta)$. 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 that the available observation is up to time $t - 1$, is calculated by

$$P(S_t = j|\Omega_{t-1}; \theta^{(n)}) = \sum_{i=1}^{k} p_{ij}^{(n)} P(S_{t-1} = i|\Omega_{t-1}; \theta^{(n)}) \tag{3.2}$$

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

$$f(y_t|\Omega_{t-1}; \theta^{(n)}) = \sum_{j=1}^{k} f(y_t|S_t = j, \Omega_{t-1}; \theta^{(n)}) P(S_t = j|\Omega_{t-1}; \theta^{(n)}) \tag{3.3}$$

where $f(y_t|S_t = j, \Omega_{t-1}; \theta) = \frac{1}{\sqrt{2\pi\sigma_{S_t}^2}} exp\left\{-\frac{(y_t-\beta_{S_t})^2}{2\sigma_{S_t}^2}\right\}$ 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

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

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

$$\begin{aligned} P(S_t = j|S_{t+1} = i,\Omega_T;\theta^{(n)}) &\approx P(S_t = j|S_{t+1} = i,\Omega_t;\theta^{(n)}) \\ &= \frac{P(S_t = j, S_{t+1} = i|\Omega_t;\theta^{(n)})}{P(S_{t+1} = i|\Omega_t;\theta^{(n)})} \\ &= \frac{P(S_t = j|\Omega_t;\theta^{(n)})p_{ij}^{(n)}}{P(S_{t+1} = i|\Omega_t;\theta^{(n)})} \end{aligned} \tag{3.5}$$

and

$$P(S_t = j|\Omega_T;\theta^{(n)}) = \sum_{i=1}^{k} P(S_t = j, S_{t+1} = i|\Omega_T;\theta^{(n)}) \tag{3.6}$$

then, the smoothed probabilities can be expressed as

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

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

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

(3.8)

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

3.5.1.2. M-step

The new estimated model parameters $\theta^{(n+1)}$ is obtained by finding a set of parameters that maximizes the Equation 3.8. This new set of parameters is more precise 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.

3.6. State prediction

A function to predict the most probable state for the new observation is implemented in this analysis (see Appendix B).

The probability of being in state j at time $T+1$ on the basis of current information is 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 the Equation 3.4 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.

3.7. Model selection

Model selection is a task of selecting the most suitable model for a given set of data based on the quality of the model. In this thesis framework, 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. It is also known as Schwarz Information Criterion (Schwarz et al., 1978).

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

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. One benefit from using BIC is that this criterion heavily penalizes model complexity as it takes into account the number of parameters in the model. BIC attempts to reduce the risk of over-fitting.

3.8. Non-parametric analysis

The parametric test statistics outperform the non-parametric test if data belongs to some known distribution families. However, the parametric test is not properly performing well in detecting change point for an unknown underlying distribution (Sharkey and Killick, 2014). Applying the non-parametric analysis to the real-world process gives a real advantage to the analysis since data collected from application, in general, does not always have a well-defined structure and prefer such analysis that is not too restricted (Hawkins and Deng, 2010). For this reason, the non-parametric analysis is implemented in order to get a heuristic idea of the change point location in this thesis framework. The obtained result is also used for comparing with the result from the Markov switching autoregressive model.

E-divisive

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

The E-divisive method is an algorithm in the *ecp* package. This algorithm performs a divisive clustering in order to estimate the multiple 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 at each iteration and it divides the time series into different segments. The algorithm also used a permutation test to compute the statistical significance of an estimated change point. More details about the estimation is described on Matteson and James (2014).

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

3.9. Simulation

Since there is no annotation for the state of the CPU utilization in the data, an accuracy can not be computed after making a state prediction. One possible solution to test and verify how well the implemented predict function performs is to use the simulation technique. The data consists of two predictor variables and one response variable with the known state is simulated. The actual models for each state are

$$y = \begin{cases} 10 + 0.6X1_t - 0.9X2_t + 0.5Y_{t-1} + \varepsilon_t^{(1)} & \varepsilon_t^{(1)} \sim N(0,1) \\ 2 + 0.8X1_t + 0.2Y_{t-1} + \varepsilon_t^{(2)} & \varepsilon_t^{(2)} \sim N(2,0.5) \\ -12 + 0.7X1_t + 0.2Y_{t-1} + \varepsilon_t^{(3)} & \varepsilon_t^{(3)} \sim N(1,1) \end{cases}$$

The simulated data contains 500 observations which has a presence of three different states – Normal, Bad and Good. Figure 3.3 presents a plot of y over a period of time and the period where observations in the data belong to one of the state is shown in Figure 3.4.

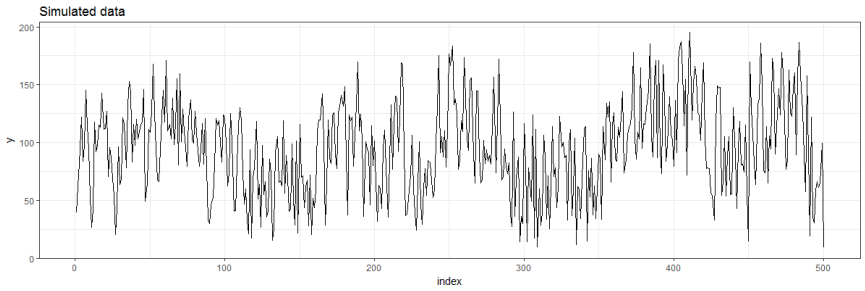

Figure 3.3.: Simulated data. The y variable is the response variable

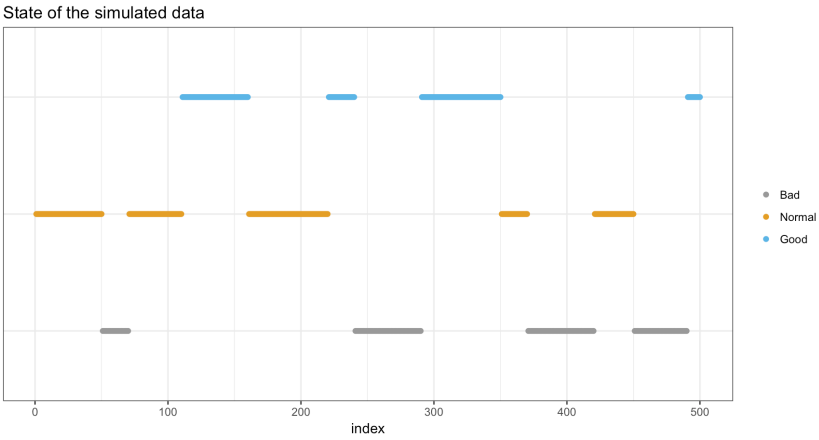

Figure 3.4.: The period in the time series when data is in each state

4. Results

The most relevant results of the analysis is shown and organized in this chapter as follow. As a first step, The number of states for the model is decided. Then, performing a model selection for the given dataset in order to find which model is the most appropriate one and which parameter should have a switching effect in the model. Residual analysis is performed in the later section to validate the models. Next, the results of the non-parametric analysis are also presented. The last two sections report the results of predicting the state for the new observation in each dataset and verification the use of predict function in simulated data.

4.1. States

To estimate the set of necessary parameters, the *MSwM* package in R is used. More details on the package can be found in Appendix B.

Markov switching autoregressive model is performed for each dataset with every parameter in the model has switching effects i.e., the coefficients can take on different values in different periods. The estimation is made under the assumptions of two or three states $S_t \in S$ where $S = 1, 2, .., k$ and $k = 2$ or 3. This comes 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). In addition, the normality assumption applies to the distribution of residuals during the estimation.

The complete linear Markov switching autoregressive model in this thesis framework is defined as

$$\begin{aligned} y_t =& \beta_{0,S_t} + \beta_{RrcConnectionSetupComplete,S_t} X_{RrcConnectionSetupComplete,t} \\ &+ \beta_{Paging,S_t} X_{Paging,t} + \beta_{X2HandoverRequest,S_t} X_{X2HandoverRequest,t} \\ &+ \beta_{DuProdName,S_t} X_{DuProdName,t} + \beta_{Fdd/Tdd,S_t} X_{Fdd/Tdd,t} \\ &+ \beta_{NumCells,S_t} X_{NumCells,t} + \phi_{1,S_t} y_{t-1} + \varepsilon_{S_t} \end{aligned} \tag{4.1}$$

Before performing the Markov switching autoregressive model, a standard linear regression model is fitted to the dataset first. It is found that one coefficient in the dataset of software release L16A is not defined because of singularity. Hence,

Table 4.1.: BICs for each dataset of software release

Software release	BIC	
	$k = 2$	$k = 3$
L16A	439.677	417.682
L16B	1,763.507	1,797.259
L17A	1,189.061	1,199.075

DuProdName variable is dropped from the Equation 4.1. The BIC values from fitting Markov switching autoregressive model are shown in Table 4.1.

For the software release L16A, the BIC value suggests that the three-state Markov switching autoregressive model gives better fit in comparison to the two-state model. Figure 4.1 presents that the Markov chain remains in State 1 for an extensive period of time before it switches to State 2. When the chain is in State 2, it stays there only a short time and then immediately moves back to State 1. There are a few switches between these two states in Figure 4.1. On the other hand, it is visible that there are more switching between states in Figure 4.2. One noticeable thing is that State 2 in the two-state model seems to be defined as State 1 in the three-state model. Moreover, State 1 which has a long duration in the two-state model is now containing a lot of switching between states in the three-state model.

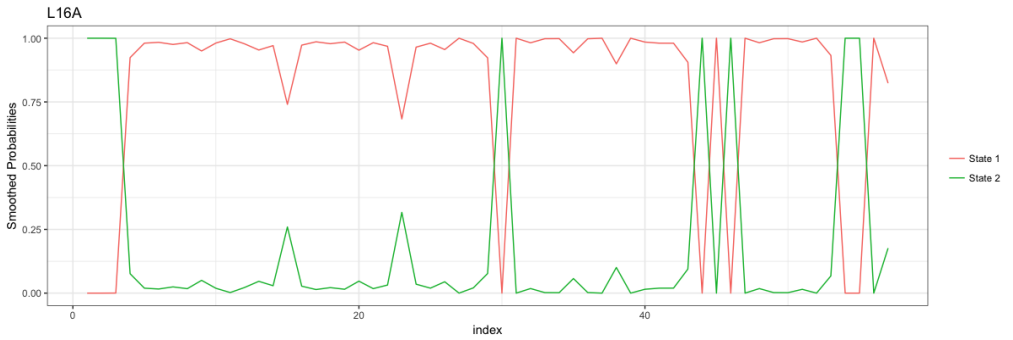

Figure 4.1.: The smoothed probabilities for software release L16A with two-state model

According to Table 4.1, the Markov switching autoregressive models with two states for the remaining two software releases, L16B and L17A, yield better results in favor of lower BICs.

In Figure 4.3, the Markov chain has several periods that switch back and forth between the two states for the software relase L16B. The duration of being in State 2 is longer compares to the duration of staying in State 1. It can be seen that the chain stays in State 2 for quite some time in the middle period (observation 91-99 and 101-114) before turning to State 1. Apparently, there are more switches between

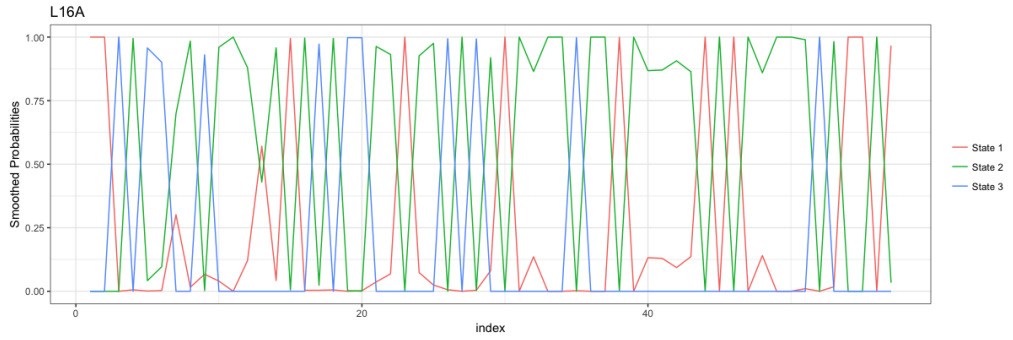

Figure 4.2.: The smoothed probabilities for software release L16A with three-state model

states in the three-state model especially in the beginning, middle and the end of the period. State 3 in Figure 4.4 appears to remain in its own state longer than any other states as can be seen throughout observation 15-39, 42-67 and 140-170.

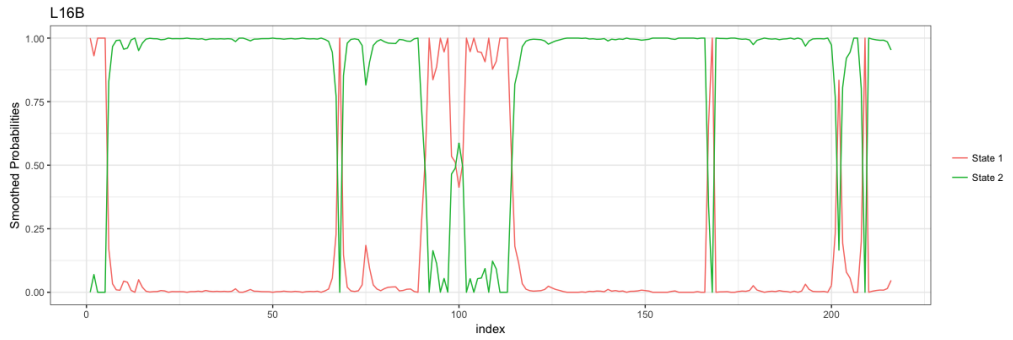

Figure 4.3.: The smoothed probabilities for software release L16B with two-state model

There are quite a lot of switches between states in the two-state model of the software release L17A. In Figure 4.5, when the Markov chain is in State 1, it stays there quite some time before leaving to State 2. It can be seen that the chain has fairly a short duration being in State 2 and quickly switch back to State 1. Figure 4.6 presents the chain which has many switches between State 1 and State 2 in the first half of the time period. There are State 2 in the beginning The chain for the three-state model stays in State 2 significantly long from observation 104 until 129, the end of the duration.

The outputs from the model along with plots provide more interpretable results when defining the model with three states in software release L16B and L17A despite the slightly higher value in BIC. In regards to this, the three-state models for each software release are further analyzed throughout the thesis.

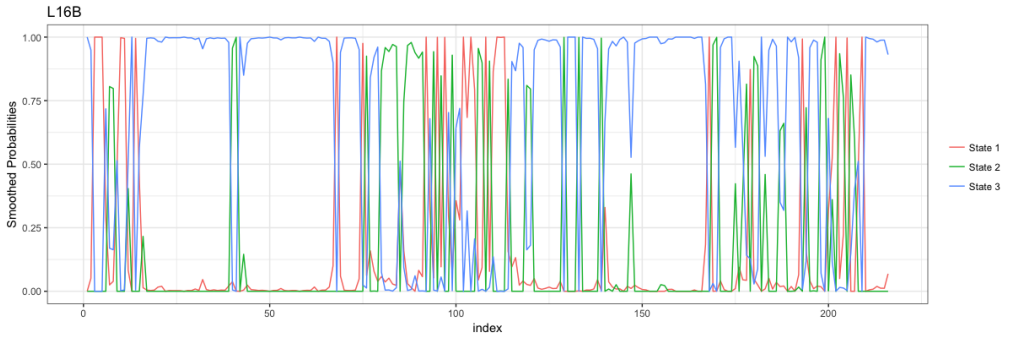

Figure 4.4.: The smoothed probabilities for software release L16B with three-state model

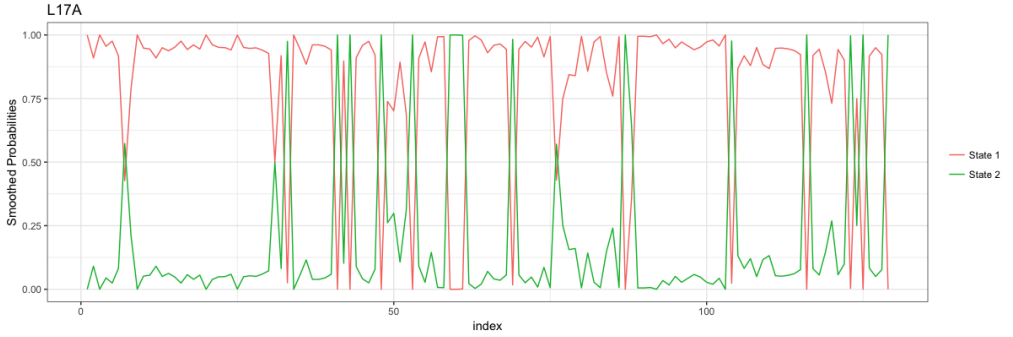

Figure 4.5.: The smoothed probabilities for software release L17A with three-state model

4.2. Switching coefficients

The fitted Markov switching autoregressive models earlier only have state-dependent parameters i.e., coefficients of the regression with switching effects. Apparently, each coefficient can have either a switching or non-switching effect. The hypothesis for this switching/non-switching coefficients is that the local events variables have switching effects while the variables considered as the test environment are possible to have non-switching effects.

The Markov switching autoregressive models are applied to each dataset. All combinations of coefficient in the regression with and without switching effect are tried out. Since DuProdName is not included in the model fitting for the dataset of software release L16A, two variables are left for trying whether it can have non-switching effect or not. The result is shown in Table 4.2.

The second model has the highest BIC value and even higher than the original model with all switching coefficients. From the obtained outputs, it is found that a linear

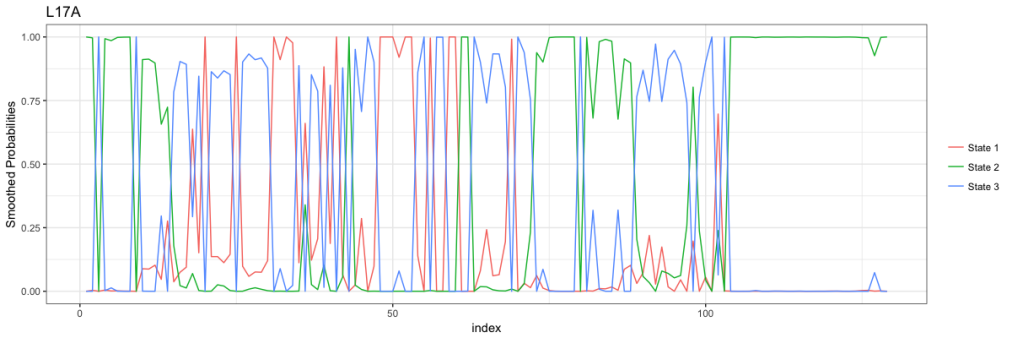

Figure 4.6.: The smoothed probabilities for software release L17A with three-state model

Table 4.2.: List of the model structure for software release L16A with different switching coefficients along with BIC

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

regression for State 3 in all three models has r-squared equals to 1 with moderately small residual standard error.

Next, Table 4.3 presents the results of fitting the model with different combinations in switching coefficients for the software release L16B. Models that have higher BIC values than the model which has switching effect in all coefficients are model 5 and 7. It is noticeable that model 2 which has DuProdName and Fdd/Tdd as coefficient with non-switching effect has a smallest value in BIC.

It can be seen in Table 4.4 that only model 2 has a higher BIC value than the model with all switching coefficients. The least BIC value is from the first model that all three parameters in the test environment have non-switching effects.

Further discussion and details about how to select the Markov switching autoregressive model for the given dataset is provided in chapter 5.

4.3. Residual analysis

The results of the selected Markov switching autoregressive model for each software release are analyzed to see how well the time series data fits the assumption of normally distributed residuals.

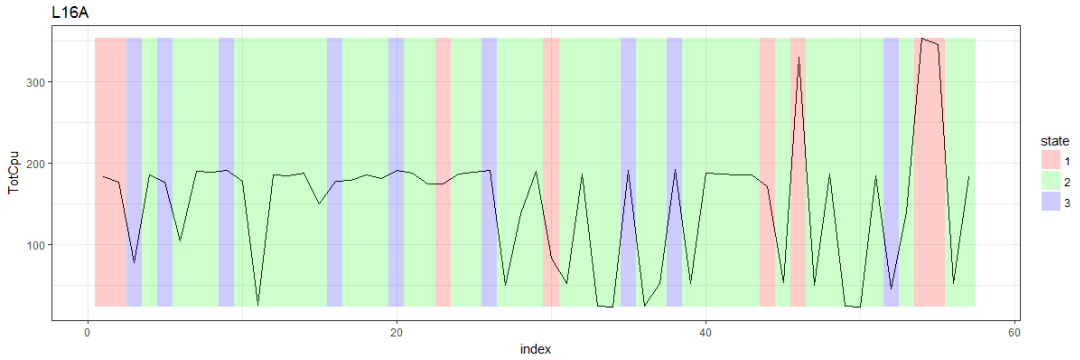

Figure 4.7.: L16A_NN

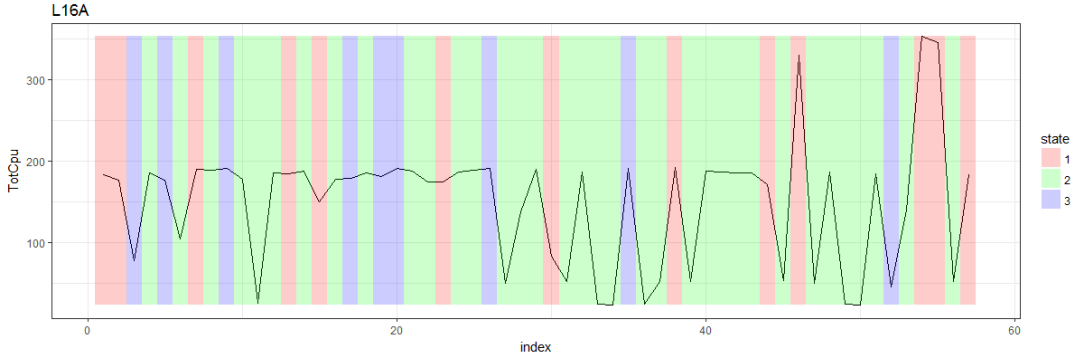

Figure 4.8.: L16A_YN

Plot of residuals against fitted values, a Normal Q-Q plot, ACF/PACF of residuals and ACF/PACF of square residuals of each model are shown below.

4.3.1. Software release L16A

df

4.3.2. Software release L16B

df

4.3.3. Software release L17A

df

Table 4.3.: List of the model structure for software release L16B with different switching coefficients along with BIC

Model	Switching effect			BIC
	DuProdName	Fdd/Tdd	NumCells	
1	N	N	N	1,787.528
2	N	N	Y	1,704.393
3	N	Y	N	1,784.384
4	N	Y	Y	1,776.102
5	Y	N	N	1,806.385
6	Y	N	Y	1,725.865
7	Y	Y	N	1,804.487

Table 4.4.: List of the model structure for software release L17A with different switching coefficients along with BIC

Model	Switching effect			BIC
	DuProdName	Fdd/Tdd	NumCells	
1	N	N	N	1,140.474
2	N	N	Y	1,204.280
3	N	Y	N	1,152.740
4	N	Y	Y	1,184.643
5	Y	N	N	1,146
6	Y	N	Y	1,189.236
7	Y	Y	N	1,157.311

4.4. Non-parametric analysis

The E-divisive procedure is applied to all three datasets. The method reports that one cluster is found for the dataset of the software release L16A. There are five clusters found in both datasets of the software release L16B and L17A. Table 4.5 shows places in the time series data where the E-divisive algorithm is able to detect the significant changes.

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

Software release	Change-point location
L16A	-
L16B	130, 135, 153, 170
L17A	9, 77, 82, 105

The CPU utilization for software release L16A, L16B and L17A along with its

estimated change points in the time series is plotted and shown in Figure 4.9, Figure 4.10 and Figure 4.11, respectively. The E-divisive method cannot identify any changes in the dataset of software release L16A.

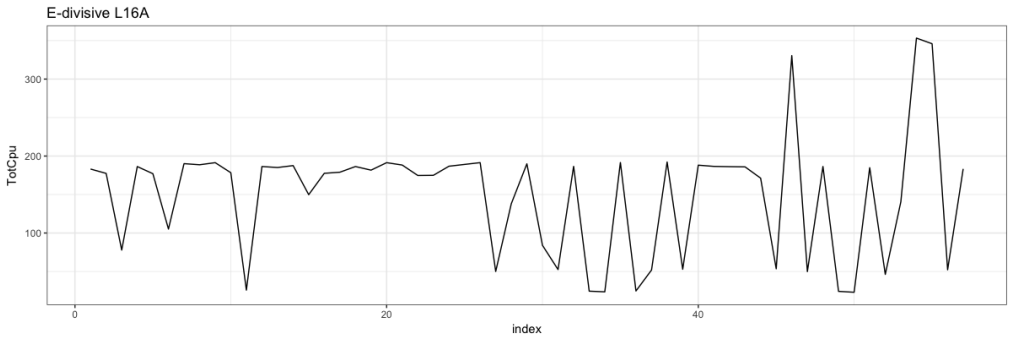

Figure 4.9.: The CPU utilization for the software release L16A

Four change points are identified from the method for the software release L16B and L17A. In The estimated change points for the dataset of software release L16B are likely to occur around the same period of time. On the contrary, the software release L17A has the estimated change points from the method rather spread out.

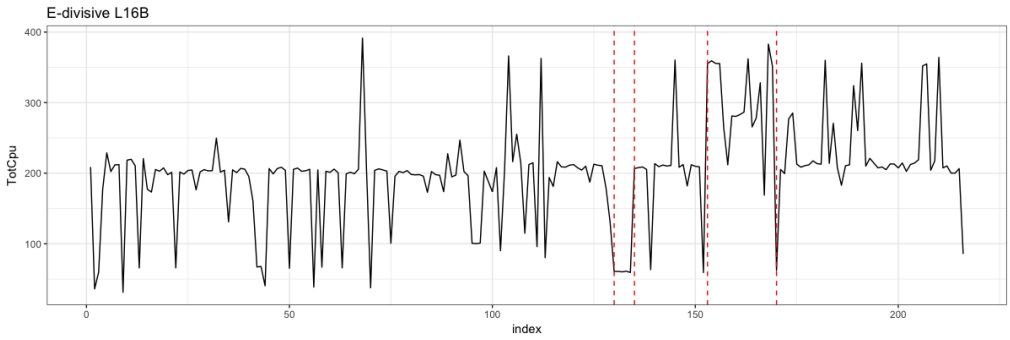

Figure 4.10.: The CPU utilization for the software release L16B. The red dashed vertical lines indicate the locations of estimated change points.

4.5. Predicting the state

After deciding which model is going to be used for the given dataset of the software release, the test set is used to find the most probable state for each new observation.

L16A

L16B

L17A

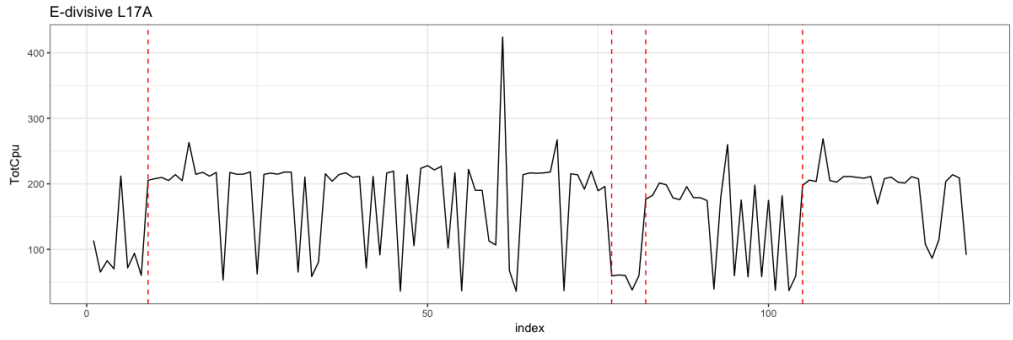

Figure 4.11.: The CPU utilization for the software release L17A. The red dashed vertical lines indicate the locations of estimated change points.

4.6. Simulation

Markov switching model is fitted with eighty percent of the observations and the remaining is used as a test set. Then, using the training model to predict the state of each observation in the test set and the result is shown in Table 4.6. There are two observations from the Bad state which are wrongly classified to the Normal state. Moreover, another two observations from the Good state are classified to the Normal state. The overall accuracy is 0.96 and the misclassification rate is 0.04.

Table 4.6.: Confusion matrix for applying the Markov switching model to fit with the test set

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

One can see that the model is able to predict the Bad and Good state correctly. The model and it has accuracy equals to 0.88.

5. Discussion

A thorough search of the relevant literature yielded that this might be the first time that Markov switching model is applied on the data rather than financial time series or signal processing. A large amount of time is spent on understanding the data and determining which methods would be the best fit to this problem.

One thing worth mentioning is that it is better to always try estimating simple models first. The model's size grows exponentially according to the number of state k, and the number of predictor variables. There would be more parameters to estimate if parameters also have switching effects. The obtained results probably be a local maximum and cannot be completely trusted (Perlin, 2015).

In this study, it is assumed that NodeName which is used to execute test cases is indifferent i.e., results are the same regardless of the ...

It is found earlier that in the dataset of software release L16A, there is one variable which is an exact linear combination of the other variables. Hence, the variable is excluded from the regression model. used only 57 test cases in training the model. The data or in this case number of test cases in each software release is rather small which make it easier for an occurrence of singularity. Therefore, unless there is more data, it is better to drop the variable from the regression model before doing a further analysis.

One has to understand and accept a risk when deciding on the number of states. There is no guarantee how many states would yield the best outcome. The one which are more appealing and desirable are selected. The same situation also applies to determining the number of switching coefficient in the model.

Markov switching model with four states is also tested in the analysis but results are rather poor compare to the model with two and three states. The four-state model gives more difficulty and perhaps impossible to make an inference on the states.

One benefit from using the non-parametric analysis is that it does not require a prior assumption of the data distribution. In this thesis framework, it is used for comparing with the results from Markov switching model. The non-parametric analysis is able to detect any type of distributional change within a time series

The computational time of the E-divisive algorithm is $O(kT^2)$, where k is the number of estimated change points and T is the number of observation in the time series data.

Simulation -> easy simulated data –> high accuracy

L16B_NNY -> smallest BIC but graph is not interpretable

Software release L16A

Software release L16B

Software release L17A

6. Conclusions

It can be concluded that

A. Software

Both R and Python are powerful programming languages for data analysis. Python has been known as a general purpose language with an easy to understand syntax. It emphasizes on code readability and has a gentle learning curve. R is developed by and for statistician. It provides a huge number of essential packages in statistics, and even starts to expand to different fields. R also has a strong reputation for data visualization. However, in terms of computation, R still cannot compete with Python which is built specifically for the computer programming. With strengths and weaknesses between R and Python described earlier, R is chosen to be used in this study. R offers more implemented algorithms to solve the problem at hand. Moreover, an effective visualization helps analyst and user to understand more about their data especially for the complex one. Visualizing data is useful in many ways such as uncover new patterns, identify some factors, and get an overall trend. R proves itself a good choice since it provides a great feature in creating an interactive graphic and visualization.

B. MSwM Package

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

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

Non-switching effect When setting variance to have a non-switching effect, the function generates a warning. It proved that there is a minor mistake in the code.

Non-invertible Hessian The package uses a Hessian, the matrix of second order partial derivatives with respect to parameters, for numerical optimization. In some cases, Hessian matrix will not be invertible as the matrix is singular. Consequently, the function can not compute 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).

Categorical variable The package does not work properly well with categorical predictor variables. Hence, a further implementation in the code for handling with categorical variables is done. Factors are used to represent the categorical variables in R. Most of the time, the order of the factor does not matter. However, sometimes the order of the factor is of interest in the analysis and needs to be specified for comparison or meaningful interpretation.

NA coefficients When performing the Markov switching model, the function in the package first randomly divides the data into different subsets and separately fit the model to get initial coefficients in each state. These coefficients are later used for further analysis e.g., computing condition means, error and likelihood 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 will also become NAs.

Due to the randomness in partitioning the data, sometimes a variable in a subset will have the same value in all observations. Then, NA coefficient is generated for this specific variable because of singularity. This NA coefficient issue occurs more frequently with a categorical variable as each categorical variable has its own number of levels. It is rather difficult for every subset to contain all levels of the variable especially if the variable contains 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.

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. The parameters in each state will then have the same coefficient values. It proved that dividing data into subsets is only a method to get the starting values for coefficients in each state and does not have much effect for the algorithm.

Predict function The implementation for the model prediction is added in the package. This function predicts the most probable state that a new observation will belong to. The input of the predict function is the training model and the new set of data. The function computes the filtered probabilities or the Equation 3.4. The output from the function is the state for the new observation which has the highest probability.

C. Output

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.

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.

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

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.

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

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.

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 change-point 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.

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.