



Chair of Petroleum and Geothermal Energy Recovery

Master's Thesis



Application of hidden Markov model in
production data analysis

Mehdi Mirzaei Tashnizi, BSc

May 2022

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Datum: 24.05.2022



Unterschrift Verfasser

Mehdi Mirzaei Tashnizi
Matrikelnummer: m00135277

AFFIDAVIT

I hereby declare that the content of this work is my own composition and has not been submitted previously for any higher degree. All extracts have been distinguished using quoted references and all information sources have been acknowledged.

Danksagung / Acknowledgement

First, I would like to thank Dipl.-Ing. Dr. Mont. Rudolf Fruhwirth for always giving me valuable advices when I ran into trouble. Thank you for always steering me in the right direction whenever I needed your help.

I would also like to thank Univ.-Prof. Dipl.-Ing. Dr. Mont. Herbert Hofstätter for giving me this great opportunity to do this interesting thesis.

Last but definitely not least, I would like to thank my dear wife and daughter for supporting me throughout my studies.

Kurzfassung

Wie wir es aus der Medizin kennen, ist Vorbeugen besser als Heilen. Um zukünftige Probleme zu vermeiden, müssen wir sie früher erkennen, daher brauchen wir vorhersagen. Da „machine learning Algorithmen“ das Potenzial haben, genauere Vorhersagen zu treffen, haben viele Wissenschaftler und Forscher usw. bereits damit begonnen, sie zu verwenden. Diese maschinellen Lernalgorithmen werden verwendet, um Vorhersagen zu erstellen, und suchen auch nach Mustern innerhalb der Wertelabels, die Datenpunkten zugeordnet sind.

Es gibt zwei Hauptgruppen des maschinellen Lernens, „supervised learning“ und „unsupervised learning“. Zusätzlich gibt es auch die sogenannten „semi-supervised learning“ Methode, die eine Kombination der beiden Hauptlernmethoden darstellen.

In dieser Masterarbeit wird das Hidden-Markov-Modell (HMM), eine unsupervised learning Methode, verwendet, um „time series data“ zu analysieren und die verborgene zustände (hidden states) zu finden, die für Vorhersageprobleme verwendet werden können, die in Ölfeldern, insbesondere in der Erdölförderung, auftreten können, z. B. Fehlerdiagnose von „Sucker Rod Pumpen (SRP)“.

Diese Masterarbeit beginnt mit den Grundlagen und der Theorie des HMM. Dann werden die drei Hauptprobleme von HMM und die Lösungen der Probleme diskutiert. Darüber hinaus werden die verfügbaren Werkzeuge und Programmiersprachen zur Generierung eigener Algorithmen und Funktionen, die für das Modell erforderlich sind, diskutiert. Dann wird das Hidden-Markov-Modell verwendet, um den Beginn und das Ende von „Upstroke und Downstroke“ aus dem Datensatz zu finden. Schließlich, HMM wird eingesetzt, um den Betrieb der Sucker Rod Pumps im Laufe der Zeit zu beobachten (Auffinden verborgener Zustände), zuerst für den gesamten Datensatz und dann für einen ausgewählten Teil des Datensatzes. Die Ergebnisse des Hidden-Markov-Modells werden mit anderen Clustering-Methoden verglichen, nämlich der Gaussian-Mixture und K-Means.

Abstract

As we know from medicine, prevention is better than cure. To avoid future problems, we have to recognize them earlier; therefore, we need prediction. Because machine learning algorithms have the potential to make more accurate predictions, many scientists and researchers, etc. have already started using them. Although these machine learning algorithms are used to create predictions, they also look for patterns within the value labels assigned to data points.

There are two main types of machine learning, supervised learning and unsupervised learning. In addition, there are also the so-called semi-supervised learning methods, which are a combination of the two main learning methods.

In this thesis, the hidden Markov model (HMM), an unsupervised learning method, is used to analyze time-series data and find the hidden states, which can be used for predicting problems that may arise in oil fields, especially in petroleum production, e.g., sucker rod pump failure diagnosis.

This thesis starts with the basics and theory of HMM. Then the three main problems of HMM and the solution for the problems will be discussed. Moreover, the tools and programming languages available to generate our own algorithms and functions required for the model will be discussed. Then hidden Markov model will be used to find the start and the end of up-and downstrokes from the dataset. Finally, using HMM to observe the sucker rod pump operation over time (finding hidden states), first for the entire dataset and then for a selected part of the dataset. The results from the hidden Markov model will be compared with other clustering methods, namely the Gaussian Mixture and K-Means.

Table of content

	Page
1 INTRODUCTION.....	9
2 LITERATURE REVIEW	10
2.1 General information	10
2.2 Application areas	10
2.3 HMM in oil business.....	11
3 HIDDEN MARKOV MODEL (HMM)	12
3.1 Basics	12
3.2 Theory of HMM	19
3.3 Tools available	41
4 SUCKER ROD PUMP ANALYSIS USING MACHINE LEARNING	44
4.1 Sucker rod pumping system	44
4.2 Machine learning	45
4.3 Motivation	46
4.4 Objective	46
5 PROCEDURE AND IMPLEMENTATION OF THE PROCEDURE.....	47
5.1 Data preparation	49
5.2 Procedure	51
6 RESULTS	53
6.1 Stroke duration and strokes per minute.....	53
6.2 Upstroke & downstroke from DC	55
6.3 Sucker rod pump operation over time.....	57
7 CONCLUSION	72
8 PUBLICATION BIBLIOGRAPHY	73
LIST OF TABLES	75
LIST OF FIGURES.....	76
ABBREVIATIONS.....	79
APPENDICES	80
Appendix A.....	80
Appendix B.....	81

Appendix C..... 84

1 Introduction

Before even starting with Hidden Markov Model (HMM), it is important to know where HMM can be used and which type or kind of problems can be solved by using such a model.

Rapid advances in the development of computer technology led scientists and researchers to pursue more demanding goals. The development of computer technology helps to surpass human abilities in certain areas, which is also the case in petroleum engineering. Meanwhile, Ordinary human intelligence no longer appears on the symbolic level but in the ability to process various sensory input data. Strive to create *artificial intelligence*, and *machine learning* (the most famous method is the so-called artificial neuronal network) has increased. However, such powerful learning methods as an *artificial neuronal network* are too general and therefore not applicable for some purposes; hence, other statistical modeling such as *Markov models* are applied. Markov models are widely used for modeling chronologically organized data. The most task areas of the Markov model are speech recognition, handwritten recognition, and bioinformatics (human genome). The application of the Markov model in the areas mentioned has been so successful that this model is recently used for different analysis tasks.

In addition to the advantage mentioned about the Markov model, there are other two strong reasons that this model, or Hidden Markov Model (HMM) became popular in the last year. First, the model is very rich in the mathematical structure and hence can form the theoretical basis for use in a wide range of applications. Second, the model, when applied properly, works very well in practice for several important applications (Rabiner 1989).

Nowadays, time-series data are in all fields and areas widely available, which is also the case in petroleum engineering, especially in petroleum production and drilling. Therefore, the importance of time-series data has increased because of knowledge that can be obtained from time-series data, and the further development of classification methods is also another reason.

Considering the advantages of the HMM, knowledge that can be obtained from time-series data, and the fact that this model is widely used in all fields but has not been used much in the oil field motivate me and encourage me to work on this topic. Therefore, this master thesis pursues to analyze production data using this model.

2 Literature review

2.1 General information

The basic theory of the Markov chain has been known to us for more than 100 years, but it has only been used explicitly in research institutions for about 50 years. The application of this model started with problems in speech recognition, but through constant refinements, both in theory and in the implementation of the Markov model, this technique has always been improved, and the scope of this method has become wider.

2.2 Application areas

“The most common application area of this technology is the automatic recognition of speech. At the beginning of the respective research, for quite a long time, it competed with symbolic approaches. However, the availability of large sample sets of speech data heralded the triumph of statistical methods. Therefore, meanwhile hidden Markov models for describing acoustic events in combination with Markov chain models for the statistical modeling of word sequences on the symbolic level represent the standard technology for building successful automatic speech recognition systems” (Fink 2014).

“Only in recent years have these methods entered a both thematically and sensorily related application area. The automatic recognition of handwritten texts — in the same way as automatic speech recognition — can be considered a segmentation problem of chronologically organized sensor data. There the time axis either runs along the text line to be processed or along the line of writing itself. By this "trick" statistical modeling techniques known from the field of automatic speech recognition usually with minor changes only can be transferred to the problem of processing handwritten documents” (Fink 2014).

“A third important application area of Markov models takes us beyond the field of man-machine interaction. Bioinformatics research is primarily concerned with cell-biological processes and their simulation and analysis by means of computer science techniques. Special attention currently lies in the analysis of the human genome. From the view of statistical pattern recognition, this genetic information — and sell products like, e.g., RNA or proteins derived from it — essentially consists of linearly organized symbol sequences. Though for quite some years, statistical techniques are used for the analysis of such biological sequences, the attention of bioinformatics research was only recently drawn to Markov models. The success of the respective methods in this application area was so convincing, that meanwhile several software packages for the application of Markov models as well as libraries of ready-made models for different analysis tasks are available” (Fink 2014).

2.3 HMM in oil business

As mentioned earlier, Hidden Markov Model (HMM) has not been widely used in the oil and gas industry. The most important application area of HMM in the oil business is for predicting *future crude oil prices*.

A large proportion of the energy required for industry worldwide is obtained from oil; thus, the crude oil price plays a large and important role in the world economy. A lot of research has been done in this area, as predicting the oil price is not just a concern of the oil and gas industry.

But since there are so many factors that influence the level of the oil price, predicting the oil price is really complicated. Different tools and methods are used, and HMM is also one of these methods. For this purpose, some have focused on *factors* that influence oil price, for example, OPEC's decision, wars, renewable energy, financial crises etc., and some others on different *approaches* like regression model, linear and non-linear time series model, and so on.

Another important application area of HMM in the oil business is *Lithology Identification*. Lithology identification of subsurface reservoirs is a challenging task because different reservoirs show different subsurface petrophysical properties, which affects the identification of lithology. The idea was a combination of HMM and random forests, where the well (logging) data and seismic data were used. Results of predicted lithology and shale content match successfully with real logging data.

Apart from the two areas mentioned above, there are some individual cases, such as the prediction of *stuck pipes during drilling*, predicting *risk severity of workers in the oil and gas sector (HSE)*, *estimation of petroleum reservoir categorical variables*, and so on, where HMM is used in the oil business.

3 Hidden Markov model (HMM)

Hidden Markov Model is a doubly embedded stochastic process with an underlying stochastic process that is not observable (it is hidden) but can only be observed through another set of stochastic processes that produce the sequence of observations (Rabiner 1989). That means the first stage is a discrete stochastic process known as Markov Chain, and the second stage is a stochastic process, where for each time step (t), an observation (V_t) is generated.

To understand HMM and get a better idea of what all these terms mean, we need to know some basic concepts such as random variables, random processes, stochastic and deterministic processes, discrete or continuous stochastic processes etc., and then Markov processes and Markov chain.

3.1 Basics

Hidden Markov Model (HMM) is a special form of the Markov process or Markov model. Thus, before even starting with HMM, we have to know, what are Markov models or Markov processes and Markov chains, because Hidden Markov Models are built on these basic concepts and their properties. But again, to understand Markov models and Markov chains, we need to be familiar with some basic terms from statistics.

We'll start with an example, as is common in statistics. Imagine tossing a coin, and the outcomes can be head or tail, which is random. In mathematics and statistic, they use the concepts of a random variable to define and describe such experiments. A *random variable* is a variable that its potential value is one or more outcomes of a random phenomenon or experiment. The possible values in our example are head or tail; these possible values or states are known as *the domain of the random variable*, also known as *sample space*, given by $\text{Domain}(\text{Outcome}) = (\text{Head}, \text{Tail})$ with a probability distribution of $P(\text{Head}) = 1/2$ and $P(\text{Tail}) = 1/2$ (assuming fair coin). In this case, the domain of the states has discrete variables; therefore, such random variables are known as *discrete random variables*. A random variable is said to be discrete, if it has a finite or countably infinite number of values, which is the case in our example. Probability mass functions (PMS) are used to represent possible outcomes of discrete random variables. There are also *continuous random variables*, which can take any arbitrary values. An example of continuous random variables is the future oil price or stock price of APPLE the next day. In the case of continuous random variables, probability density functions (pdf) are used to represent possible outcomes.

In our example, we talked about outcomes of a single random phenomenon, but what about random events over a certain period of time. In this case, we will have a set of random variables, where each random variable represents the outcome or outcomes at the given time. As an example, consider we want to represent the oil price for the whole of next week with a time step of 24 hours (one day). Now we have a set of random variables that represent random variables (oil price) over a period of time (each day); these sets are known as *random processes*.

As mentioned earlier, HMM is a doubly embedded stochastic process. Now it is important to know what stochastic and deterministic random processes are?

We start again with an example. Just imagine a car standing and not moving. Now the car starts to move with a velocity/ speed that does not change over time. If we know the velocity at which the car is moving, we will be able to determine the distance that this car has traveled at any given time. Such random processes, where we are able deterministically to find the value/ state of the random variable by knowing the initial conditions (here, the car with constant velocity) and parameters of the system (acceleration yes or no, if yes, constant rate or not), are known as *deterministic random processes* or *deterministic processes*. The opposite of a deterministic process is a stochastic process, where we are not able deterministically to calculate the value/ state of a random variable for a given time, even if we know the probability distribution of the next state. Now think about the previous example, representing the future price of oil. Even if we know the probability distribution of the next state, initial conditions, and also parameters of the system, nevertheless it is not possible to calculate the exact price of oil for the next step of the time. Such cases are known as *stochastic random processes* or *stochastic processes*. Depending on the type of domain of the random variables, there are discrete and continuous stochastic processes (or deterministic processes).

3.1.1 Markov models and Markov chains

A stochastic process where the state of a random variables at time t depends only on the previous state at time $t-1$ is called *Markov process* or *Markov model*. This property, that the future state of the system depends only on the current state and not on any other previous states, is known as the *Markov property*.

According to the discrete and continuous time and also discrete and continuous state space¹, there are four types of Markov processes (Ankan and Panda 2018):

	Discrete State Space	Continuous State Space
Discrete Time	discrete time and discrete state space (<i>Markov chain</i>)	discrete time and continuous state space (<i>Markov chain</i>)
Continuous Time	continuous time and discrete state space	continuous time and continuous state space

The simplest types of Markov processes are Markov chains. Markov chains are discrete-time Markov processes, but there is no decisive agreement under authors and researchers about the use of some terms that refer to specific cases of Markov processes. However, most authors and researchers use the Markov chain to refer to a process with discrete-time. That means, in a Markov chain the time is discrete, but there is no limitation for state-space (at least no definitive agreement). Ultimately we can say, a stochastic Markov process (with discrete-time) that satisfies the Markov property, namely that the probability of being in the current state

¹ State space is a set of all feasible/ possible states of our system (domain of the random variables).

depends only on the last previous state, is called a *Markov chain*. They are important because the basic concepts of HMM are built on the Markov chain and its properties.

Markov chains have different properties, most of them are outside the scope of this thesis, but there are three properties that are important to know. The first one is *irreducibility*, when in a Markov chain, any state is accessible from any other state. The next one is *ergodicity* when a certain state is recurring. And the next one is *the final or absorbing state*, when the system enters a state and cannot leave this state again (the transition probability of this state to any other state is equal to 0, except for itself).

To get an idea of what Markov chains are, we will go through a simple example of a Markov chain. Consider there are three weather options, sunny, cloudy, and rainy, as illustrated in figure 1.

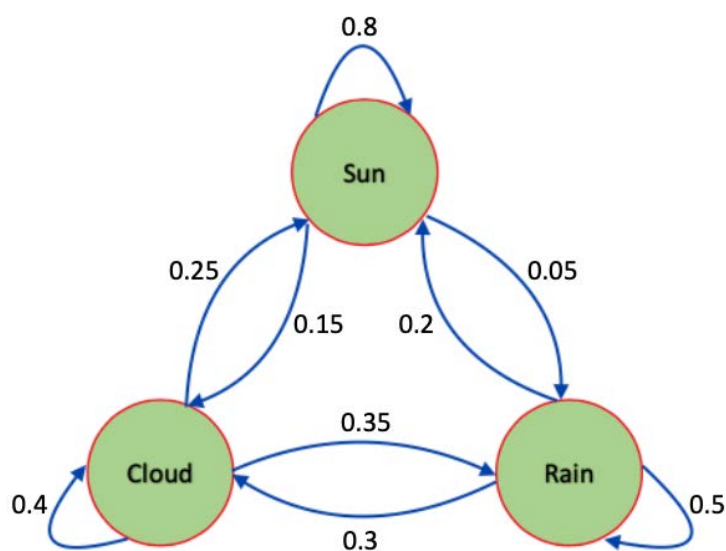


Figure 1: A simple Markov chain

At equally spaced (discrete) time, the weather will change from one state to the next (discrete state space) according to the transition probabilities (we will later discuss the term transition probability in details). In case that the tomorrow's weather is only dependent on the today's weather and not any other days before (known as *first-order Markov chain*), the probability of weather at time $(t+1)$, given weather at time (t) , can be expressed as follows:

$$P(\text{weather}(t + 1) | \text{weather}(t))$$

As you may have noticed, Markov chains consist of three parameters; possible states, initial- and transition probability (these terms will be discussed later in detail), and also that events happen, that their occurrence depends on other events. Therefore, we also need to know what conditional probabilities and Expectation-Maximizations are, before we start with HMM itself.

3.1.2 Conditional probabilities

Conditional probability is a measure of the probability of an event occurring, given that another event (by assumption, presumption, assertion, or evidence) has already occurred. If the event of interest is A and the event B is known or assumed to have occurred, "the conditional probability of A given B" or "the probability of A under the condition B", is usually written as $P(A|B)$. It does not necessarily have to be a relationship between A and B, or necessarily be dependent on each other or occur simultaneously. This can also be expressed as the fraction of probability B that intersects with A (Wikipedia o. J.b):

$$P(A | B) = \frac{P(A \cap B)}{P(B)}$$

For example, we know that the possible outcomes of tossing a (fair) die are {1, 2, 3, 4, 5, 6}. The probability that we roll the die and we get '2' is 1/6: $P(2) = 1/6$. This is an unconditional probability, which is the opposite of conditional probability. Unconditional probability is the probability of an occurrence or event without any other occurrence or event being taken into account. We roll the die again, but this time the probability of getting '2', given that the number showing up is an even number, is 1/3 as there exist three even numbers: $P(2|\text{even}) = 1/3$. In this case, it is a conditional probability.

It should be clear that $P(A|B)$ typically differs from $P(B|A)$, and falsely equating the two probabilities can lead to various errors of reasoning. For example, if a person has dengue fever, they might have a 90% chance of testing positive for the disease. In this case, what is being measured is that if event B (having dengue) has occurred, the probability of A (testing positive) given that B occurred is 90%: $P(A|B) = 90\%$. Alternatively, if a person tests positive for dengue fever, they may have only a 15% chance of actually having this rare disease due to high false-positive rates. In this case, the probability of event B (having dengue) given that event A (testing positive) has occurred is 15%: $P(B|A) = 15\%$ (Wikipedia o. J.b).

A conditional probability table (CPT) is a very useful option to show conditional probabilities, and it illustrates the relationship between events. A conditional probability table (CPT) is defined as a set of discrete and mutually dependent random variables to display conditional probabilities of a single variable with respect to the others. A conditional probability table can be put into matrix form. As an example, suppose that two binary variables, X and Y have the joint probability distribution given in Table 1, where each of the four central cells shows the probability of a particular combination of X and Y values (Wikipedia o. J.b):

Table 1: Conditional probability table (CPT)

	X = 0	X = 1	P (Y)
Y = 0	4/9	1/9	5/9
Y = 1	2/9	2/9	4/9
P (X)	6/9	3/9	1

Joint probability is the probability that event A and event B occur simultaneously. It is the probability of the intersection of two or more events. The probability of the intersection of A and B may be written as $P(A \cap B)$ (Wikipedia o. J.b).

The first column sum is the probability that $X = 0$ and Y equals any of the values it can have; that is, the column sum $6/9$ is the marginal probability that $X = 0$. Marginal probability is the probability of event A occurring, expressed as $P(A)$, which can be considered as unconditional probability, as its occurrence is not conditioned to any other event. If we want to find the probability that $Y = 0$ given that $X = 0$, we compute the fraction of the probabilities in the $X = 0$ column that have the value $Y = 0$, which is $4/9 \div 6/9 = 4/6$. Likewise, in the same column, we find that the probability that $Y = 1$ given that $X = 0$ is $2/9 \div 6/9 = 2/6$. In the same way, we can also find the conditional probabilities for Y equalling 0 or 1, given that $X = 1$. Combining these pieces of information gives us this table of conditional probabilities for Y (Wikipedia o. J.b).

3.1.2.1 Bayes' theorem

While conditional probabilities can provide extremely useful information, limited information is often supplied or at hand. Therefore, it can be useful to reverse or convert a condition probability using Bayes' theorem (Wikipedia o. J.a):

Bayes' theorem (also known as Bayes' Law or Bayes' Rule) offers a way to modify the existing forecasts by using some new or additional information. It describes the probability of an event based on prior knowledge of conditions that might be (is or can be) related to the event. For example, if the risk of developing health problems is known to increase with age, Bayes' theorem allows the risk to an individual of a known age to be assessed more accurately (by conditioning it on their age) than simply assuming that the individual is typical of the population as a whole. Bayes' theorem is mathematically expressed as (Wikipedia o. J.a):

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$

Where A and B are the events, and have to be different events, and $P(B) \neq 0$.

- $P(A|B)$ is a conditional probability, the probability of event A occurring given that B is true. It is also called the posterior probability of A given B.
- $P(B|A)$ is also a conditional probability, the probability of event B occurring given that A is true. It can also be interpreted as the likelihood of A given a fixed B.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B without any given conditions (unconditional probabilities). They are known as the marginal or prior probability.

In order to derive posterior probabilities, Bayes' theorem relies on adding prior probability distributions. In Bayesian statistical inference, prior probability is the likelihood of an event occurring before additional data is gathered. The updated probability of an event occurring after additional information is taken into account is known as posterior probability. Using Bayes' theorem, the posterior probability is derived by updating the prior probability. The posterior probability, in statistical terminology, is the likelihood of event A happening after event B has already occurred (Wikipedia o. J.a).

The main application of Bayes' theorem is Bayesian inference, a special approach to statistical inference. Bayes' theorem is very popular in finance to rate the risk and medicine to determine the accuracy of test results.

The following example helps us to understand Bayes' theorem. "Suppose a particular test for whether someone has been using cannabis is 90% sensitive, meaning the true positive rate (TPR)=0.90. Therefore, it leads to 90% true positive results for cannabis users. The test is also 80% specific, meaning true negative rate (TNR)=0.80. Therefore, the test correctly identifies 80% of non-use for non-users but also generates 20% false positives, or false-positive rate (FPR)=0.20, for non-users.

Assuming 0.05 prevalence, meaning 5% of people use cannabis, what is the probability that a random person who tests positive is really a cannabis user?

The Positive predictive value (PPV) of a test is the proportion of persons who are actually positive out of all those testing positive and can be calculated from a sample as:

$$\text{PPV} = \text{True positive} / \text{Tested positive}$$

If sensitivity, specificity, and prevalence are known, PPV can be calculated using the Bayes theorem. Let $P(\text{User}|\text{Positive})$ mean "the probability that someone is a cannabis user given that they test positive," which is what is meant by PPV. We can write:

$$P(\text{User}|\text{Positive}) = \frac{P(\text{Positive}|\text{User}) P(\text{User})}{P(\text{Positive})}$$

The application of the *Law of Total Probability* leads to the fact that:

$$P(\text{Positive}) = P(\text{Positive}|\text{User}) P(\text{User}) + P(\text{Positive}|\text{Non-user}) P(\text{Non-user})$$

Therefore, we can rewrite the above equation as:

$$P(\text{User}|\text{Positive}) = \frac{P(\text{Positive}|\text{User}) P(\text{User})}{P(\text{Positive}|\text{User}) P(\text{User}) + P(\text{Positive}|\text{Non-user}) P(\text{Non-user})}$$

And after inserting associated values into the equation, we obtain the following probability:

$$P(\text{User}|\text{Positive}) = \frac{0.90 * 0.05}{0.90 * 0.05 + 0.20 * 0.95} = \frac{0.045}{0.045 + 0.19} \approx 19\%$$

Even if someone tests positive, the probability they are a cannabis user is only 19% because in this group, only 5% of people are users, and most positives are false positives coming from the remaining 95%" (Wikipedia o. J.a).

3.1.3 Expectation - Maximization (EM) algorithm

The target of the Expectation-Maximization (EM) algorithm is to estimate (local) maximum likelihood parameters of a statistical model if there is no possibility of solving the equations directly. It could either be because there are missing values under the data, or the model could be formulated more simply, assuming that additional not-visible (unobserved) data points are

available. Normally, such a model consists of *latent (hidden) variables*, *unknown parameters*, and *known observation data points*.

Often, in real cases of problem statements, we have a lot of relevant characteristics to build a model with, but only a tiny percentage of them are observable. Since the values for the latent (hidden) variables are unknown, the Expectation-Maximization method attempts to estimate the optimal values for these variables using the available data and, after that determine the model parameters.

The EM algorithm can generally be called a latent variable model, which contains observable as well as unobservable variables. Observed variables are those variables, which can be determined, but the unobserved (latent) variable can be derived from observed variables.

The name Expectation-Maximization is derived from the two major processing steps that are performed within the iterative re-estimation procedure. In the expectation step or E-step, the EM algorithm tries to use existing observed data of the dataset to estimate the missing data or values of hidden variables. Then in the maximization step or M-step, this data or values will be used to update the values of the parameters.

The following four main steps are done by applying the EM algorithm, also illustrated as a flow chart in figure 2 (Analytics Vidhya o. J.):

- **Initialization step:** In this step, we initialize the parameter values with a set of initial values, then give the set of incomplete observed data to the system with the assumption that the observed data comes from a specific model i.e., probability distribution.
- **Expectation step (E-Step):** In this step, by using the observed data to estimate or guess the values of the missing or incomplete data. It is used to update the variables.
- **Maximization step (M-Step):** In this step, we use the complete data generated in the “Expectation” step to update the values of the parameters, i.e., update the hypothesis.
- **Checking of converge step:** Now, in this step, we checked whether the values are converging or not; if yes, then stop otherwise repeat these two steps, i.e., the “Expectation” step and “Maximization” step, until the convergence occurs.

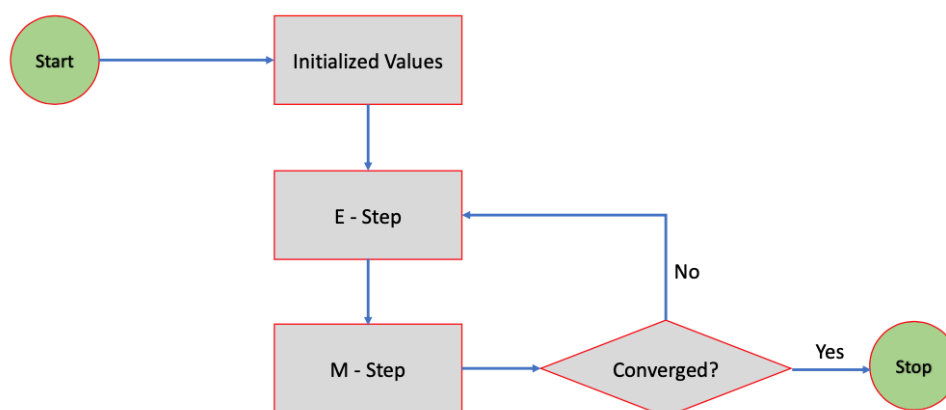


Figure 2: Flow chart of an EM algorithm

The scope of applications of the EM algorithm is wide; some of the most important fields where EM can be used are, fill in the missing data during a sample, calculating the Gaussian density of a function, application in several areas of NLP (Natural Language Processing), reconstruction of images, especially in medicine etc. But the most important application of EM in our case is the estimation of parameters of a Hidden Markov Model (using the Baum-Welch algorithm) and also finding values of hidden (latent) variables.

Of course, like all other methods, EM also has some advantages and disadvantages. The maybe biggest advantage is that the implementation of the E-Step and M-Step is pretty easy, and usually an increase in likelihood can be expected after each step of the iteration. The disadvantages of the EM algorithm are that it converges slowly and it converges to the local optimum only. Another disadvantage is that it considers both forward and backward probability, which is in contrast to that of numerical optimization, which takes only forward probabilities into account.

To estimate the parameters of Hidden Markov Models, the Baum-Welch algorithm is used, which is a special form of EM algorithm. The Baum-Welch algorithm and how it works will be later explained in the section evaluation of HMM in detail.

3.2 Theory of HMM

Now, where we are familiar with the necessary basics of statistics, and also know, what Markov processes and Markov chains are, we can start with Hidden Markov Model itself.

As mentioned, Hidden Markov Model is a doubly embedded stochastic process with an underlying stochastic process that is not observable (it is hidden) but can only be observed through another set of stochastic processes that produce the sequence of observations (Rabiner 1989). In other words, in HMM as the name suggests, the states are hidden, but at any time step, they emit a visible symbol to us.

Let's start again with an example. Imagine the following scenario, and there is a man who talks on the phone every day with his daughter, that lives in another city far away from him. His daughter has a mood that changes with the weather. He knows that if it's sunny, she is 80% happy and 20% sad, and if it's cloudy, she is 50% happy and 50% sad, and finally when it's rainy, she is 25% happy and 75% sad, as illustrated in figure 3. Now let's say, last week that they talked, he observed the following observation sequence: Mood = [Happy, Happy, Sad, Happy, Sad, Sad, Happy]. Now the question is, how was the weather last week in the city where she lives?

In this example the states (weather) are not visible to him but her moods in the last week are the visible symbols that she emitted to him. Therefore, the sequence of Mood is the observation, and the weather that we want to find out (based on observation sequence), is the hidden state sequence.

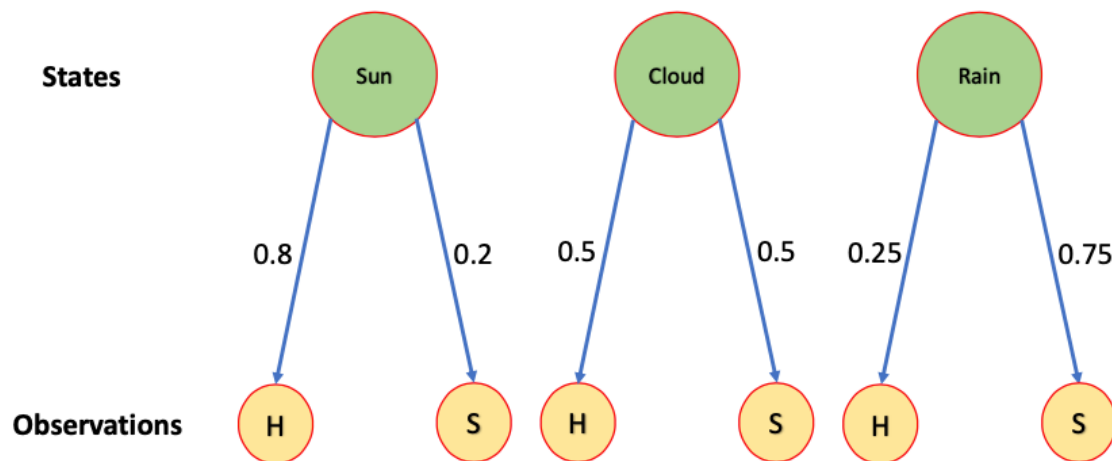


Figure 3: States and observations diagram of the weather

To get the hidden states with HMM, we need to parameterize HMM (initial distribution, transition probability and emission probability), and also find out how we can solve the three basic problems of HMM (evaluation problem, optimization problem and predicting best sequence).

What these terms are and what role they play at HMM will be discussed in detail in the next two sections.

3.2.1 Parameterizing of HMM

The previous example gave us an idea how an HMM works, now we can define and explain the elements of HMMs. An HMM consists of following parameters:

- N: The number of possible states of the system $S = \{S_1, S_2, \dots, S_N\}$
- M: The number of possible observations per state $O = \{O_1, O_2, \dots, O_M\}$
- V: The Observation Sequence $V = \{V_1, V_2, \dots, V_T\}$
- Q: The states sequence $Q = \{q_1, q_2, \dots, q_T\}$
- T: The length of observation and/ or state sequence
- a: Transition probability
- b: Emission probability
- π : Initial probability of states

For a complete description of an HMM, at least five elements need to be specified, the two model parameters N and M, and the probability measures a, b and π . But to parameterize an HMM, we need these three important parameters, namely initial distribution π , transition probability a, and emission probability b. Now let's find out what initial distribution, transition probability, and emission probability are.

The probability of moving from one state to the next one is called *transition probability*. Transition probabilities are normally denoted as a_{ij} , which means the probability of moving from state i to state j. The general equation of transition probability can be expressed as follow (Rabiner 1989):

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i) \quad (\text{Eq. 1})$$

with $1 \leq i, j \leq N$, and the transition probability distribution $A = \{a_{ij}\}$. As mentioned earlier, when the system enters a state and cannot leave this state again is known as the final or absorbing state, in such cases, the $a_{ij} = 0$ and $a_{ii} = 1$. Let's go back to the example that we mentioned earlier in part Markov chain to get a better idea about transition probability. Consider there are three weather options, sunny, cloudy and rainy. The states and the associated transition probabilities will be shown in figure 4.

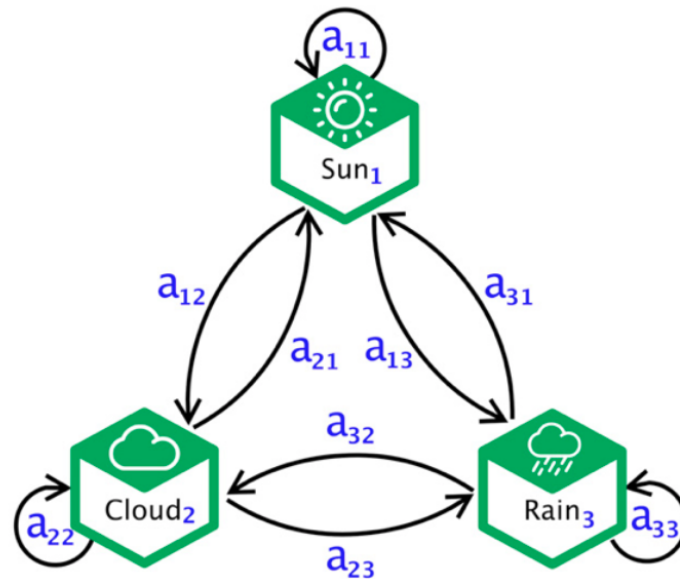


Figure 4: States and the transition probabilities (A Developer Diary o. J.)

In this case, in which we have three states, we will have nine transition probabilities, where staying in the same state is also possible. The transition probability matrix A looks as follows:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Table 2 shows the transition probability matrix for the Markov chain illustrated in figure 1 (part Markov chain). This table shows the probability values for the weather, moving from the states in the rows to the states in the column. As an example, the transition probability of having a rainy day after a cloudy day is defined as a_{23} , which has a probability of 35%.

Table 2: Transition Probability Matrix

	Sunny	Cloudy	Rainy
Sunny	0,8	0,15	0,05
Cloudy	0,25	0,4	0,35
Rainy	0,2	0,3	0,5

It is self-evident that the sum of all transition probabilities of each row should be equal to 1.

Up to here we have always considered the next states, or considered a state where another state happened before. But what about the first step or state of the system at time step $t = 1$. This initial states of HMM, which is donated as π is called *initial probability or initial probability distribution*. In our example, with three possible weather states, the probability to be at time $t = 1$ in one of these three states is $1/3$, therefore the initial probability distribution is defined as $\pi = [1/3, 1/3, 1/3]$. Here also the sum of all probabilities should be equal 1. General equation of initial probability can be expressed as follow (Rabiner 1989):

$$\pi = P(q_1 = S_i) \quad (\text{Eq. 2})$$

with $1 \leq i \leq N$, and the initial probability distribution $\pi = \{\pi_i\}$.

Now let's go back to the previous example with the man and his daughter. As you can remember, the man knew if the weather is sunny, she is to 80% happy and 20% sad, and if it is cloudy, she is to 50% happy and 50% sad, and finally when it is rainy, she is to 25% happy and 75% sad. In this case, the weather is the hidden state and the moods, which are visible to us, are observations. The *emission probability* b_{ij} indicates the probability of making observation V_j at state S_i , as illustrated in figure 4. The general equation of emission probability can be expressed as follow (Rabiner 1989):

$$b_j(k) = P(V_k \text{ at } t \mid q_t = S_i) \quad (\text{Eq. 3})$$

with $1 \leq j \leq N$ and $1 \leq k \leq M$, and the emission probability distribution $B = \{b_{ij}\}$. In figure 5, V_1 stands for Happy and V_2 for Sad, therefore, one of these observations ($V_1 = \text{Happy}$ or $V_2 = \text{Sad}$) have to be emitted from each state.

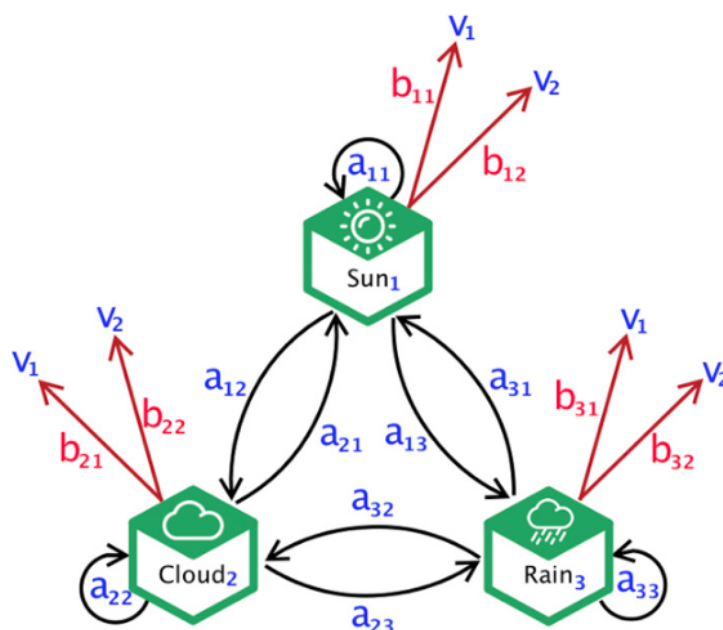


Figure 5: States and the associated transition and emission probabilities (A Developer Diary o. J.)

Emission probabilities can be like transition probabilities defined as matrix B, where the states in the rows emit the observations in the columns:

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

Table 3 shows the probability values for the Markov chain illustrated in figure 3. Of course, also here the sum of all emission probabilities of each row should be equal to 1.

Table 3: Emission Probability Matrix

	Happy	Sad
Sunny	0,8	0,2
Cloudy	0,5	0,5
Rainy	0,25	0,75

Please note that if the visible symbol V_n is discrete (like in the previous example), the emission probability might be a conditional probability (see table 3), but if the visible symbol V_n is continuous, then the distribution may take a Gaussian distribution, with the parameters mean μ and variance σ .

From now on, we use $\lambda = [A, B, \pi]$ for convenience to point all parameters of the model.

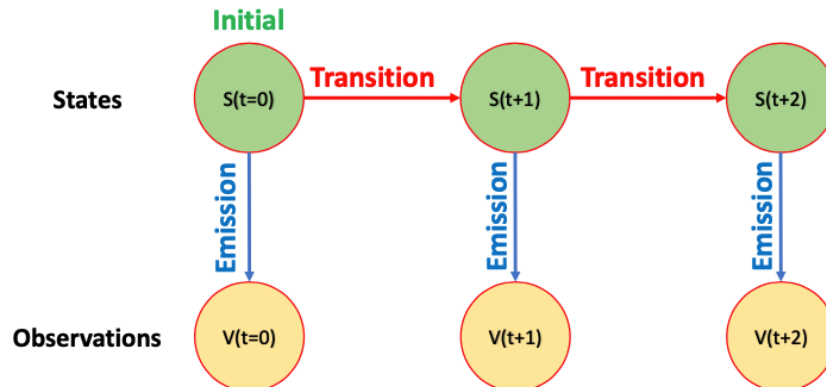


Figure 6: A summary of the hidden Markov model and its components

Now that we have learned something about the model parameters, we are one step closer to our goal of predicting hidden states. Before diving into the next topic, just to have an idea, let's briefly talk about what happens if one or more of these parameters is missing?

If the model parameters λ (a , b , and π) and the number of possible states N are known to us, we can use the forward algorithm and then the Viterbi algorithm to obtain the most probable state sequence (hidden states). If it is not the case, that means model parameters λ and/or possible states are unknown, and we have to find these parameters and values from the data. We can use for example, Gaussian Mixture to get the possible states. But if only the model parameters are missing, we can use the Baum-Welch algorithm to estimate these parameters.

Forward, Backward, Viterbi and Baum-Welch algorithms and also Gaussian Mixture will be discussed in detail in the next sections.

3.2.2 Evaluation of HMM

In previous sections, we discussed the Markov chain, the basics and theory of HMM, and the most important parameters of HMM and how they can be defined. In this section, we come to the most important and challenging part of HMM, *the evaluation of HMM*. In order to be able to use HMM for real cases, we need to know how to solve the three most important problems that arise when using HMM, namely the evaluation problem, optimization problem, and predicting the best sequence. Fortunately, there are one or more solutions to each problem.

Evaluation problem: Given a set of observation $V = \{V_1, V_2, \dots, V_T\}$, and the parameters of model $\lambda = [A, B, \pi]$, how to estimate the probability of observation sequence $P(V|\lambda)$?

The efficient solution to this problem is the first step toward optimization problems and predicting the best sequence problem. The solution to this problem can be obtained in two steps; first, we have to find all possible state sequences that can produce this observation sequence, and the second step is, from all possible state sequences, to find that state sequence that most likely generates this observation sequence.

In the section Forward-Backward algorithm, we will discuss in detail which problems will arise by performing these two steps and how we can escape them.

Predicting best sequence: Given a set of observations $V = \{V_1, V_2, \dots, V_T\}$, and the parameters of model $\lambda = [A, B, \pi]$, how to find the most probable (hidden) state sequence, that is supposed to produce this observation set.

To find the best or most likely state sequence (also known as decoding problem) Viterbi algorithm or Viterbi decoder can be used. The Viterbi algorithm that is based on dynamic programming is used to get MAP (maximum a posteriori estimation) of the most likely state sequence or path.

Optimization problem: Given a set of observations $V = \{V_1, V_2, \dots, V_T\}$, how can we optimize or learn model parameters $\lambda = [A, B, \pi]$ to maximize $P(V|\lambda)$?

The optimization problem (also known as the learning problem) is the most difficult problem of the Hidden Markov Model. A forward-Backward algorithm can be used to solve the optimization problem, but the most widely used and much more efficient algorithm is the Baum-Welch algorithm, which is a special case of the Expectation-Maximization algorithm. Based on observation V , the Baum-Welch algorithm optimizes a given model λ in such a manner that the optimized model generates the training set with equal or higher probability (Fink 2014).

Later we will see that the three problems are closely related in our probabilistic framework.

3.2.3 Evaluation problem (Forward - Backward algorithm)

As mentioned earlier, the easiest way to solve evaluation problem (the first step) is to enumerate every possible sequence of states of length T , where T is the number or length of observations. Consider such a fixed sequence of states (Rabiner 1989):

$$Q = q_1, q_2, \dots, q_T$$

as we already know, q_1 is the initial state. Then the probability of observation sequence V for this state sequence Q is (Rabiner 1989):

$$P(V | Q, \lambda) = \prod_{t=1}^T P(V_t | q_t, \lambda) \quad (\text{Eq. 4a})$$

among assumption that observations are statistical independence, we will get (Rabiner 1989):

$$P(V | Q, \lambda) = b_{q_1}(V_1) \cdot b_{q_2}(V_2) \cdot \dots \cdot b_{q_T}(V_T) \quad (\text{Eq. 4b})$$

and the probability of state sequence Q can be written as (Rabiner 1989):

$$P(Q | \lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T} \quad (\text{Eq. 5})$$

and the probability that observation sequence V and state sequence Q follow at the same time, or in other words, the joint probability of V and Q is the product of (4b) and (5) (Rabiner 1989):

$$P(V, Q | \lambda) = P(V | Q, \lambda) P(Q, \lambda) \quad (\text{Eq. 6})$$

and finally, the probability of observation sequence V can be received by summing the joint probability over all possible state sequence q (Rabiner 1989):

$$P(V | \lambda) = \sum_{\text{all } Q} P(V | Q, \lambda) P(Q | \lambda) \quad (\text{Eq. 7a})$$

$$P(V | \lambda) = \sum_{q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(V_1) a_{q_1 q_2} b_{q_2}(V_2) \dots a_{q_{T-1} q_T} b_{q_T}(V_T) \quad (\text{Eq. 7b})$$

The equation (Eq. 7b) means that at time $t = 1$ the system is in the initial state q_1 with the probability π_{q_1} which emits the observation V_1 with the probability $b_{q_1}(V_1)$, and in the next time step $t+1$ system moves to the state q_2 with the transition probability $a_{q_1 q_2}$ which emits the observation V_2 with the probability $b_{q_2}(V_2)$. The system continues moving in the same way until it reaches the transition from q_{T-1} to q_T with the transition probability $a_{q_{T-1} q_T}$, which emits the observation V_T with the probability $b_{q_T}(V_T)$.

A brief but close look at the equations makes us clear if N possible states (at every $t = 1, 2, \dots, T$) can be reached, then N^T state sequences are possible. And for each term in the sum of

equation (Eq. 7b), $2T$ calculations are needed to solve each state sequence, which means $2T \cdot N^T$ calculations are required. It makes this method computationally impracticable, for example, if we consider a small system with only five states ($N = 5$) and 100 observations ($T = 100$), then we will have $2 * 100 * 5^{100} = 10^{72}$ calculations, which will be even higher for real cases. Therefore, even if we have this calculation method, another method with higher efficiency is needed to overcome the evaluation problem. As mentioned, fortunately, there are solutions namely *forward* and *backward algorithms*. We will see later how the forward algorithm can be used for solving of (Eq. 7b) or rather evaluation problem to overcome this large computation problem.

Forward and backward algorithms are perfect examples of dynamic programming¹. These two recursive² dynamic programming approaches help us by solving evaluation problem and optimization problem. The dynamic programming was used for type of optimization of problems, or more generally, it can be used for making inferences.

3.2.3.1 Forward algorithm

In order to be able to solve the first problem, namely the evaluation problem, or in other words, to estimate the probability of observation sequence $P(V|\lambda)$ given model parameters, we can use the forward algorithm. In the forward algorithm, the already calculated probability of the current state at time step t will be used to calculate the probability of the next state at time step $t+1$, mathematically defined as (Rabiner 1989):

$$\alpha_t(i) = P(V_{1:t}, q_t = S_i | \lambda) \quad (\text{Eq. 8})$$

the α is the forward variable. The initialization for the first step at time $t = 1$ is as follows (Rabiner 1989):

$$\alpha_1(i) = \pi_i b_i(V_1) \quad (\text{Eq. 9})$$

with $1 \leq i \leq N$. It is the joint probability of state S_i and the first observation V_1 . But the generalized equation is the main part (induction) of forward algorithm which is for all other steps (Rabiner 1989):

$$\alpha_{t+1}(j) = b_j(V_{t+1}) \sum_{i=1}^N \alpha_t(i) a_{ij} \quad (\text{Eq. 10})$$

with $1 \leq j \leq N$ and with $1 \leq t \leq T-1$.

¹ Dynamic programming is a programming model, where complex problems are divided into smaller sub-problems with systematically storing intermediate results.

² Recursive means that we can reuse prior values as input to obtain next values.

We utilize Trellis Diagram to get the idea behind the Forward Algorithm. A lattice form of states and observation is shown in figure 7.

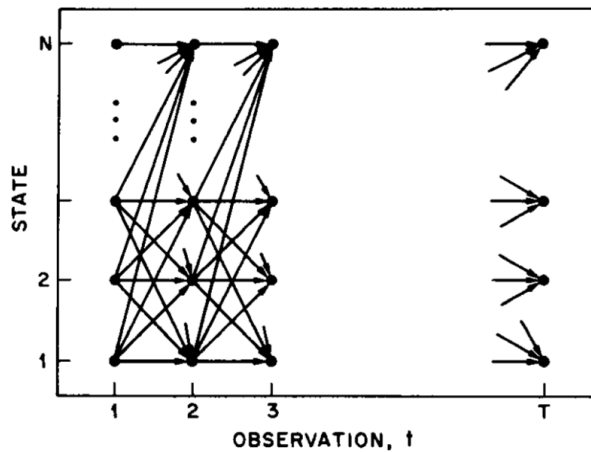


Figure 7: Observations vs states in form of a lattice (Rabiner 1989)

Look at the Trellis diagram shown in figure 8 and assume that the likelihood of the system being in (hidden) state S_1 at time $t-1$ is $\alpha_{t-1}(i) = \alpha_{t-1}(S_1)$. At time step t , the probability of transitioning to (hidden) state $S_j = S_2$ may now be expressed as: $\alpha_t(j = 2) = \alpha_{t-1}(S_1) a_{12}$. Similarly, adding all the probabilities of the system transitioning to state $S_{j=2}$ at time t from any state at time $t-1$ yields the total probability of a transition from any state at $t-1$ to $S_{j=2}$ at time step t , mathematically expressed as: $\sum_{i=1}^N \alpha_{t-1}(i) a_{ij}$.

Then finally, we can say the likelihood that the system is at state $S_{j=2}$ at time t after sending out first t number of visible observations from sequence V_T is provided by (simply multiply the emission probability to the above formula): $b_{qt}(V_t) \sum_{i=1}^N \alpha_{t-1}(i) a_{ij}$

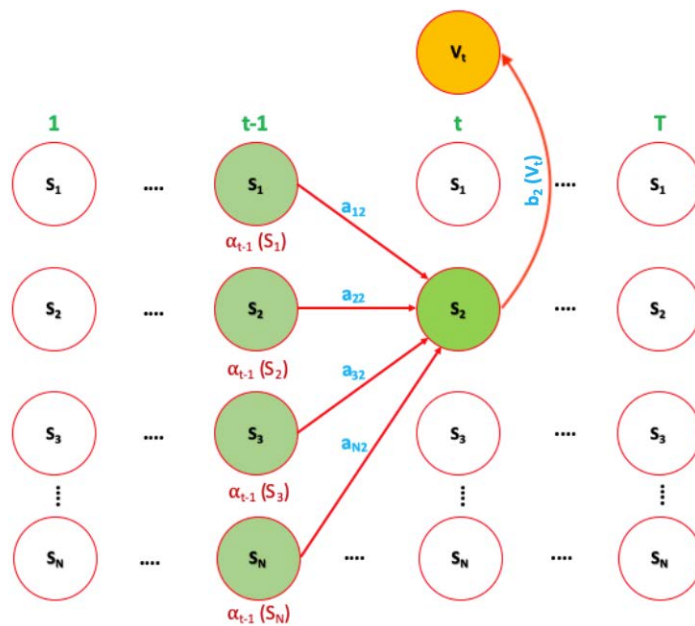


Figure 8: Trellis diagram for forward algorithm

Now we can expand this to a recursive algorithm to find the likelihood that series V_T was produced by the model λ . Mathematically can be expressed as follow:

$$\alpha_{t+1}(j) = \begin{cases} \pi_i b_i(V_1) & \text{for } t = 1 \\ b_j(V_{t+1}) \sum_{i=1}^N \alpha_t(i) a_{ij} & \text{for } t > 1 \end{cases}$$

The $\alpha_{t+1}(j)$ is the probability that the system will be at state S_j at time step $t+1$ after sending out first t visible series of observations.

Finally, we come to the termination part of forward algorithm, which gives the desired calculation of evaluation problem $P(V|\lambda)$, which is the sum of all $\alpha_T(i)$ (Rabiner 1989):

$$P(V | \lambda) = \sum_{i=1}^N \alpha_T(i) \quad (\text{Eq. 11})$$

As mentioned before, by using the direct calculation method, for a small system with only five states ($N = 5$) and 100 observations ($T = 100$), around 10^{72} computations were required. Now when using the forward variable $\alpha_t(i)$, N^2T calculations are needed, which are in this case about 2500 computations, clearly fewer calculations.

Now we can go through a practical example to make it clearer, how can we obtain α and also the probability of observation sequence $P(V|\lambda)$ by using Forward algorithms with the programming language Python. Remember the previous example with the man and his daughter again. As you may notice, in order to be able to use the forward algorithm, we assume that the model parameter λ (a , b and π) and the possible states, possible observations, and the observation sequence V (observed moods) are already known, as shown in figure 9. A brief description of assumptions, inputs, outputs and also the purpose of the forward algorithm is shown in figure 10. As a reminder, the first requirement (which is valid for all algorithms) is that the number of possible observation M and observation sequence V are known to us. The second requirement (here, and not for all algorithms) is that the number of possible states N is known, otherwise, first N has to find out using other algorithms or methods, such as Gaussian Mixture.

```
# Possible States and Observations
States = ["Sun", "Cloud", "Rain"]
Observation = ["Happy", "Sad"]

# Observation sequence V, i.e. Moods observed by father
V = ['Happy', 'Happy', 'Sad', 'Happy', 'Sad', 'Sad', 'Happy']

# Transition Probabilities
a = np.array(((0.80, 0.15, 0.05),
              (0.25, 0.40, 0.35),
              (0.20, 0.30, 0.50)))

# Emission Probabilities
b = np.array(((0.80, 0.20),
              (0.50, 0.50),
              (0.25, 0.75)))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.33, 0.33, 0.33))
```

Figure 9: Model parameters, states, observations and observed sequence

```

"""
Hidden Markov Model: Forward Algorithm
Assumptions:
Model parameters (a, b and  $\pi$ ) are known
Goal:
Calculation of the joint distribution of a given (hidden) state  $q(t)$ 
based on the observations  $V(1)$  through  $V(t)$ 
Input Parameters:
Observation Sequence V: 1-D array
Transition Matrix a: 2-D array
Emission Matrix b: 2-D array
Initial Distribution: 1-D array
-----
Output:
Forward Variable Alpha: 2-D array
"""

```

Figure 10: Assumptions, inputs and outputs of forward algorithm

Using the following Python code for forward algorithm (figure 11),

```

def forward(V, a, b, initial_distribution):
    alpha = np.zeros((V.shape[0], a.shape[0]))
    alpha[0, :] = initial_distribution * b[:, V[0]]

    for t in range(1, V.shape[0]):
        for j in range(a.shape[0]):
            # Matrix Computation Steps
            # ((1x2) . (1x2)) * (1)
            # (1) * (1)
            alpha[t, j] = alpha[t - 1].dot(a[:, j]) * b[j, V[t]]

    return alpha

```

Figure 11: Python code for forward algorithm (A Developer Diary o. J.)

The outputs are as follows (figure 12):

```

Alpha at each time step:
      Sun      Cloud      Rain
Time step 1: 0.264000 0.165000 0.082500
Time step 2: 0.215160 0.065175 0.028050
Time step 3: 0.038806 0.033380 0.035696
Time step 4: 0.037223 0.014941 0.007868
Time step 5: 0.007017 0.006960 0.008268
Time step 6: 0.001802 0.003159 0.005191
Time step 7: 0.002615 0.001545 0.000948

Probability of this observation sequence  $P(V|\lambda)$ :
0.005108

```

Figure 12: Alpha for each state at each time step t , and probability of obs. sequence $P(V|\lambda)$

Once again, to remember, the α 's are the probability of being in a particular state at a particular time. In other words, they are intermediate results.

As you can see in figure 12, the rows stand for the time steps and the columns for three possible states Sun, Cloud and Rain. We can easily see what the probability that we will be in one of the three states at a certain time is. Now, if we look at the termination equation for the Forward algorithm (Eq. 11), the probability of observation sequence $P(V|\lambda)$ is the sum of all α 's of the last step (marked in red in the picture), which is **0.005108** ($\approx 0.51\%$). And thus, the first problem is solved using the Forward algorithm.

If we take a closer look at the probabilities at each step and take the state with the higher probability for each time step, we get the state sequence that this observation sequence may have emitted, States = [Sun, Sun, Sun, Sun, Rain, Rain, Sun] (figure 18 & 25), but that is a task of Viterbi's algorithm that will be discussed in more detail later.

Just to get an overview, let's calculate two examples of the outputs of Alpha. The first line of Alpha is the first-time step, means the initial probability $\alpha_1(i) = \pi_i b_i(V_1)$. For example, from observation sequence, we know that V_1 is Happy, if we consider the first (hidden) state as sunny, the forward variable α will be (with the probability $P(q_1 = S_1 | V_1 = O_1)$):

$$\alpha_1(\text{sun}) = \pi_{\text{sun}} b_{\text{sun}}(\text{Happy})$$

$$\alpha_1(\text{sun}) = 0.33 * 0.80 = \mathbf{0.264}$$

Now let's go to the second time step if the (hidden) state will be cloudy, where the observation is again Happy (that means $P(q_2=S_2 | V_2=O_1)$). From there on we have to consider transition probability as well, therefore:

$$\alpha_{t+1}(j) = b_j(V_{t+1}) \sum_{i=1}^N \alpha_t(i) a_{ij}$$

$$\alpha_2(\text{Cloud}) = b_{\text{cloud}}(\text{Happy}) * [(\alpha_1(\text{Sun}) * a_{\text{Sun,cloud}}) + (\alpha_1(\text{Cloud}) * a_{\text{cloud,cloud}}) + (\alpha_1(\text{Rain}) * a_{\text{Rain,cloud}})]$$

$$\alpha_2(\text{Cloud}) = 0.50 * [(0.264 * 0.15) + (0.165 * 0.4) + (0.0825 * 0.3)] = \mathbf{0.065175}$$

3.2.3.2 Backward algorithm

It's important to know that the backward algorithm will actually be used as part of the solution for the optimization problem and not the evaluation problem, but since the concept of the forward algorithm has already been explained here, it's easier to understand the backward algorithm.

The backward algorithm is very similar to the forward algorithm, but in contrast to the forward algorithm, the backward algorithm is time-reversed, which means we have to find the probability that the system will be in the state S_i at a given time step and will create the remaining part of the set of visible observations V_T . In other words, the probability of that part of the observation sequence from time step $t+1$ up to the end of observations, where state S_i at time step t and the model parameter λ are known. In a similar way to the forward variable α , the backward variable $\beta_t(i)$ can be defined as (Rabiner 1989):

$$\beta_t(i) = P(V_{t+1}, V_{t+2}, \dots, V_T | q_t = S_i, \lambda) \quad (\text{Eq. 12})$$

The initialization is as follows, where $\beta_T(i)$ will be voluntary 1 for all i (Rabiner 1989):

$$\beta_T(i) = 1 \quad (\text{Eq. 13})$$

with $1 \leq i \leq N$. The generalized equation or induction part of backward algorithm is as follows (Rabiner 1989):

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(V_{t+1}) \beta_{t+1}(j) \tag{Eq. 14}$$

with $1 \leq j \leq N$ and with $t = T-1, T-2, \dots, 1$. The trellis diagram illustrated in figure 13 shows the series of operations needed for calculation of backward variable $\beta_t(i)$.

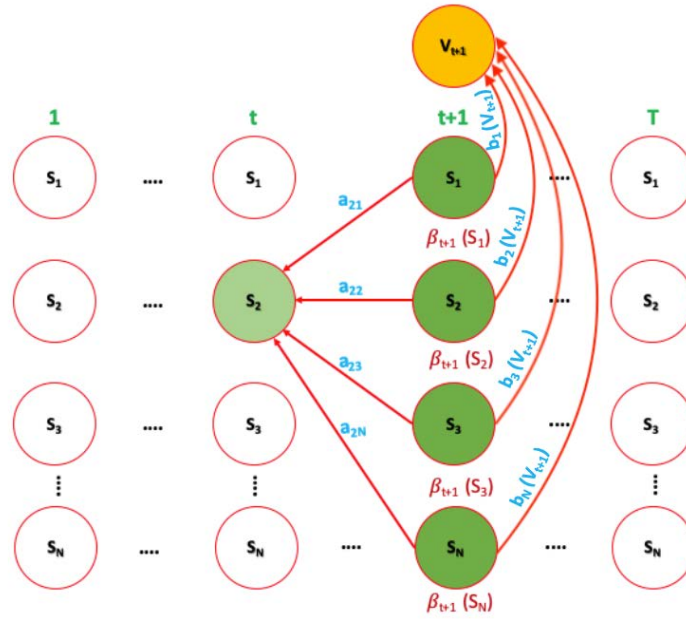


Figure 13: Trellis diagram for backward algorithm

Backward algorithm can be mathematically expressed as follow:

$$\beta_t(i) = \begin{cases} 1 & \text{for } t = T \\ \sum_{j=1}^N a_{ij} b_j(V_{t+1}) \beta_{t+1}(j) & \text{for } t < T \end{cases}$$

In order to calculate $\beta_t(i)$, as was the case in the forward algorithm, N^2T calculations are needed.

Here we can also go through a practical example to make it clearer, how can we obtain β by using the backward algorithms with the programming language Python. We use again the previous example mentioned in the section forward algorithm with the man and his daughter. The model parameter λ (a , b and π) and the possible states, possible observations and the observation sequence V (observed moods) are shown in figure 9. A brief description of assumptions, inputs, outputs and also the purpose of the backward algorithm is shown in figure 14. As a reminder, the first requirement for all algorithms is that the number of possible observation M and observation sequence V are known to us. The second requirement is that the number of possible states N is known, otherwise, first we have to find out N using other algorithms or methods, such as Gaussian Mixture.

```

"""
Hidden Markov Model: Backward Algorithm
Assumptions:
Model parameters (a, b and  $\pi$ ) are known
Goal:
Calculation of the joint distribution of a given (hidden) state  $q(t)$ 
based on the observations  $V(t+1)$  through  $V(t)$ 
Input Parameters:
Observation Sequence V: 1-D array
Transition Matrix a: 2-D array
Emission Matrix b: 2-D array
-----
Output:
Backward Variable Beta: 2-D array
"""

```

Figure 14: Assumptions, inputs and outputs of Backward algorithm

Using the following Python code for backward algorithms (figure 15):

```

def backward(V, a, b):
    beta = np.zeros((V.shape[0], a.shape[0]))

    # setting beta(T) = 1
    beta[V.shape[0] - 1] = np.ones((a.shape[0]))

    # Loop in backward way from T-1 to 0
    # Due to python indexing the actual loop will be T-2 to 0
    for t in range(V.shape[0] - 2, -1, -1):
        for j in range(a.shape[0]):
            beta[t, j] = (beta[t + 1] * b[:, V[t + 1]]).dot(a[j, :])

    return beta

```

Figure 15: Python code for Backward algorithm (A Developer Diary o. J.)

The outputs are as follows (figure 16):

```

Beta at each time step:
          Sun      Cloud      Rain
Time step T-6:  0.010430  0.009761  0.009024
Time step T-5:  0.012934  0.024269  0.026513
Time step T-4:  0.046983  0.049229  0.045998
Time step T-3:  0.055639  0.127731  0.143487
Time step T-2:  0.169275  0.248063  0.265350
Time step T-1:  0.727500  0.487500  0.435000
Time step T:    1.000000  1.000000  1.000000

```

Figure 16: Beta for each state and each time step t (7 days)

Once again to remember, the β 's are the probability of being in a particular state at a particular time. In other words, they are intermediate results. It is if we suppose that we are at a certain time step in a certain state and want to find out the probability of observing all subsequent events from this state.

As you can see in figure 16, the rows stand for the time steps and the columns for three possible states Sun, Cloud and Rain.

Again, just to get an overview, let's calculate an example of the outputs of beta. In the case of backward algorithm and calculation of beta, consider the last line of beta (Timestep T). As

mentioned earlier, the initialization of $\beta_T(i)$ will be voluntary 1 for all i 's at time step T . Now let's calculate as an example the time step $T-1$, if the state will be sunny.

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(V_{t+1}) \beta_{t+1}(j)$$

$$\beta_{T-1}(Sun) = \left(a_{Sun,Sun} * b_{Sun}(Happy) * \beta_T(Sun) \right) + \left(a_{Cloud,Sun} * b_{Cloud}(Happy) * \beta_T(Cloud) \right) \\ + \left(a_{Sun,Rain} * b_{Rain}(Happy) * \beta_T(Rain) \right)$$

$$\beta_2(Sun) = (0.8 * 0.8 * 1) + (0.5 * 0.15 * 1) + (0.25 * 0.05 * 1) = \mathbf{0.7275}$$

Forward and backward algorithms can be briefly summarized as follows: first of all, it assumes that transition and emission probabilities and initial distribution are all known. Forward and backward algorithms are recursive functions, which means we can reuse prior values or outcomes as input to obtain the next values or outcomes, in this way, we save time. The aim of the Forward-Backward algorithm is computing the marginal probability of one of the (hidden) states given the observation sequence, which means $P(q_t | V_{1:T})$, where $V_{1:T}$ is the entire observation sequence. Forward-Backward-Algorithm can be subdivided into Forward-Algorithm and Backward-Algorithm. The aim of the Forward-Algorithm is to compute the joint distribution of a given (hidden) state q_t and V_1 through V_t , which means $P(q_t, V_{1:t})$, and as you can see, there is no conditioning here. Backward-Algorithm is used to compute the joint distribution of V_{t+1} to V_T given (hidden) state q_t , which means $P(V_{t+1:T} | q_t)$.

As mentioned, Forward-Backward-Algorithm assumes that model parameter λ is known and tries to compute the conditional distribution of the hidden state, where observations are given. Therefore $P(q_t | V_{1:T})$ is proportional to joint distribution by q_t and $V_{1:T}$, which means:

$$P(q_t | V_{1:T}) \propto P(q_t, V_{1:T})$$

We can also say that the $V = (V_{1:t}, V_{t+1:T})$ and rewrite the above equation as follow (Ankan and Panda 2018):

$$P(q_t | V_{1:T}) \propto P(q_t, V_{1:t}, V_{t+1:T})$$

and after applying chain rule and independence property, we get the following equation (Ankan and Panda 2018):

$$P(q_t | V_{1:T}) = P(V_{t+1:T} | q_t) P(q_t, V_{1:t}) \tag{Eq. 15}$$

Now, look at the (Eq. 15), the part $P(V_{t+1:T}|q_t)$ is backward algorithm, and $P(q_t, V_{1:t})$ is the forward algorithm. That means *Forward-Backward-Algorithm can be simply computed by multiplying the backward algorithm with the forward algorithm*. This is very helpful in solving optimization (or learning) problems.

Later we will see how the use of these two algorithms helps us to solve the other two problems.

3.2.4 Predicting best state sequence (Viterbi algorithm)

Actually, to solve the second problem (predicting the best state sequence), we could first find all the different hidden state scenarios for the given observation sequence and then try to identify the most likely one. But as the Viterbi algorithm is very similar to the forward algorithm, we get the same problem again, which means there is a large number of computations required. How to overcome this problem will be discussed in this section.

But before diving into the problem solving, let's mention an interesting difference between the Viterbi and Forward algorithms. The most important difference between these two algorithms is discussed below, but there is another and also an important difference between these two algorithms. Since the Forward algorithm tries to calculate $P(V|\lambda)$, it does not consider which state is at each step most probable, but Viterbi finds the most likely state in every iteration and then returns this state sequence through backtracking, and also returns the probability of produced state sequence based on the observation sequence, as illustrated in figure 18.

3.2.4.1 Viterbi algorithm

First, let's go through the mathematical understanding, and then again use this algorithm for the example with the weather for better understanding.

In contrast to the evaluation problem, where there was only one specific solution to the problem, fortunately, there are several solutions for the second problem, depending on the optimality criteria. But since there are several optimality criteria, the question is, which criteria lead best to the optimal state sequence. If we for example, take the optimality criterion that selects the states q_t , where they are individually most likely, the solutions for the second problem will be as follows (Rabiner 1989):

$$\gamma_t(i) = P(q_t = S_i | V, \lambda)$$

This is the probability of being in state S_i at time t , where observation sequence and model parameters are known. By using forward variable α backward variable β , and the normalization factor $P(V|\lambda)$ to make $\gamma_t(i)$ a probability measure, the following equation can be obtained (Rabiner 1989):

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{P(V|\lambda)} = \frac{\alpha_t(i) \beta_t(i)}{\sum_{i=1}^N \alpha_t(i) \beta_t(i)}$$

where $\alpha_t(i)$ stands for observation sequence $V_{1:t}$, and $\beta_t(i)$ for $V_{t+1:T}$. Now the individually most likely state q_t at time t can be obtained as (Rabiner 1989):

$$q_t = \operatorname{argmax}_{1 \leq i \leq N} [\gamma_t(i)]$$

with $1 \leq t \leq T$. But since it maximizes the expected number of correct states and also computes the most likely state at each time step t , can some problems with the resulting state sequence occur if there are transition states that are equal to zero ($a_{ij} = 0$). In other words, since this optimality criterion does not consider the entire state sequence but only each individual state, it could actually be impossible a transition from one of these selected states to the next one if $a_{ij} = 0$. To overcome this problem, a method is needed that considers more than one-time step

and state (for example, pairs of states or triples of states, etc.). The Viterbi algorithm can be used to get the best state sequence without having the problem already mentioned. Viterbi algorithm is based on dynamic programming and is very similar to the forward algorithm. It tries to find the best *path* or find the *single best state sequence*, which maximizes $P(Q | V, \lambda)$ or rather $P(Q, V | \lambda)$.

The first step of the Viterbi algorithm is the initialization (Rabiner 1989):

$$\begin{aligned}\chi_1(i) &= \pi_i b_i(V_1) \\ \psi_1(i) &= 0\end{aligned}\tag{Eq. 16}$$

with $1 \leq i \leq N$ and $\psi_t(j)$ as array for backtracking. The second step is the recursive part, with $1 \leq j \leq N$ and $2 \leq t \leq T$ (Rabiner 1989):

$$\begin{aligned}\chi_t(j) &= \max_{1 \leq i \leq N} [\chi_{t-1}(i) a_{ij}] b_i(V_t) \\ \psi_t(j) &= \operatorname{argmax}_{1 \leq i \leq N} [\chi_{t-1}(i) a_{ij}]\end{aligned}\tag{Eq. 17}$$

The third step is the termination (Rabiner 1989):

$$\begin{aligned}P^* &= \max_{1 \leq i \leq N} [\chi_T(i)] \\ q_T^* &= \operatorname{argmax}_{1 \leq i \leq N} [\chi_T(i)]\end{aligned}\tag{Eq. 18}$$

And finally, the last part is the path or state sequence backtracking (Rabiner 1989):

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$

with $t = T-1, T-2, \dots, 1$.

If we go back to the previous example with the weather and moods, based on observation sequence (see figure 9) and apply the Viterbi Algorithm, best state sequence is predicted and its probability is calculated as shown in figure 18. A brief description of assumptions, inputs, outputs and also the purpose of the Viterbi algorithm is shown in figure 17. As a reminder, the first requirement for all algorithms is that the number of possible observation M and observation sequence V are known to us. The second requirement is that the number of possible states N is known, otherwise, first we have to find out N using other algorithms or methods (will be explained later).

```

"""
Hidden Markov Model: Viterbi Algorithm
Assumptions:
-----
Alpha's from Forward algorithm are known
Goal:
-----
Obtain best state sequence, given observation sequence and model parameters.
Input Parameters:
-----
Observation Sequence V: 1-D array
Transition Matrix a:    2-D array
Emission Matrix b:     2-D array
Initial Distribution:  1-D array
-----
Outputs:
-----
The probability of the most probable state, Omega: 2-D array
Predicted (hidden) state :                        1-D array
"""

```

Figure 17: Assumptions, inputs and outputs of the Viterbi algorithm

```

Most probable state sequence & assoc. probabilities:
Pred. States Probabilities
Time step 1:      Sun      0.264
Time step 2:      Sun      0.2152
Time step 3:      Sun      0.0388
Time step 4:      Sun      0.0372
Time step 5:      Rain     0.0083
Time step 6:      Rain     0.0052
Time step 7:      Sun      0.0026

Probability of best state sequence:
9.21e-12

```

```
Predicted States using Viterbi: ['Sun', 'Sun', 'Sun', 'Sun', 'Rain', 'Rain', 'Sun']
```

Figure 18: Best state seq. and its probability predicted, using the Viterbi algorithm

The Viterbi algorithm returns the best state sequence and its probability ($9,21 \cdot 10^{-12}$), shown in figure 18, based on the given observation sequence. And thus, the second problem is solved using the Viterbi algorithm. It can be that for the real-life cases, due to the difference between the Viterbi and Forward algorithm mentioned above, different results will be returned.

In the next section (Baum-Welch algorithm), another example of the Viterbi algorithm is discussed together with Baum-Welch algorithm.

3.2.5 Baum - Welch algorithm

Up to here, we introduced algorithms, such as forward, backward or Viterbi, to make inferences over Hidden Markov Model. The first assumption was that the model parameters λ and the possible states are known to us. But in real-life, this is rarely the case; we have to find these parameters and values from the data. In this part, we will see which algorithms are available and how we can do this.

As mentioned earlier, optimization or learning problem is the most difficult problem of HMM. There is currently no known technique to solve for the model that optimizes the probability of the observation sequence analytically. Actually, there is no optimal technique of estimating model parameters given any finite observation sequence as training data (Rabiner 1989). Therefore, what we can do is, applying an iterative approach that selects model parameter λ in such a manner that maximizes locally $P(O|\lambda)$. Baum-Welch algorithm, which is a special

case of EM (Expectation Maximization), is the most widely used technique for this purpose. Other methods for this purpose are also available, methods such as *Maximum Likelihood Estimation (MLE)* or *Viterbi Learning Algorithm*¹, but the Baum-Welch algorithm (also known as *Forward-Backward-Algorithm*)² is more efficient and best suited for HMM. Based on observation V , the Baum-Welch algorithm optimizes a given model λ in such a manner that the optimized model generates the training set with equal or higher probability (Fink 2014).

These two main methods can be used to solve the Baum-Welch algorithm. The first method is *Lagrange multipliers*, which constrains optimization problems to find model parameter λ (a, b and π) by maximizing $P(V|\lambda)$. The second method is *the probabilistic approach*, which will be discussed in detail here.

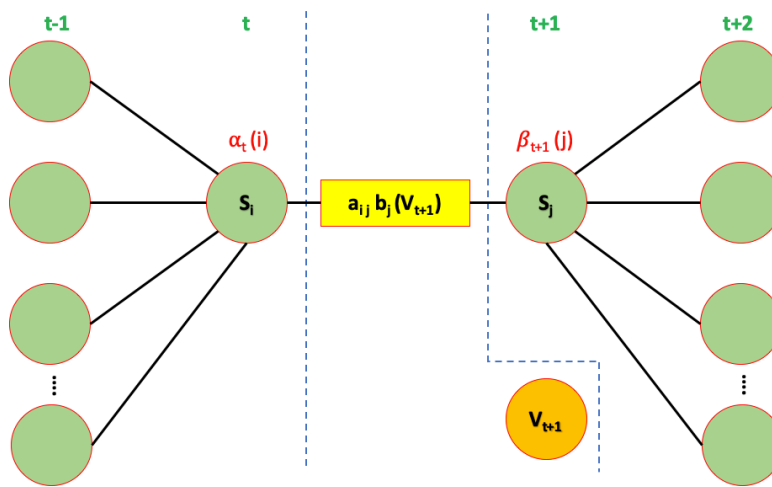


Figure 19: Operations sequence needed to calculate joint event by a transition from S_i to S_j

First, we define $\xi_t(i, j)$ to re-estimate model parameters as follow (Rabiner 1989):

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j | V, \lambda) \quad (\text{Eq. 19})$$

that means the probability of being in state S_i at time state t and S_j at time step $t+1$, as shown in figure 19, where the model parameter λ and observation sequence are known.

We know the following from probability theory (A Developer Diary o. J.):

$$\begin{aligned} P(A, B | C) &= P(A | B, C) P(B | C) \\ \Rightarrow P(A | B, C) &= \frac{P(A, B | C)}{P(B | C)} \end{aligned}$$

Knowing this probability theory and also forward and backward variables α and β , now we can rewrite the (Eq. 19) as follow (Rabiner 1989):

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(V_{t+1}) \beta_{t+1}(j)}{P(V | \lambda)} \quad (\text{Eq. 20})$$

¹ Do not be confused with Viterbi algorithm, which is for prediction best (hidden) state sequence.

² Due to use of Forward-Backward-Algorithm in expectation step (E-Step).

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(V_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(V_{t+1}) \beta_{t+1}(j)}$$

We already know (from section Viterbi algorithm) that $\gamma_t(i)$ is the probability being in state S_i at time t , where observation sequence and model parameters are known. If we relate $\gamma_t(i)$ to $\xi_t(i, j)$ by summing over j , following equation can be obtained (Rabiner 1989):

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j)$$

Now if we sum $\gamma_t(i)$ over time t , we obtain a quantity that can be explained as the expected number of times that a certain state S_i occurs. Likewise, if we sum $\xi_t(i, j)$ over time t ($t_1: T-1$), it can be explained as the expected number of times that transition from S_i to S_j occurs. That can be written as (Rabiner 1989):

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transition from } S_i$$

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{expected number of transition from } S_i \text{ to } S_j$$

Considering these equations and also the concept of counting event occurrences, the following set of re-estimation equations for model parameters a , b and π can be defined (Rabiner 1989):

$$\bar{\pi} = \text{expected frequency (number of times) in state } S_i \text{ at time } (t = 1)$$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } V_k}{\text{expected number of times in state } j}$$

with $1 \leq k \leq M$. Mathematically expressed as (Rabiner 1989):

$$\bar{\pi} = \gamma_1(i)$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \quad \text{s.t. } V_t = O_k$$

Earlier mentioned that based on observation V , the Baum-Welch algorithm optimizes a given model λ in such a manner that the optimized model generates the training set with equal or higher probability. That means if λ (a , b and π) is an initial model parameter, which calculates the right-hand sides of the above equations, and $\bar{\lambda}$ (\bar{a} , \bar{b} and $\bar{\pi}$) is a re-estimated model

parameter that is calculated by the initial model parameter λ , then these two scenarios are possible, either in worst case $\bar{\lambda} = \lambda$, or the probability of newfound model is higher than initial model parameters, $P(V | \bar{\lambda}) > P(V | \lambda)$, where the observation sequence is more likely to have been produced.

If we utilize this method, and we iteratively use $\bar{\lambda}$ instead of λ and repeat the re-estimating computation until it converges, thus we obtain an improvement of the probability of observation sequence V . As already noticed, this re-estimating method can be interpreted as an implementation of EM algorithm, where the EM iteration changes between an Expectation step (E) that calculates the auxiliary function $Q(\lambda, \bar{\lambda}) = \sum_Q P(Q | V, \lambda) \log[P(V, Q | \bar{\lambda})]$, and a maximization step (M) that is a maximization over $\bar{\lambda}$. The procedure is as follows:

EM algorithm:

- **Initialize** a and b
- **Iteration** (until convergence)
 - **E-Step**
 - $\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(V_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(V_{t+1}) \beta_{t+1}(j)}$
 - $\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j)$
 - **M-Step**
 - $\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$
 - $\bar{b}_j(k) = \frac{\sum_{t=1}^T \sum_{s.t. V_t=O_k} \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$
- **Return** a, b

Again, to get a better insight into Baum-Welch algorithm and also the Viterbi algorithm, let's go back to the example with the weather and mood. As already mentioned at the beginning of this section, if the model parameters λ (A, B and π), and the number of possible states are known to us, we can use the forward algorithm and Viterbi algorithm to obtain the $P(V|\lambda)$ and the most probable state sequence. If it is not the case, that means model parameters λ and/ or possible states are unknown, and we have to find these parameters and values from the data. We can use Gaussian Mixture, Bayesian Information Criterion (BIC)¹ or Akaike Information Criterion (AIC)² (will be discussed in detail in chapter 6) etc. to get an assessment of the possible states. But if only the model parameters are missing, we can use the Baum-Welch algorithm to estimate these parameters.

Since we have not still talked about Gaussian Mixture etc., in this example, we assume that the possible states are known (Sun, Cloud and Rain). Now the first step is initializing the model parameters. As illustrated in figure 20, we set equal transition probabilities (33%) and also equal emission probabilities (50%) for all states, but we use different values for initial probability

¹ https://en.wikipedia.org/wiki/Bayesian_information_criterion

² https://en.wikipedia.org/wiki/Akaike_information_criterion

(50%, 30% and 20%) to avoid all values being the same for each state. Notice, that we could also use the model parameters that we used up to here for Forward-, Backward- and Viterbi algorithm, but to demonstrate that it is not necessary at all, equal initial- and emission probabilities will be used.

```
# Possible States and Observations
States = ["Sun", "Cloud", "Rain"]
Observation = ["Happy", "Sad"]

# Observation sequence V, i.e. Moods observed by father
V = ['Happy', 'Happy', 'Sad', 'Happy', 'Sad', 'Sad', 'Happy']

# Equal transition Probabilities
a = np.array(((0.33, 0.33, 0.33),
              (0.33, 0.33, 0.33),
              (0.33, 0.33, 0.33)))

# Emission Probabilities
b = np.array(((0.50, 0.50),
              (0.50, 0.50),
              (0.50, 0.50)))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.50, 0.30, 0.20))
```

Figure 20: States, observations and model parameters a , b and π

The transition and emission probabilities (a & b) are estimated applying Baum-Welch algorithm (see Appendix A). A comparison with initial transition and emission probabilities is shown in figures 22 & 23. A brief description about assumptions, inputs, outputs and also the purpose of the Baum-Welch algorithm is shown in figure 21. As a reminder, the first requirement for all algorithms is that the number of possible observations M and observation sequence V are known to us. The second requirement is that the number of possible states N is known, otherwise first we have to find out N using other algorithms or methods, such as Gaussian Mixture, BIC or AIC.

```
"""
Hidden Markov Model: Baum-Welch Algorithm
-----
Assumptions:

Number of possible states N is known
Initialization of Model parameters (a, b and  $\pi$ ) is done
-----
Goal:

Optimization of Model parameters (a, b)
-----
Input Parameters:

Observation Sequence V: 1-D array
Transition Matrix a:    2-D array
Emission Matrix b:     2-D array
Initial Distribution:   1-D array
Number of iterations:  integer
-----
Output:

Model parameter a:     2-D array
Model parameter b:     2-D array
"""
```

Figure 21: Assumptions, inputs and outputs of the Baum-Welch algorithm


```

a optimized by Baum-Welch:
      Sun   Cloud   Rain
Sun   0.0204  0.8344  0.1452
Cloud 0.0023  0.0900  0.9077
Rain  0.2568  0.4376  0.3056

b optimized by Baum-Welch:
      Happy   Sad
Sun   0.9987  0.0013
Cloud 0.8831  0.1169
Rain  0.0560  0.9440

```

Figure 22: a & b estimated by Baum-Welch

```

a initialized:
      Sun   Cloud   Rain
Sun   0.3333  0.3333  0.3333
Cloud 0.3333  0.3333  0.3333
Rain  0.3333  0.3333  0.3333

b initialized:
      Happy   Sad
Sun   0.5  0.5
Cloud 0.5  0.5
Rain  0.5  0.5

```

Figure 23: a & b initial

Since we got the missing transition and emission probabilities, based on this new information we can again apply the Viterbi algorithm to obtain the most probable (hidden) state sequence. The predicted hidden state sequence using a & b estimated by Baum-Welch and a comparison with initial transition and emission probabilities (which we used up to here by implementing Forward-, Backward- and Viterbi algorithm) is shown in figure 24 & 25.

```

Predicted States using a and b from Baum-Welch:
['Sun', 'Cloud', 'Rain', 'Cloud', 'Rain', 'Rain', 'Cloud']

```

Figure 24: Predicted (hidden) states using a & b from Baum-Welch algorithm

```

Predicted States using initial a & b:
['Sun', 'Sun', 'Sun', 'Sun', 'Rain', 'Rain', 'Sun']

```

Figure 25: Predicted (hidden) states using a & b initial

Compared to the predicted state with the initial transition and emission probabilities, some small changes can be noticed.

3.3 Tools available

In previous sections, after we discussed some basic terms about statistics, we talked about the theory of HMM. We also discussed parameterizing of HMM (initial distribution, transition and emission probabilities) and evaluation of HMM (evaluation problem, optimization problem and prediction of best sequence) in detail. We have seen that at least one solution is available for each problem. In this section, we will see how we can use this information to define and build a model that will solve a given problem.

Of course, we can use a programming language to generate our own algorithms and functions required for the model, as was the case up to here. But it is usually not necessary because there are many different applications, libraries and tools available that provide the ability to implement Hidden Markov Models. Depending on the given problem, each tool varies in the level of expertise needed to utilize them and also the functions available to the users. Since almost every programming language has tools or packages for HMM, we obviously can't discuss them all here, but below are some very helpful tools and packages.

In MATLAB¹, all essential function and algorithms (Like Forward, Backward, Viterbi and Baum-Welch etc.) are available to implement HMM. The main functions related to HMM are available, without installing of tools or packages were needed. But as mentioned, depending on the given problem, other functions probably were needed, therefore some additional packages and tools have been provided and added to MATLAB. For example, *Hidden Markov Model Toolbox (HMM)*² is one of them, which contains functions that model time series data with HMM.

In R, *HMM* package, which contains the four essential algorithms (Forward, Backward, Viterbi and Baum-Welch), is available to implement HMM. Another important and useful package in R is *depmixS4-package*. The depmixS4 is a framework for specifying and fitting dependent mixture models, otherwise known as hidden or latent Markov models. Optimization is done with the EM algorithm or optionally with Rdonlp2 when (general linear (in-)equality) constraints on the parameters need to be incorporated. Models can be fitted on (multiple) sets of observations. The response densities for each state may be chosen from the GLM family or a multinomial. User-defined response densities are easy to add; for the latter, an example is given for the ex-gauss distribution as well as the multivariate normal distribution. Mixture or latent class (regression) models can also be fitted; these are the limit case in which the length of observed time series is 1 for all cases (Visser and Speekenbrink o. J.).

In Python, the *hmmlearn* package is available to implement HMM. From now on, we will discuss this package from Python and how it can be used to implement HMM.

There are three models available, using *hmmlearn*³:

- *Gaussian Model (hmm.GaussianHMM)*: HMM with Gaussian emissions.
- *Gaussian Mixture Model (hmm.GMMHMM)*: HMM with Gaussian mixture emissions.
- *Multinomial Model (hmm.MultinomialHMM)*: HMM with Multinomial (discrete) emissions.

Gaussian- and Gaussian Mixture Models are used for continuous data, for example, time-series data. A multinomial model is used for fixed or finite number of variables. For instance, the example with the weather can be solved using the Multinomial model. Since in this master thesis HMM is used to analyze time-series data (continuous data), we will only deal with (normal) Gaussian and Gaussian Mixture from now on.

By applying one of these models, we can solve the three main problems of the hidden Markov model. We can call *the fit ()* method to train and estimate model parameters (optimization problem) and after that, inferring the hidden states by calling the *decode ()* or *predict ()* method.

¹ <https://de.mathworks.com/help/stats/hidden-markov-models-hmm.html>

² Mo Chen (2021). Hidden Markov Model Toolbox (HMM) MATLAB Central File Exchange. Retrieved October 9, 2021 (<https://www.mathworks.com/matlabcentral/fileexchange/55866-hidden-markov-model-toolbox-hmm>)

³ There is also *hmmlearn.base* model available, if you want to implement a custom emission probability (e.g. Poisson), you have to subclass `_BaseHMM` and override some methods.

The decode method can be specified with a decoder algorithm, Viterbi or MAP, to solve the second problem (predicting the best state sequence). Or calling *score ()* method to estimate the probability of observation sequence $P(V|\lambda)$, which means the evaluation problem (hmmlearn 2010). These are just some examples to get an insight into these three models. In the next section, we will discuss how Gaussian mixture (and/or normal Gaussian) can be used to predict hidden states in time series data of sucker rod pumps.

In this work, the results from HMM will be compared with other clustering methods, namely the Gaussian Mixture and K-Means. Therefore, in addition to the package *hmmlearn* the *scikitlearn* package is required, which is also a machine learning library for Python programming language, to implement Gaussian Mixture and K-Means. Notice that in this work, Gaussian Mixture HMM (GMMHMM) is used, which is based on Gaussian Mixture. The difference between Gaussian Mixture and Gaussian Mixture HMM is that Gaussian Mixture HMM uses the four main algorithms (Forward, Backward, Viterbi and Baum-Welch) to solve the problems.

4 Sucker rod pump analysis using machine learning

4.1 Sucker rod pumping system

The artificial lift market will experience substantial growth in the forthcoming decade, where sucker rod pumps will take a significant share. Sucker rod pumping systems represent the oldest and most widely used artificial lift method, with several hundreds of thousands of units worldwide. Their application ranges from lifting oil in mature conventional oil fields, production in stripper wells and unconventional oil fields, to unloading gas wells. Stripper wells represent a high percentage of vertical oil wells and produce less than 10 barrels per day. Sucker rod pumps (SRP) are well known for their flexibility to match the well capacity during the natural decline in production; they can be used to maximize the drawdown in the well, have high efficiency, and a relatively simple design. The system can lift high-temperature and viscous oils. Scale and corrosion treatments can be performed efficiently during the operation.

The sucker rod pumping system is composed of the surface unit, the sucker rod string, and the downhole pump. The surface unit, called the pump jack, represents the drive of the system. Figure 26 presents the schematics of the conventional pump jack design.

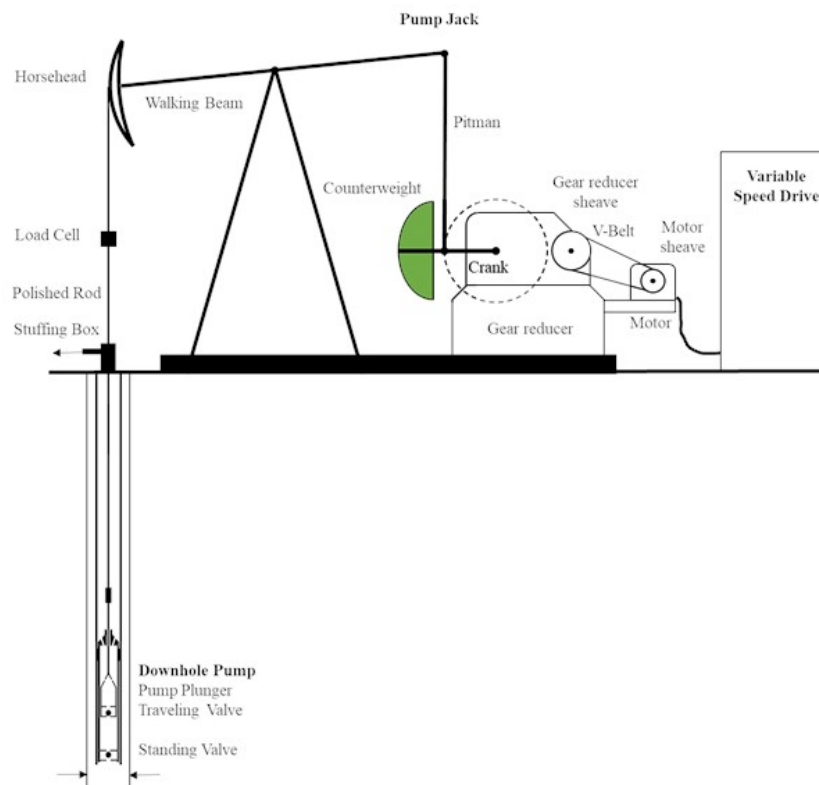


Figure 26: Sucker rod pumping system (Langbauer et al. o. J.)

The first section of the pump jack reduces the rotation speed, whereas the second transforms the rotation into translation. The electrical motor's sheave drives the V-belt, which drives the gear reducer sheave and represents the first stage of speed reduction. The gear reducer sheave is positioned at the two-stage gear reducer input shaft, which further reduces the rotation speed. At the output gearbox shaft, the final speed is obtained. This transmission system typically reduces a motor speed of 1000 rpm by a factor of about 150 to 200. Finally,

a system speed in the range between 3 and 15 rpm is reached. The cranks carry the counterweight, which balances the rod string's weight and is connected with the pitman. The pitman is connected to one end of the walking beam. On the other end of the walking beam, the horsehead is positioned. The horsehead's wireline hangers carry the polished rod, which passes through the wellhead's stuffing box. The polished rod is connected to the sucker rod string, which transmits the pump jack's motion to the downhole pump plunger.

During the upstroke of the downhole pump's plunger, the traveling valve as part of the plunger is closed, the fluid load is carried by the rod string and lifted. Simultaneously, the standing valve is opened to allow inflow into the pump's intake chamber as part of the fixed barrel. During the plunger's downstroke, the standing valve, which carries the fluid load now, is closed, and the plunger moves through the fluid column back to its bottom dead center. The cyclic load changes cause dynamics in the pumping system, which need to be handled by the pump jack. Continuous research on the optimization and improvement of surface and downhole components of the pumping system is performed in the laboratory and the field.

The electric motor can be driven directly from the electric grid or by a frequency converter. The electric grid's direct connection is a cheap solution but requires additional hardware, a so-called soft start, and a change in the belt pulley sizes to adjust the pump's strokes per minute. Using frequency converters (FC) for industrial pump applications is state-of-the-art and provides many advantages, like controlling the speed of rotation and torque of the equipment and reducing the grid load at start-ups through limiting the inrush current. Many publications deal with FC and electric motor combinations for specific applications, increasing efficiency or power density, enhancing control strategies, or extending the power capacity. A sucker rod pumping system driven by a frequency converter is known as a variable speed drive system (VSD). Conventional VSD applications change the motor speed independent of the supply grid frequency. A high degree of flexibility is achieved to adjust the pump speed to the reservoir's performance.

Pumping unit variable-speed drive technology is seen as one of the most advanced energy-saving technologies applied in oilfields. FCs overcome the pump jack's limitation of the four-bar mechanism to achieve a full-cycle variable speed-controlled operation (Langbauer et al. o. J.).

4.2 Machine learning

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data, in this way the program can be considered to be "learning" from the data. Once these models have been fit to the data, they can be used to predict and understand aspects of newly observed data.

Now let's see what types of machine learning exist. At the most fundamental level, machine learning can be categorized into two main types: *supervised learning* and *unsupervised learning*.

Supervised learning involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into *classification* tasks and *regression* tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities.

Unsupervised learning involves modeling the features of a dataset without reference to any label, and is often described as “letting the dataset speak for itself.” These models include tasks such as *clustering* and *dimensionality reduction*. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data.

In addition, there are so-called *semi-supervised learning* methods, which fall somewhere between supervised learning and unsupervised learning. Semi-supervised learning methods are often useful when only incomplete labels are available (VanderPlas 2017).

4.3 Motivation

We know from medicine, prevention is better than cure. To avoid future problems, we have to recognize them earlier, therefore we need prediction. Hidden Markov Model (HMM) can be used to predict problems that may arise in oil fields by analyzing time series data or images, especially in petroleum production (e.g. sucker rod pump).

As our world gets increasingly instrumented, sensors and systems are constantly emitting a relentless stream of time series data. Such data has numerous applications across various industries. Time series data is gathered, stored, visualized and analyzed for various purposes across various domains:

- In data mining, pattern recognition and machine learning, time series analysis is used for clustering, classification, query by content, anomaly detection and forecasting.
- In signal processing, control engineering and communication engineering, time-series data is used for signal detection and estimation.
- In statistics, econometrics, quantitative finance, seismology, meteorology, and geophysics the time series analysis is used for forecasting.

Time series analysis helps identify trends, cycles, and seasonal variances to aid in the forecasting of a future event. Time series analysis can be useful to see how a given variable changes over time (while time itself, in time series data, is often the independent variable). Time series analysis can also be used to examine how the changes associated with the chosen data point compare to shifts in other variables over the same time period (Influxdata Inc. o. J.).

4.4 Objective

The aim of this master thesis is to use hidden Markov model to analyze time series data gathered from the sucker rod pump and find the hidden states, which can be used for prediction of a future event, e.g. sucker rod pump failure diagnosis.

5 Procedure and implementation of the procedure

The results from HMM will be compared with other clustering methods, namely the Gaussian Mixture and K-Means. Therefore, it is better, before diving into the implementation of HMM on our data, to get an intuition about the difference between K-Means and Gaussian Mixture Model (GMM) or rather Gaussian Mixture HMM (GMMHMM).

Since a detailed discussion of the differences between K-Means and GMM is outside the scope of this thesis, only some very important differences will be discussed here.

The K-Means clustering model is simple and relatively easy to understand, but its simplicity leads to practical challenges in its application. In particular, the non-probabilistic nature of K-Means and its use of simple distance-from-cluster-center to assign cluster membership leads to poor performance in many real-world situations. The K-Means (an unsupervised learning algorithm) searches for a predetermined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like (VanderPlas 2017):

- The “cluster center” is the arithmetic mean of all the points belonging to the cluster.
- Each point is closer to its own cluster center than to other cluster centers.

Those two assumptions are the basis of the K-Means model. Gaussian mixture models (or rather Gaussian Mixture Hidden Markov Model, GMMHMM), also an unsupervised learning algorithm, can be viewed as an extension of the ideas behind K-Means, but can also be a powerful tool for estimation beyond simple clustering. Here are some differences between these two models (VanderPlas 2017):

- Lack of flexibility in cluster shape by K-Means: In K-Means, cluster models must be circular, K-Means has no built-in way of accounting for oblong or elliptical clusters. In contrast, in GMM each cluster is associated not with a hard-edged sphere but with a smooth Gaussian model (covariance type can be selected).
- How many components: The number of clusters must be selected beforehand if using K-Means (it cannot learn the number of clusters from the data). But the fact that GMM is a generative model gives us a natural means of determining the optimal number of components for a given dataset (BIC & AIC).
- Lack of probabilistic cluster assignment by K-Means: GMMs are probabilistic models, K-Means non-probabilistic. This leads to so-called *soft* and *hard* clustering. K-Means model has no true measure of probability or uncertainty of cluster assignment, where the probabilistic nature of GMM makes it possible to find probabilistic cluster assignments. That means GMM provides a probability for each cluster (state), which can help to make more precise decisions and predictions. For example, if there are two hidden states that a certain sample can belong to, in K-Means, this sample can take the values of 0 or 1, which shows that this sample belongs to one of them (or not). This is known as hard clustering. But GMM (or rather GMMHMM) returns the exact probability of a sample that belongs to a given state. For instance, see the following figures 27, 28 & 29. In figures 28 and 29, we can clearly see the exact probability of

each sample at any time where GMMHMM is used. This is known as soft (or fuzzy) clustering.

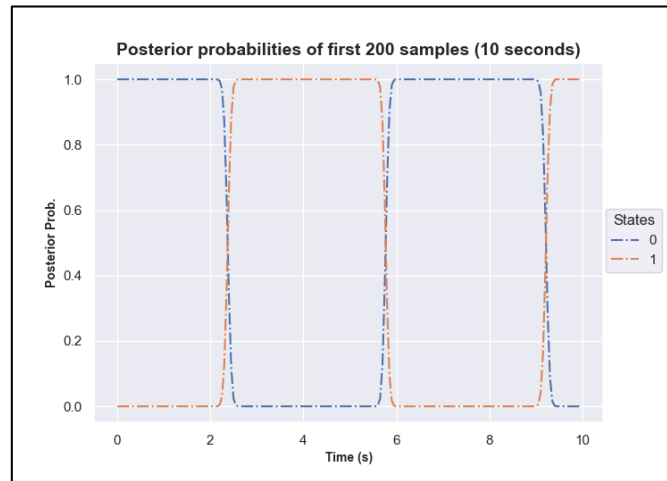


Figure 27: Posterior probabilities of first 10 seconds of polished rod position

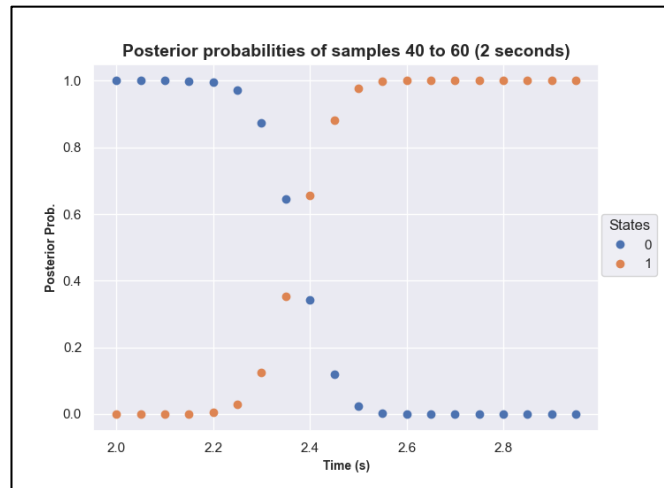


Figure 28: Posterior probabilities of samples 40 to 60 of polished rod position

Posterior Probabilities:			
	Time[s]	State 0	State 1
40	1.9998	1.0000	0.0000
41	2.0498	1.0000	0.0000
42	2.0998	1.0000	0.0000
43	2.1498	0.9996	0.0004
44	2.1998	0.9959	0.0041
45	2.2498	0.9717	0.0283
46	2.2998	0.8746	0.1254
47	2.3498	0.6465	0.3535
48	2.3998	0.3437	0.6563
49	2.4498	0.1185	0.8815
50	2.4998	0.0241	0.9759
51	2.5498	0.0026	0.9974
52	2.5998	0.0002	0.9998
53	2.6498	0.0000	1.0000
54	2.6998	0.0000	1.0000
55	2.7498	0.0000	1.0000
56	2.7998	0.0000	1.0000
57	2.8498	0.0000	1.0000
58	2.8998	0.0000	1.0000
59	2.9498	0.0000	1.0000

Figure 29: Values of posterior probabilities for the samples 40 to 60

There are other issues to be aware of when using the K-Means (VanderPlas 2017):

- The globally optimal result may not be achieved: Although the E-M procedure is guaranteed to improve the result in each step, there is no assurance that it will lead to the global best solution.
- K-Means can be slow for large numbers of samples: Because each iteration of K-Means must access every point in the dataset, the algorithm can be relatively slow as the number of samples grows.

5.1 Data preparation

For this work, data from a sucker rod pump was provided, which was recorded and collected for a period of 4-hours, as shown in figure 30. To get a very accurate representation of the pumping system's operating conditions, samples were recorded for every 50 ms (20 samples per second), which is the typical sampling frequency. Since samples were recorded for every 50 ms, it started out at 288000 samples. For the purposes of this work, all samples were not required, therefore for simplicity and to save time by implementation, the data were reduced to $\approx 10\%$ of the initial dataset (28883 samples), where the transitions that are important for this work were retained (see figures 30 & 32).

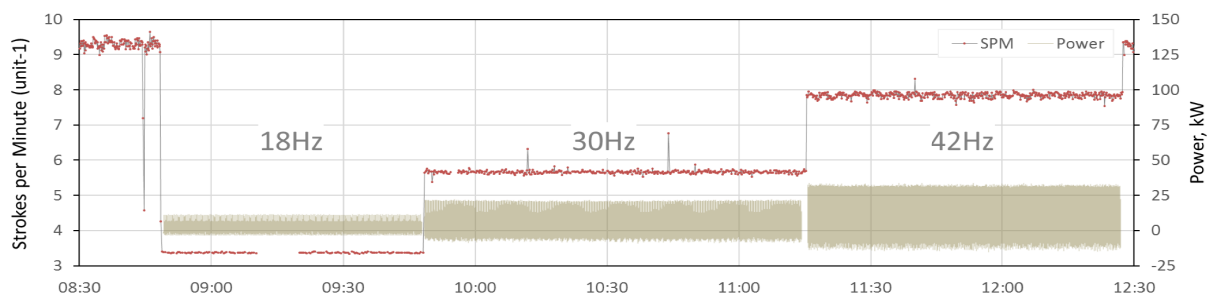


Figure 30: Motor power and strokes per minute over time

#	Name	meaning	unit	Comment
1	TS date (unit-1)	date time		
2	TS sec (unit-1)	time, sec	sec	
3	Pmo, kW	motor power	kW	
4	Psch, kW	motor power	kW	not used
5	Pme, kW	mechanical power	kW	
6	motPHI:uw	motor rotational angle	rad	
7	motRPM:uw	motor rotational speed	RPM	
8	motRPA:uw	motor rotational acceleration		
9	redPHI:uw	reducer rotational angle	rad	
10	redRPM:uw	reducer rotational speed	RPM	
11	redRPA:uw	reducer rotational acceleration		
12	wbPHI:uw	walking beam rotational angle	rad	
13	wbRPM:uw	walking beam rotational speed	RPM	
14	wbRPA:uw	walking beam rotational acceleration		
15	wbPRP:uw, m	polished rod position	m	
16	wbPRV:uw, m/s	polished rod velocity	m/s	
17	wbPRA:uw, m/s ²	polished rod acceleration	m/s ²	
18	mrRPMratio	motor/reducer RPM ratio	—	
19	LC, kN	load from polished rod load cell	kN	
20	Ptub, bar	tubing pressure	bar	
21	Pcas, bar	casing pressure	bar	not used
22	Tcis, °C	container inside temperature	°C	not used
23	Tcos, °C	container outside temperature	°C	not used
24	DFL, m	dynamic fluid level	m	not used
25	stype	stroke type (begin upstroke, begin downstroke)	BuS, BdS	
26	spos	polished rod position @ BuS or BdS	m	
27	sload	load at polished rod position @ BuS or BdS	kN	

Figure 31: Features recorded and collected from the sucker rod pump

The values of 27 features such as motor power, polished rod position, motor rotation speed etc. were measured and recorded for each sample/ time step, as shown in figure 31, where the most important features of the dataset are highlighted in red.

To avoid a negative impact on the analysis, some incomplete columns where values were missing have been removed. Therefore, the (final) dataset that will be processed and analyzed consists of the following columns (19 columns) and 28883 rows (samples), as shown in figure 32. In addition to the number of samples and features, some other important information about the dataset, such as datatype, mean, standard deviation, min and max etc. can be gathered from the dataset, as shown in figure 33.

```

RangeIndex: 28883 entries, 0 to 28882
Data columns (total 19 columns):
#   Column              Non-Null Count  Dtype
---  -
0   TS sec (unit-1)      28883 non-null  float64
1   Pmo, kW              28883 non-null  float64
2   Pme, kW              28883 non-null  float64
3   motPHI:uw           28883 non-null  float64
4   motRPM:uw           28883 non-null  float64
5   motRPA:uw           28883 non-null  float64
6   redPHI:uw           28883 non-null  float64
7   redRPM:uw           28883 non-null  float64
8   redRPA:uw           28883 non-null  float64
9   wbPHI:uw            28883 non-null  float64
10  wbRPM:uw            28883 non-null  float64
11  wbRPA:uw            28883 non-null  float64
12  wbPRP:uw, m         28883 non-null  float64
13  wbPRV:uw, m/s      28883 non-null  float64
14  wbPRA:uw, m/s2     28883 non-null  float64
15  mrRPMratio          28883 non-null  float64
16  LC, kN              28883 non-null  float64
17  Ptub, bar           28883 non-null  float64
18  stype                296 non-null    object
dtypes: float64(18), object(1)

```

Figure 32: Info about dataset after removing incomplete columns

	TS sec (unit-1)	Pmo, kW	motRPM:uw	wbPRP:uw, m	wbPRV:uw, m/s	wbPRA:uw, m/s2	mrRPMratio	LC, kN	Ptub, bar
count	28883.000000	28883.000000	28883.000000	28883.000000	28883.000000	28883.000000	28883.000000	28883.000000	28883.000000
mean	722.049800	7.793482	753.373604	2.002051	-0.000482	0.000533	3.847518	33.624539	7.484151
std	416.897412	11.951141	243.932693	1.246111	0.894779	0.773589	0.005883	7.508672	0.302600
min	-0.000200	-14.427902	346.503857	0.000908	-1.688625	-1.384618	3.825752	18.825101	6.815331
25%	361.024800	-1.390939	592.908261	0.766431	-0.683724	-0.513104	3.843832	26.795957	7.289643
50%	722.049800	5.724768	832.136514	2.145193	-0.044160	-0.110692	3.847685	34.086825	7.468729
75%	1083.074800	15.691447	1010.790074	3.231840	0.722943	0.368616	3.850572	40.249211	7.719228
max	1444.099800	41.969149	1024.254875	3.647703	1.642056	2.102924	3.873284	51.901198	8.100005

Figure 33: Important information about dataset

How the dataset looks like and how they change over time is shown in figure 34. If we take a closer look at the figure, we can clearly see that there are four different stages of motor power or motor RPM in the dataset. This is not the case with the polished rod position, where this difference cannot be clearly determined. Whether there are four (hidden) states or more or maybe less than four will be discussed in the coming sections.

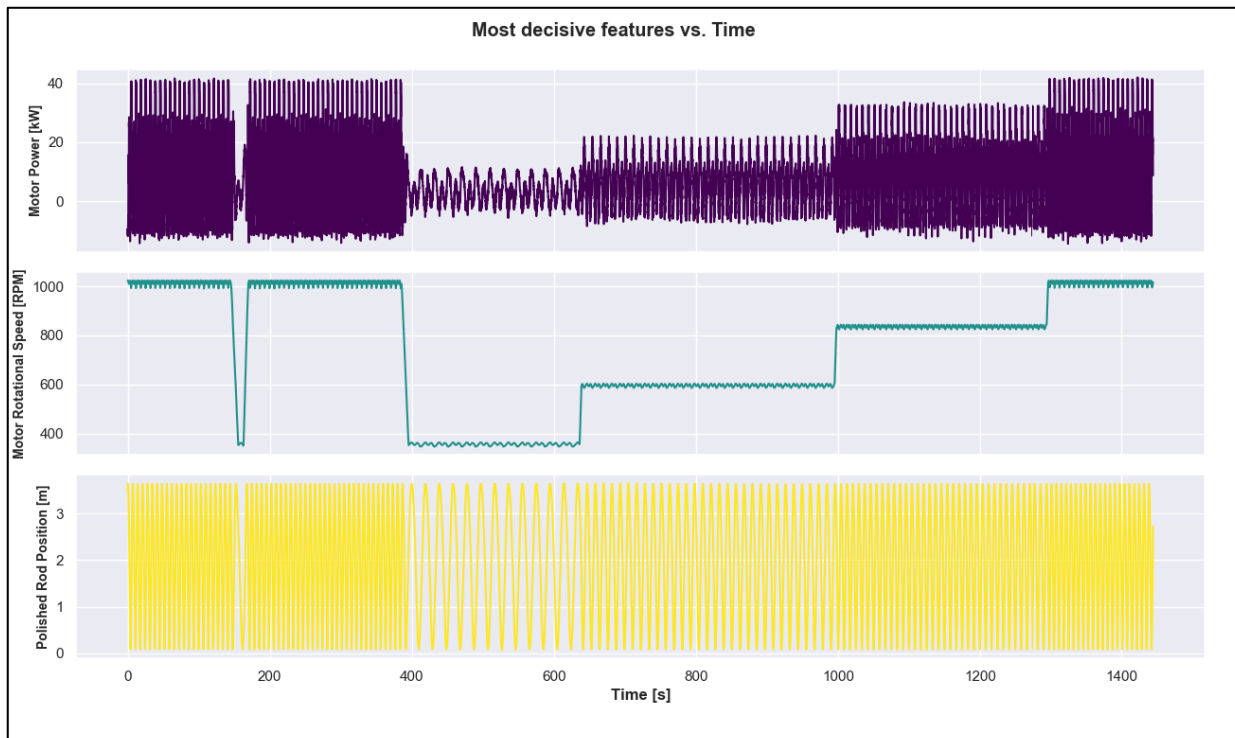


Figure 34: Development of three most decisive features over time

5.2 Procedure

As already mentioned in section Baum-Welch algorithm, if the model parameters λ (a , b and π), and the number of possible states are known to us, we can use the forward algorithm and Viterbi algorithm to obtain the $P(V|\lambda)$ and the most probable state sequence. If it is not the case, that means model parameters λ and/or possible states are unknown, and we have to find these parameters and values from the data. We can use BIC (Bayesian Information Criterion), AIC (Akaike Information Criterion) or Silhouette analysis (this will be discussed in detail) to get an assessment of the possible states. But if only the model parameters are missing, we can use the Baum-Welch algorithm to estimate these parameters.

In this work, time series data that was collected from the sucker rod pump will be processed and analyzed. We don't know the relationship between measured features of the data. That means we neither know the number of hidden states nor the model parameters. Since this knowledge about the data is the prerequisite to using a supervised learning method, which is missing in our case, we have to select an unsupervised learning method. Therefore, the following main steps are necessary:

- Selection and preparation of data required for certain model
- Estimation of the number of possible hidden states
- Train data to obtain model parameters λ (third problem of HMM)
- Estimation of probability for observation sequence $P(V|\lambda)$ (first problem of HMM)
- Predicting best state sequence (second problem of HMM)

For that purpose, programming processes are required. In this work, the programming language *Python 3* is used. And once again, the two main libraries (or packages) used for the construction and implementation of the models are *hmmlearn* and *scikitlearn*.

6 Results

We know that the amount of fluid lifted by the sucker rod pump depends on pump size, tubing size, stroke length and strokes per minute. We also know that surface pumping speed can cause some problems, for example, an excessive surface pumping speed leads to a shortening of stroke. Therefore, we can try to use HMM to find stroke duration, the number of strokes per minute and change in surface pumping speed. Then we can try to use HMM to find the start and end of upstroke and downstroke as it can maybe help us in failure diagnosis. Finally, using HMM to observe the sucker rod pump operation over time (finding hidden states). Of course, in order to be able to complete each task in the best possible way, we first have to think about which features can help us with the respective task.

There are many models to solve typical unsupervised learning problems, and the Gaussian Mixture Model or rather the Gaussian Mixture Hidden Markov Model (GMMHMM), is one of them.

6.1 Stroke duration and strokes per minute

“The polished rod is the top and strongest part of the sucker rod string, and it connects the rods to the pumping unit. Its main functions are:

- To transfer the pumping loads to the surface pumping unit
- To provide a seal in the installation's stuffing box to prevent well fluids from entering the atmosphere

In order to fulfill its functions, the polished rod must be strong enough to carry the full load during the pumping cycle. Polished rods are standardized in API Spec 11B and come in different sizes with outside diameters of 1 1/8 in, 1 1/4 in, and 1 1/2 in. Available standard lengths are between 8 ft and 36 ft, and their proper length is calculated so as to equal the sum of the following measurements (ScienceDirect o. J.; Takacs 2015):

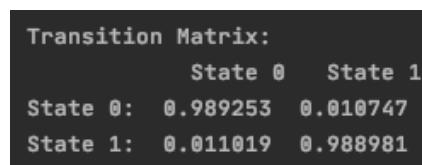
- The maximum anticipated stroke length of the pumping unit
- The distance above the stuffing box to the lowest position of the carrier bar
- Some extra lengths (safety distances) for accommodating (a) the polished rod clamp and (b) extra length below the stuffing box”

Therefore, since the stroke length is proportional to the polished rod displacement, the polished rod position can be considered as a good choice to obtain the length and strokes per minute.

If a sucker rod pump works properly and without any significant problem, the polished rod will move up and down (Up- and Downstroke), which means two possible states were expected. If we look at the polished rod position in figure 34, we can see that it is true, there might be two hidden states. As previously mentioned, if we know the number of hidden states and also the model parameters, the Forward and Viterbi algorithms can be used to find $P(O|\lambda)$ and the best hidden states sequence. Since we are dealing with continuous data and know only the number of possible hidden states but not the model parameters, we have to use Gaussian HMM or Gaussian Mixture HMM. We know that there is more than one normal distribution (possible

number of hidden states), therefore, we will use the Gaussian Mixture HMM model (GMMHMM) for this purpose.

As mentioned, we only know the number of possible hidden states but not the model parameters. Therefore first, we have to train the data to get the model parameters and then predict the hidden states, as illustrated in figure 36. The initial probability π will be equal for both states (50%) since in our case, the first step happens only once. We won't have emission probability (or emission matrix) if we use GMMHMM, because the model doesn't map to discrete data. Therefore, the emission probability will be described by means and covariance of the trained data. The transition probability (or transition matrix) is shown in figure 35. Since there are only two states (Up- and Downstroke) and there is always a movement from one state to the other, one would expect the probability of moving from state 0 to state one and vice versa to be 1. But as you can see, it is not the case, it looks like the system stays in the same state most of the time. Don't be confused if remember, we said that GMMHMMs are probabilistic models and make so-called soft clustering, which leads to find probabilistic cluster assignments. That means, GMMHMM labels each sample to a certain state. Since data is recorded for every 0,05 second (20 samples per second), and each stroke lasts several seconds, we have a state change every 0,05 second, which moves to the same state most of the time (see figure 36).



```
Transition Matrix:
      State 0   State 1
State 0: 0.989253 0.010747
State 1: 0.011019 0.988981
```

Figure 35: Transition matrix using feature polished rod position

It is important to note that the entire column of polished rod position has been trained, but in the figure below only a small part of polished rod position vs. time (sample 2500 to 8500) has been shown to have a better view. Histogram and density function plots represent the whole samples.

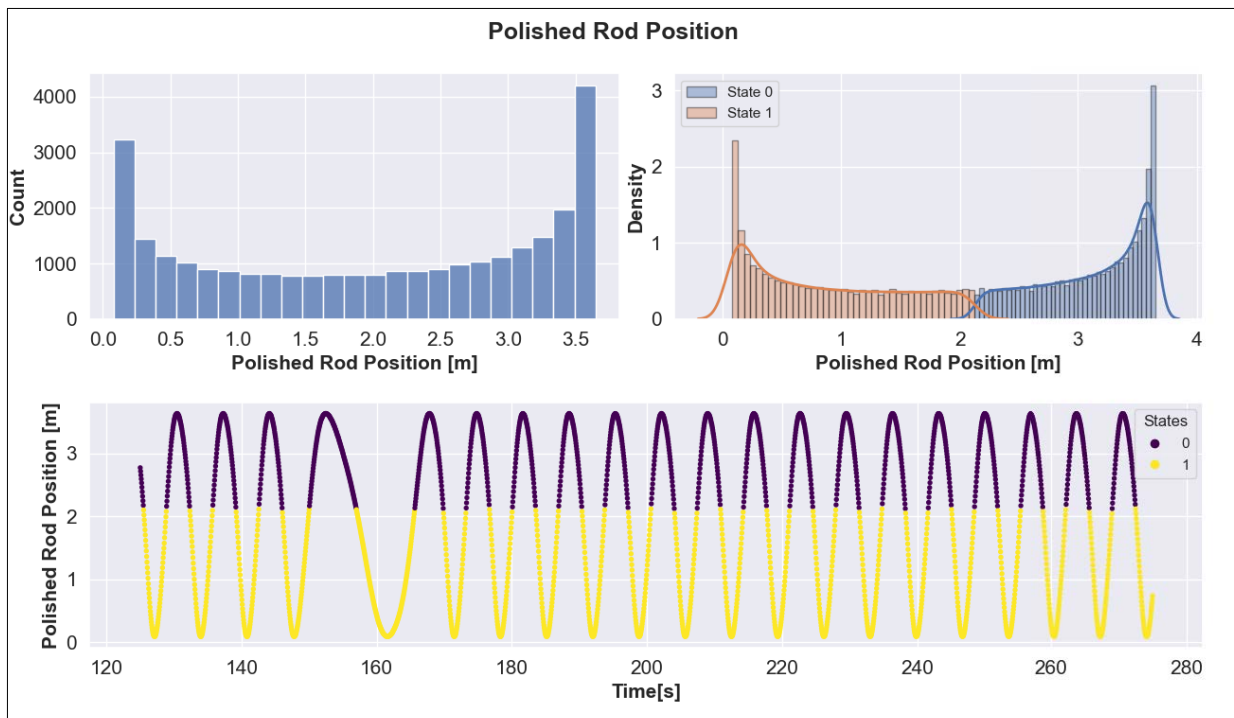


Figure 36: Polished rod position and the associated histogram and density function

In order to find stroke length and strokes per minute, we do not need all the procedures and results explained and presented above. The procedure and results are just a small introduction to show how GMMHMM works. To find out stroke length and strokes per minute using HMM, you can proceed in different ways. To obtain stroke duration and strokes per minute, as shown in figure 37, only mean and standard deviation calculated from GMMHMM were used.

	Stroke Duration[s]	≈Strokes/minute
RPM ≈ 355 (18 Hz)	19.50	3.0
RPM ≈ 600 (30 Hz)	11.60	5.0
RPM ≈ 835 (42 Hz)	8.25	7.0
RPM ≈ 1010 (55 Hz)	6.85	9.0

Number of all strokes:	157	
Avg. strokes per minute:	6.52	

Figure 37: Stroke duration and strokes per minute

6.2 Upstroke & downstroke from DC

“The efficiency of sucker rod pumping units is usually analyzed using the information from a pump dynagraph and polisher rod dynamometer cards (DC). The main purpose of a dynamometer survey, which can be classified into surface dynamometers and downhole dynamometers, is to obtain a representative dynamometer card (DC) that properly characterizes the stabilized operation of the pumping system. Such cards represent the steady-state operation of the sucker-rod pump as well as other subsurface and surface equipment” (ScienceDirect o. J.; Takacs 2015).

HMM is a very powerful tool for image classification. If there is enough DC's as image available to us, HMM can be used to detect and predict future failures reading and classifying DC

images¹. In this work we will not be able to deal with images classification for two reasons, firstly because in this work we are dealing with time series data (and not images), which was recorded and collected for a period of only 4-hours; there is no significant change in the shape of DC (as you can see in figure 39) to recognize or predict failures. Second, there are a large number of typical dynamometer cards, train HMM to classify, recognize and predict failures that might occur in the future based on DC-Images, which is out of the scope of this work. Therefore, what we can do in this work as the next step is simply to use HMM to find the start and end of upstroke and downstroke, which can be considered as the first step of the failure diagnosis.

To do this, we must first consider which features available to us could be used for this purpose. Polished rod kinematic parameters, which are PR position, PR velocity and PR acceleration, are important for analyzing the performance of surface pumping units. We know if the polished rod reaches its highest or lowest point, the polished rod velocity is around zero. If we take a closer look at figure 38, we see that it's indeed the case here as well, when the PR-velocity is around zero, PR-position reaches its highest or lowest point, which is the start (or end) of upstrokes and downstrokes.

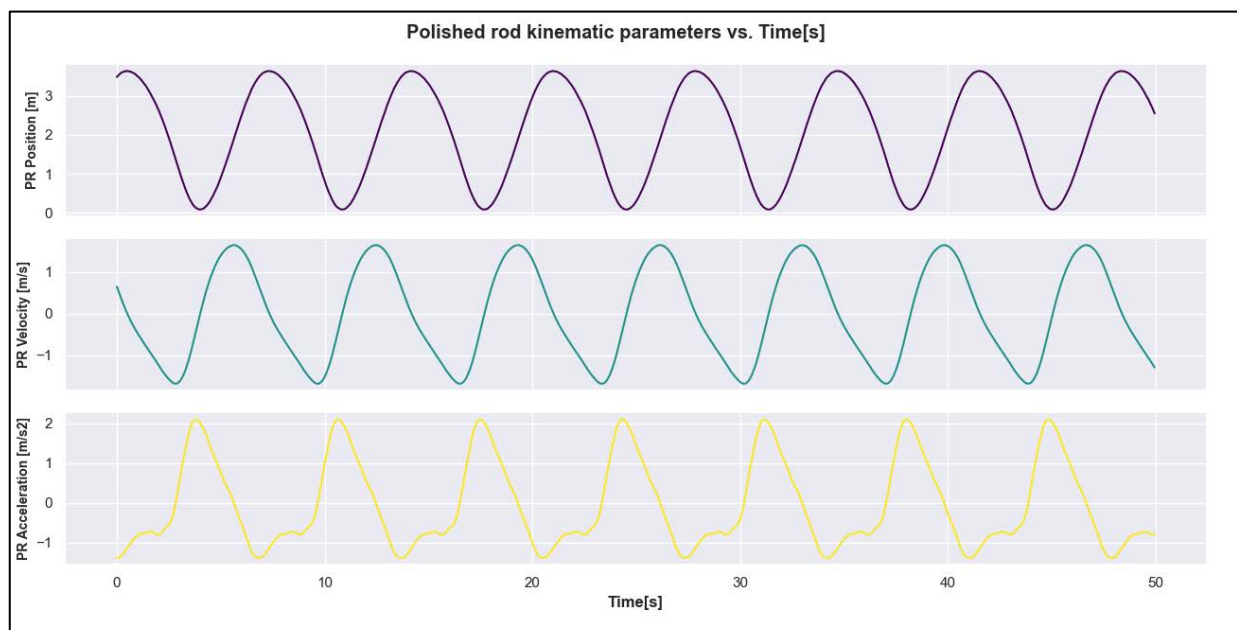


Figure 38: Polished rod kinematic parameters vs. time

The dataset that we need for this part of the work is the polished rod velocity. But again, we only have a premonition of the number of possible hidden states, but not the model parameters. Therefore first, we have to train the dataset to get the model parameters and then predict the hidden states (Up- and Downstroke), as illustrated in figure 39. The states 0 and 1 stand for upstroke and downstroke respectively.

¹ Zheng B, Gao X. Diagnosis of sucker rod pumping based on dynamometer card decomposition and hidden Markov model. Transactions of the Institute of Measurement and Control. 2018;40(16):4309-4320. doi:10.1177/0142331217746492

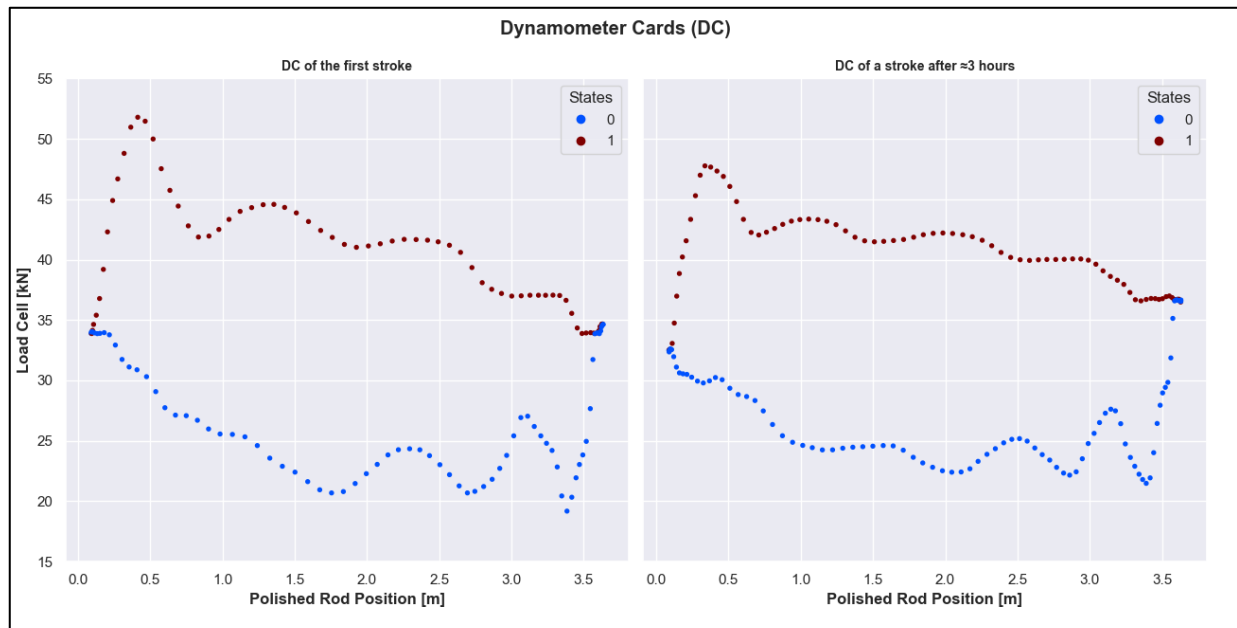


Figure 39: Start and end of up- and downstroke of 2 different cycles

6.3 Sucker rod pump operation over time

Another facet of this work is to observe the transformation and the changes of features over time and thereby find out the hidden states. It will be very helpful in prediction future problems that may occur. For example, the future performance of the motor power and/ or motor rotational speed can be predicted, which are the most decisive factors by optimizing downhole fluid production and also the most decisive constituent of operating expenditures.

As mentioned earlier, in order to be able to complete each task in the best possible way, we first have to think about which features can help us with the respective task, as it was the case with other 2 examples. Therefore, first we will try it with the entire dataset, then with a part of the dataset selected for this purpose (motor rotational speed, motor rotational acceleration, reducer rotational acceleration and polished rod acceleration).

Previously we talked about advantages of GMMHMM (or GMM) compared to K-Means, advantages like soft clustering, density estimation etc., but there are two disadvantages that all three clustering algorithms applied here have. The first one is, the EM algorithms need to be initialized (see chapter 3.1.3), which is done by default in Python but also can be done manually. The second disadvantage and the most important is that the number of clusters (components or hidden states) must be defined. Train a model with an incorrect number of components can result in overfitting or underfitting, that leads to poor performance in machine learning. To be able to use clustering algorithms, we have to have a premonition of the number of states.

6.3.1 Number of components

In the last two sections, to accomplish our task, we had a premonition of the number of hidden states in the dataset. Now, by a closer look at the development of the data over time in figure 34, 4 different zones (states) can be distinguished. But it is only possible because this dataset

is recorded for a short time (4 hours), for a large dataset it is not the right approach. Therefore, other techniques have to be employed to get a rough idea about number of components (hidden states).

Some statistical approaches like Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) and also Silhouette Analysis (will be explained in chapter 6.3.2) can be used for this purpose. Scikit-Learn's package in Python includes all these methods. As mentioned, the results of GMMHMM will be compared with GMM and K-Means methods. Therefore, we use the Silhouette method that covers all three methods because BIC and AIC are standard methods only for GMM. Of course, only GMM with one of these approaches can be used to get an idea about the number of components, but a comparison with other two models would not be possible.

Note, it does not matter which approach we will use, the estimated number of clusters or hidden states does not necessarily mean that there is exactly that number of clusters in the data. We need to train the data with a different number of components but around that number and then we compare the results.

For instance, we can go through the example of the Scikit-Learn's package, which has been adapted to our dataset with some minor modifications. For this purpose, motor power and polished rod kinematic parameters (PR-position, PR-velocity and PR-acceleration) were used. The Bayesian information criterion (BIC) approach and the Gaussian mixture model GMM with four different covariance types (spherical, tied, diagonal and full) were applied, as illustrated in figure 40.

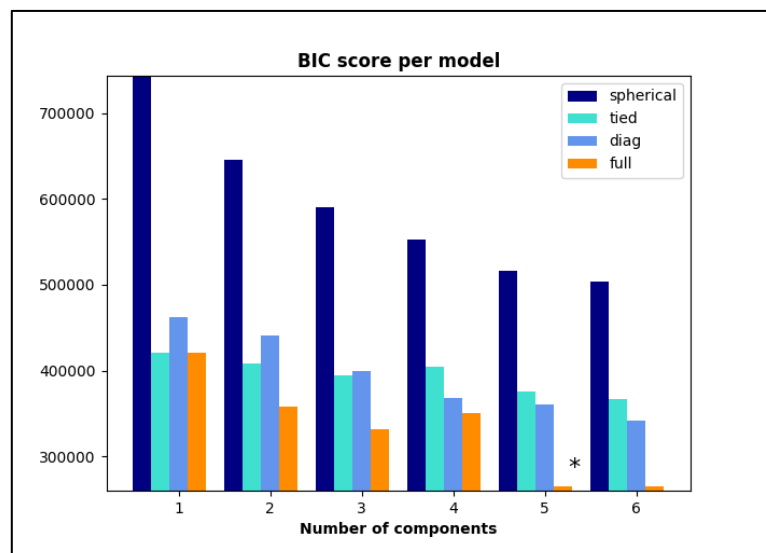


Figure 40: BIC for 6 diff. components (states) and 4 diff. covariance type

The lower the BIC, the better is the model. As you can see, it turns out that we receive best results using a model with five components and the covariance type “full”. That means, we should train the data with 4, 5 and 6 components to get best results, using for example 2 or 10 components leads to underfitting or overfitting.

6.3.2 Outcomes of sucker rod pump operation over time

As the first step, Silhouette analysis will be done to get a number of components. “Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like a number of clusters visually. This measure has a range of $[-1, 1]$. Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters, and negative values indicate that those samples might have been assigned to the wrong cluster” (scikit-learn 2007).

For training the model to get the right number of components, just features (columns) have to be used that describe best the behavior of the sucker rod pump. Therefore, features like time (because it is only increasing) have to be removed from the dataset that will be used to train the model. Here the dataset illustrated in figure 32 was used (the entire dataset), except for the time-column. The Silhouette analysis was performed for all three models and five different states (components).

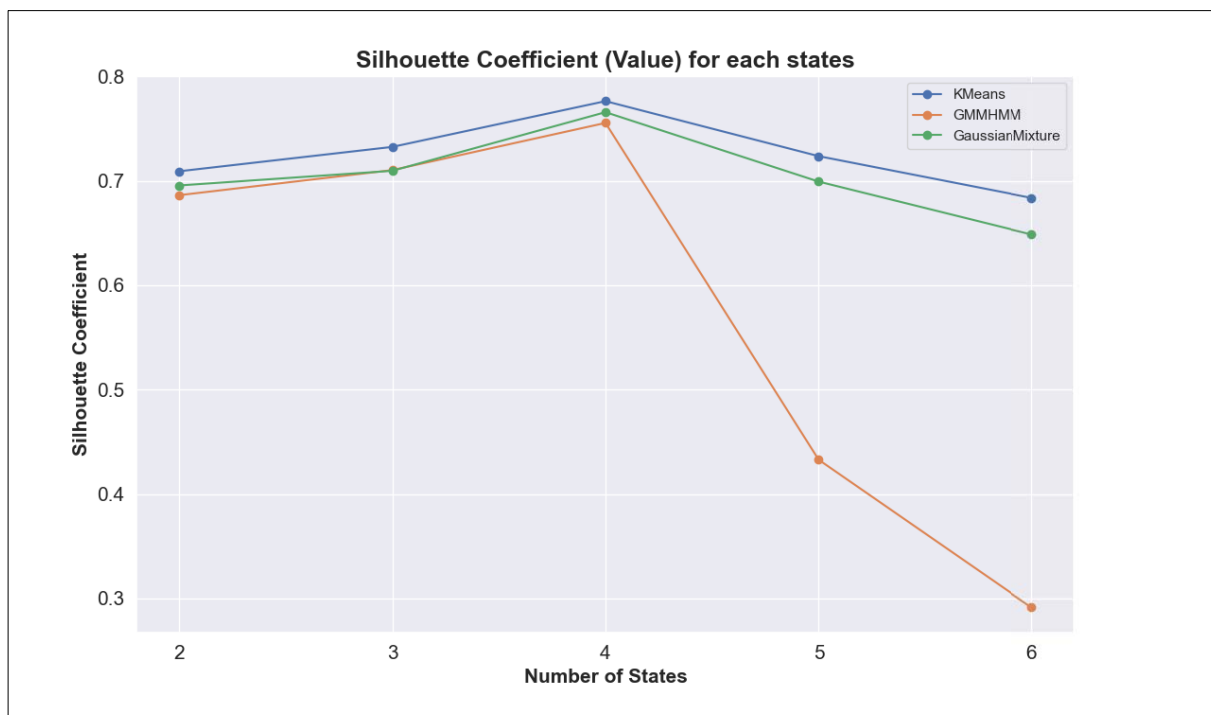


Figure 41: Silhouette coefficients for each model at each state (for the entire dataset)

```
--- Model KMeans
For n_states = 2 The average silhouette_score is : 0.7091262799219481
For n_states = 3 The average silhouette_score is : 0.7324665029770392
For n_states = 4 The average silhouette_score is : 0.7762963454472978
For n_states = 5 The average silhouette_score is : 0.7236017723154845
For n_states = 6 The average silhouette_score is : 0.6837121750675028

--- Model GMMHMM
For n_states = 2 The average silhouette_score is : 0.6862177633488876
For n_states = 3 The average silhouette_score is : 0.7102543077433218
For n_states = 4 The average silhouette_score is : 0.7554259721300092
For n_states = 5 The average silhouette_score is : 0.4327977286238491
For n_states = 6 The average silhouette_score is : 0.2915203041310399

--- Model GaussianMixture
For n_states = 2 The average silhouette_score is : 0.6955225714617388
For n_states = 3 The average silhouette_score is : 0.7095899026179208
For n_states = 4 The average silhouette_score is : 0.7657243998459122
For n_states = 5 The average silhouette_score is : 0.6992171239913999
For n_states = 6 The average silhouette_score is : 0.6486457536495285
```

Figure 42: Silhouette coefficients values

All three methods show that a model with four components (hidden states) will supply the best results, where 3 or 5 components would also be possible. Silhouette coefficients for four components are between 0.75 and 0.77, which are pretty good values (positive values and near 1).

Note, GMM and GMMHMM try to detect the underlying groups or states of data, where K-Means more or less tries to divide data into different parts. That's why K-Means shows slightly better values than the other two models, but it does not mean that it does a better detection of underlying states.

Now with this information about the number of components and also covariance type, which we got from BIC, the model can be trained and then the hidden states can be predicted. Implementation is done again for all three models with five different states. Figures 43-51 show the models with 3, 4 and 5 states, the models with 2 and 6 states can be found in Appendix B. All three models with three states are underfitted (figures 43, 44 & 45), and with five states overfitted (figures 46, 47 & 48), which is also the case with 2 or 6 states (see Appendix B).

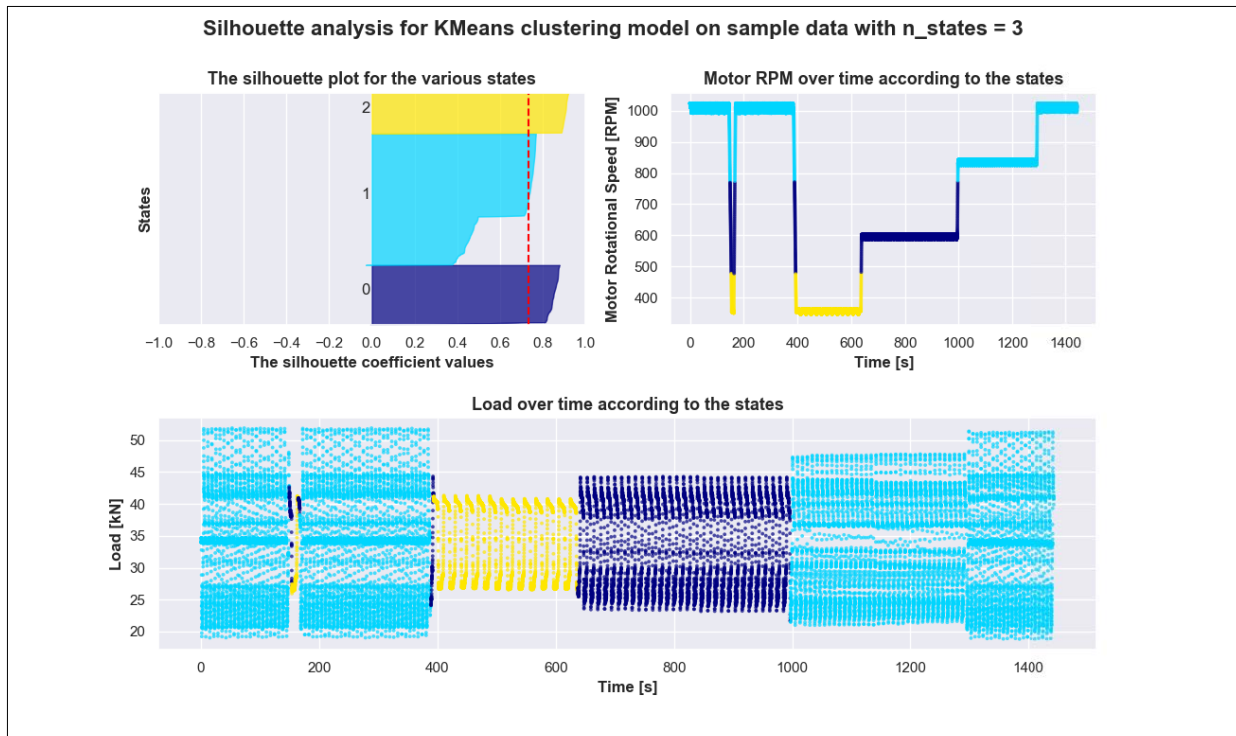


Figure 43: K-Mean with three states

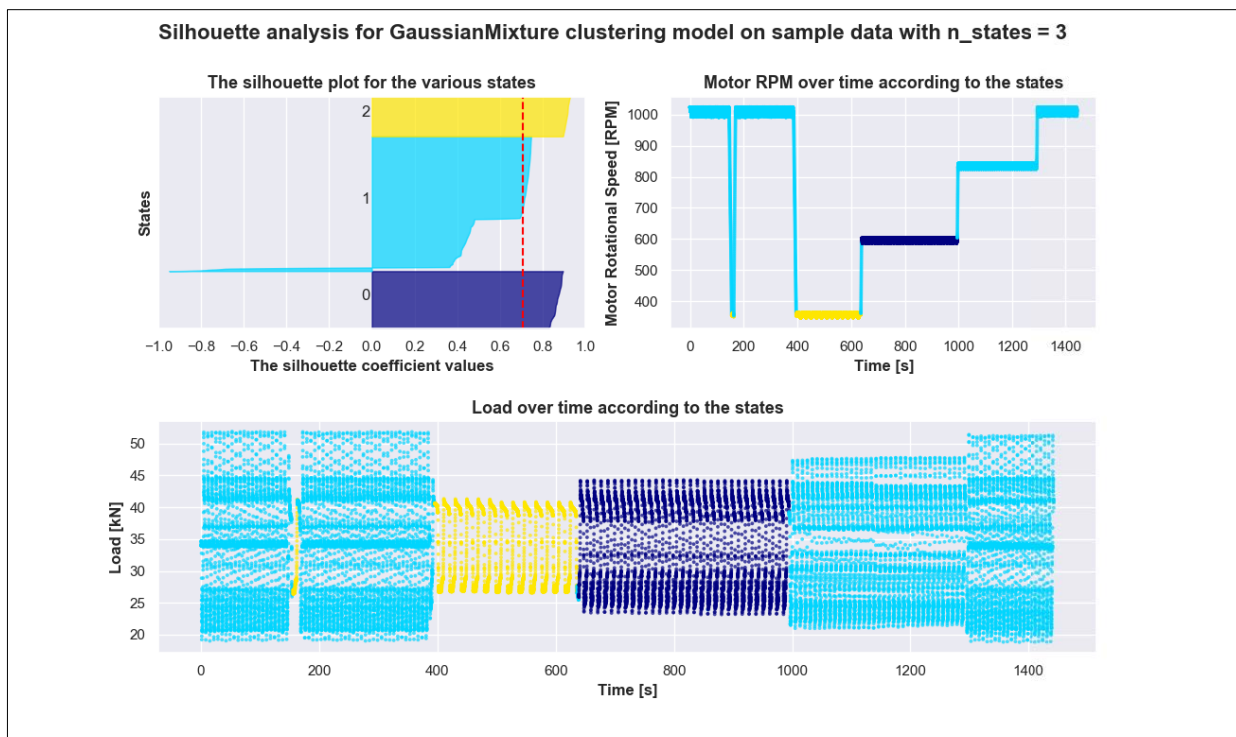


Figure 44: Gaussian Mixture (GMM) with three states

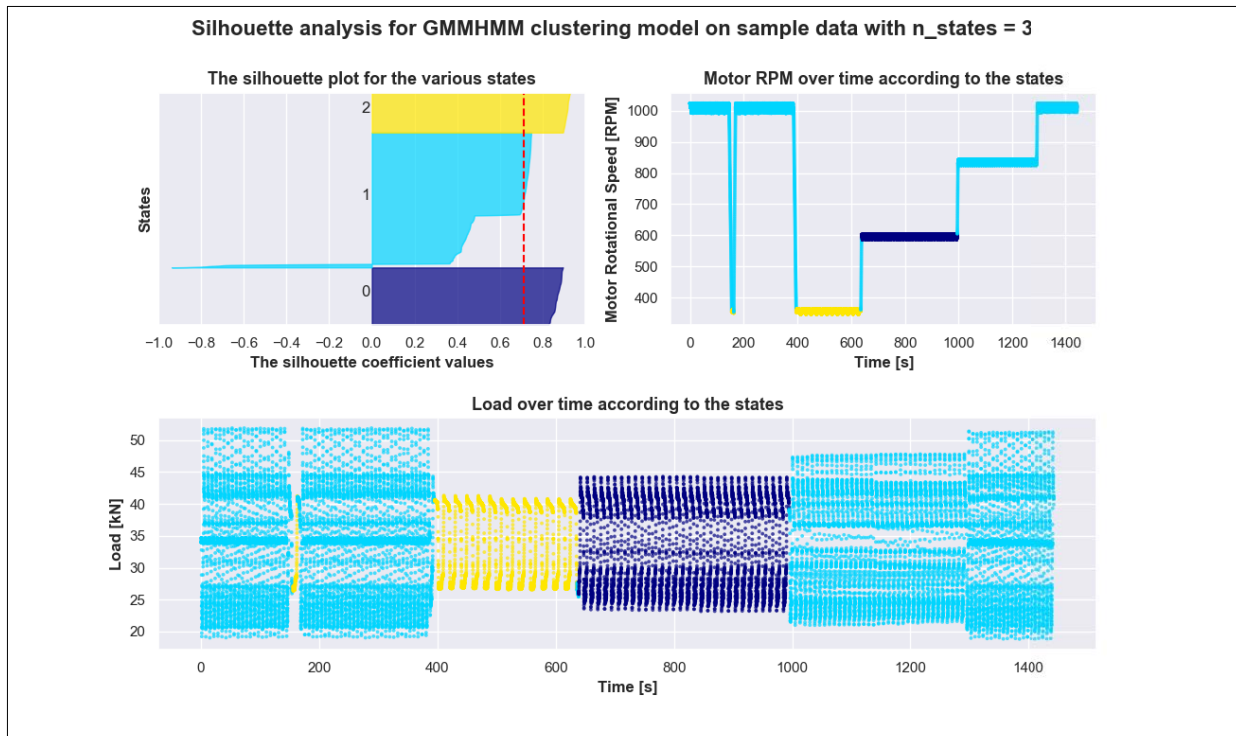


Figure 45: GMMHMM with three states

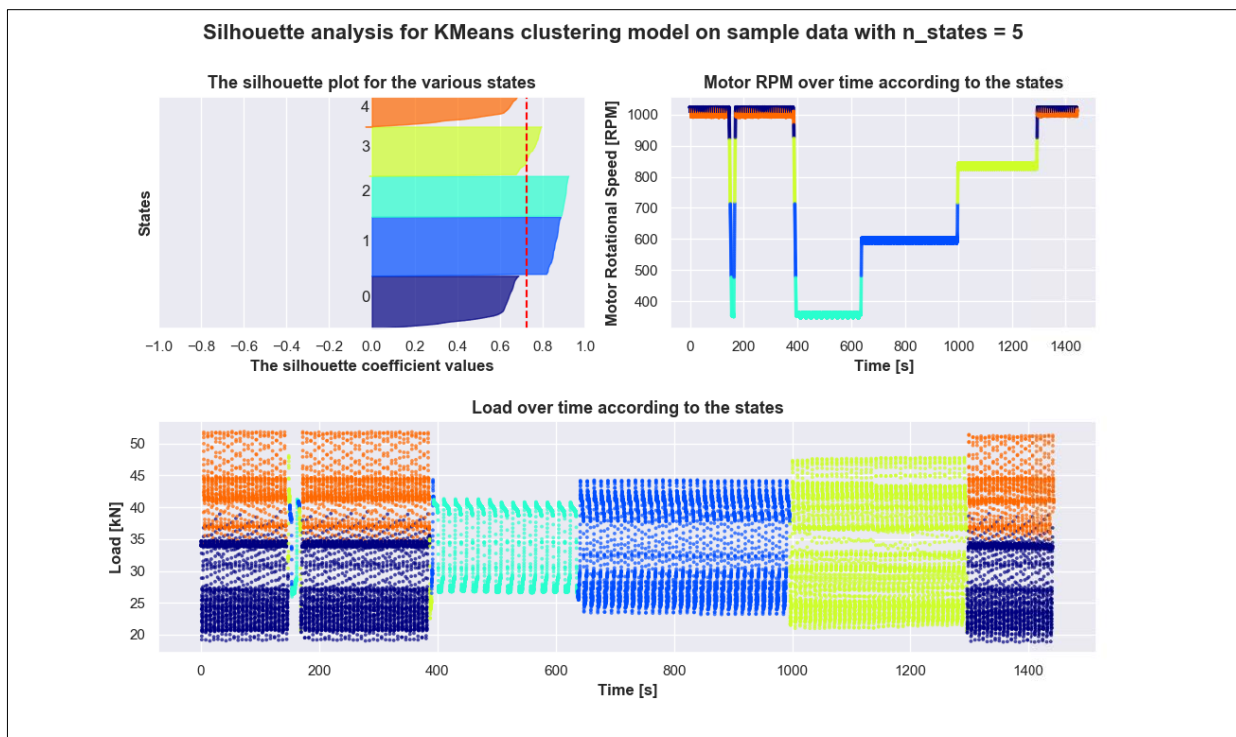


Figure 46: K-Mean with five states

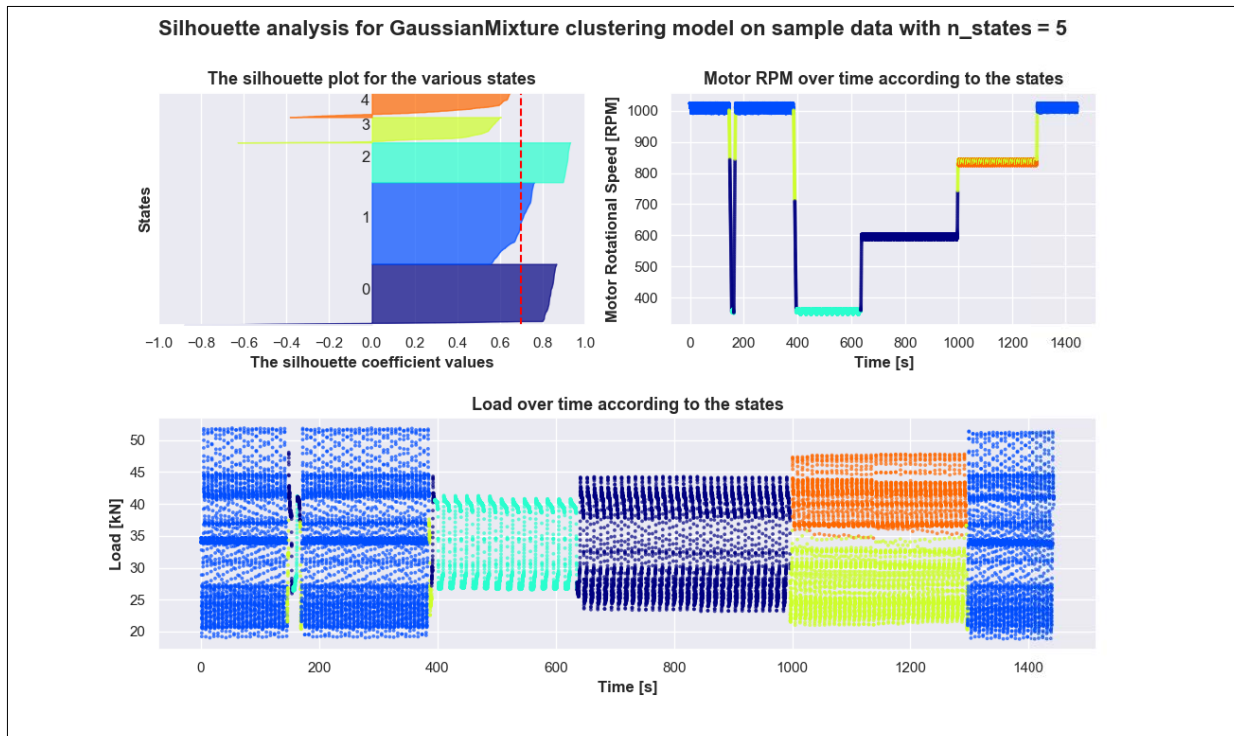


Figure 47: Gaussian Mixture (GMM) with five states

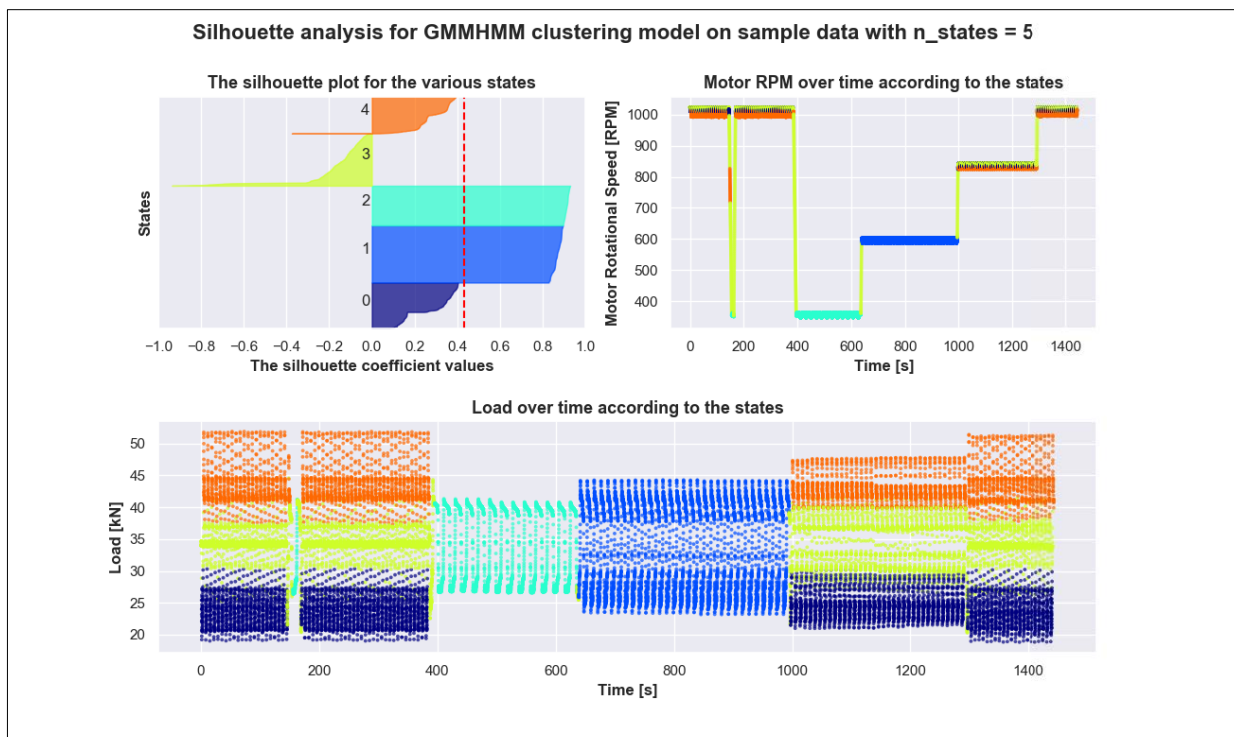


Figure 48: GMMHMM with five states

All models with four states show the best results, which was to be expected. All models correctly recognize the four main states, however, there are small differences that become apparent when moving from one (main) state to another one. GMM and GMMHMM try to detect the underlying groups or states of data, where K-Means more or less tries to divide data into different parts. It can be clearly seen in figure 46 (top right), that K-Means divided data into four different parts. GMM shows more or less the same results as K-Means. GMMHMM

identifies the behavior of the samples of the entire transition phase as the third states (see figure 51 & 52).

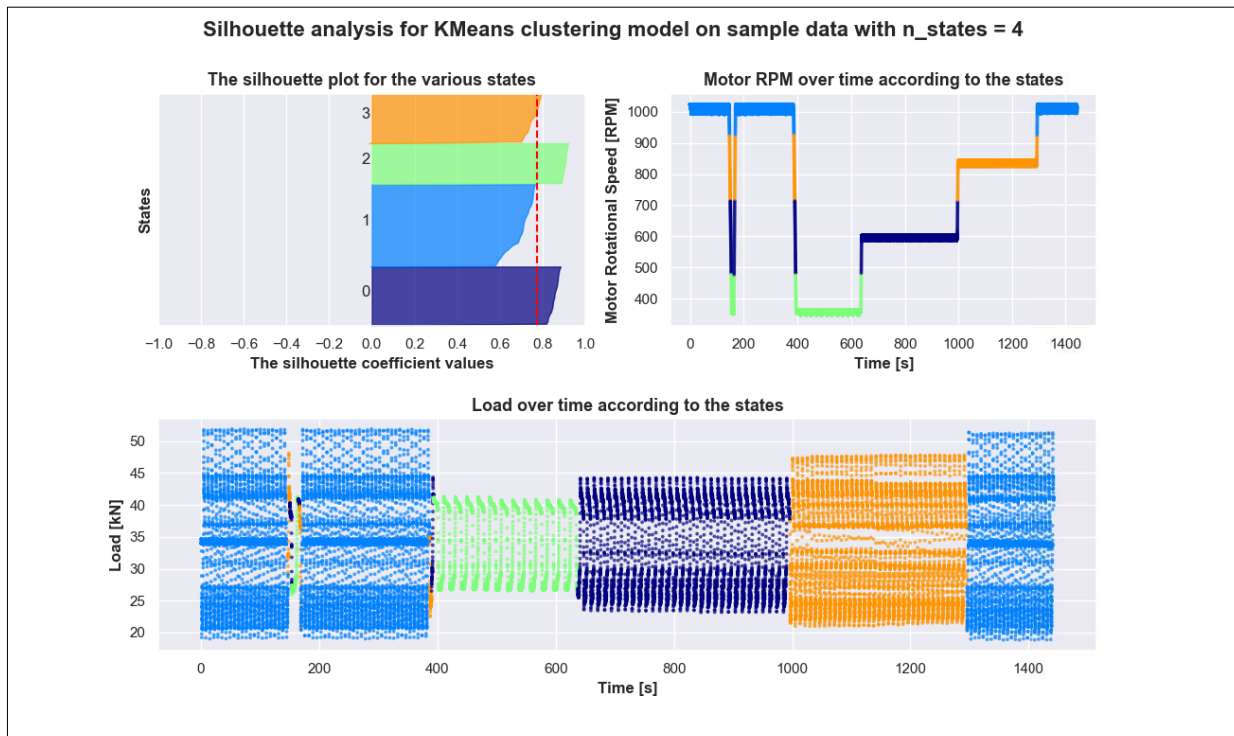


Figure 49: K-Mean with four states

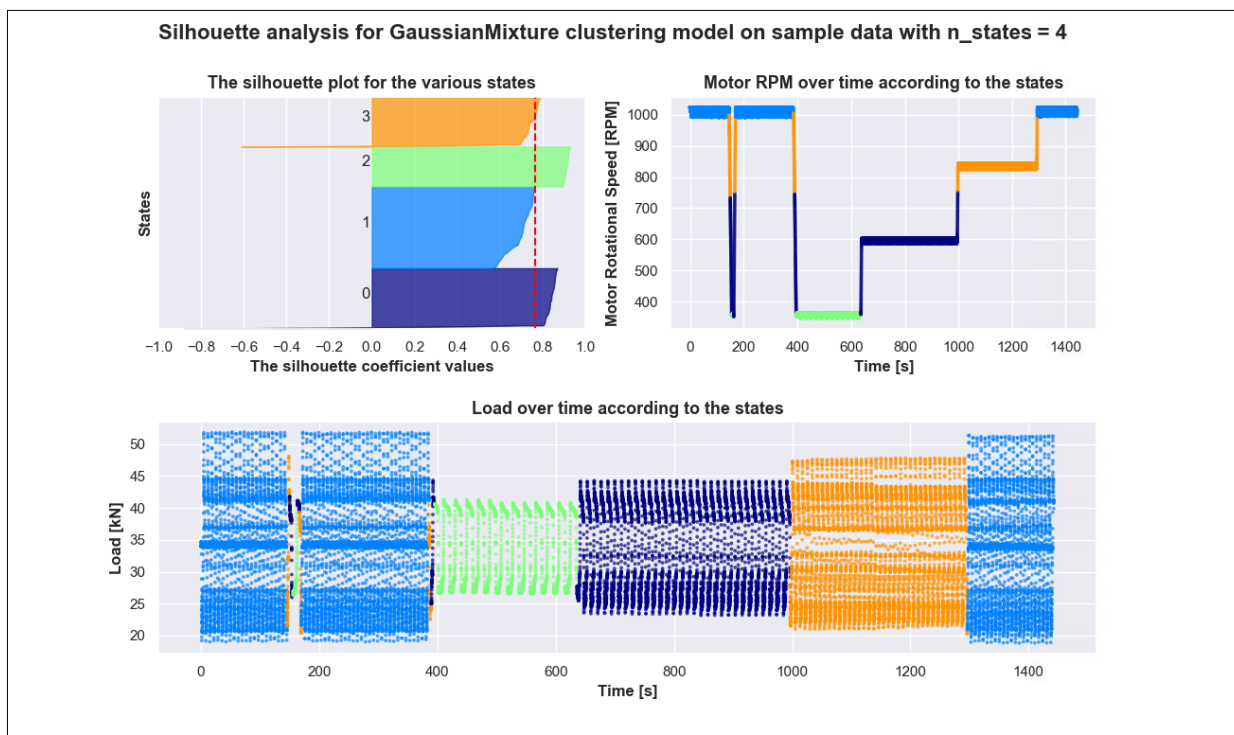


Figure 50: Gaussian Mixture (GMM) with four states

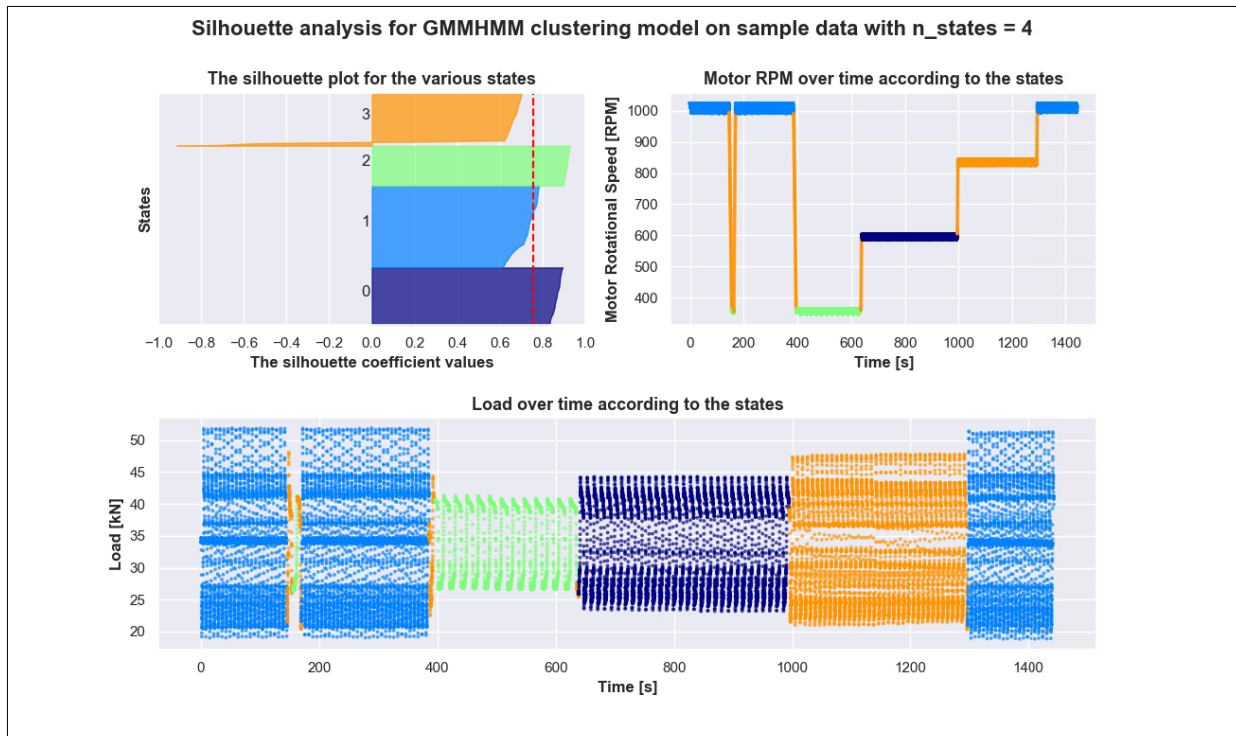


Figure 51: GMMHMM with four states

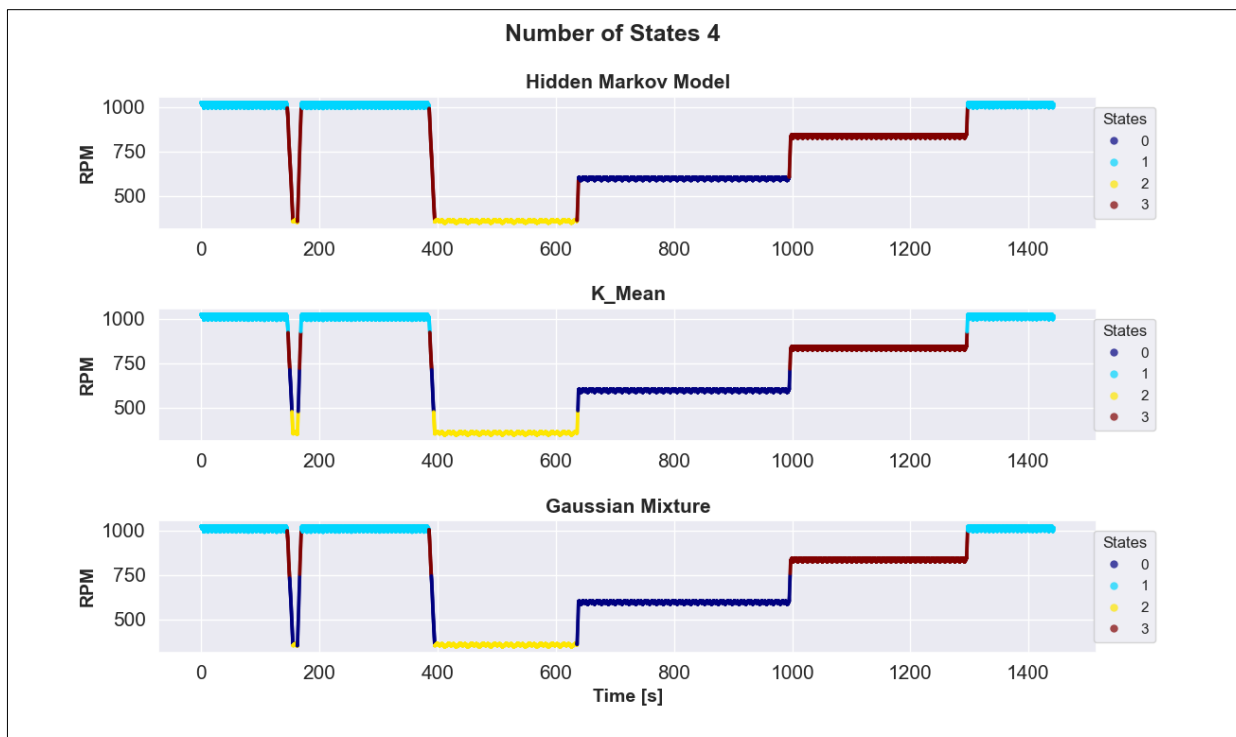


Figure 52: Comparison of all three models performed with four states (RPM vs. time)

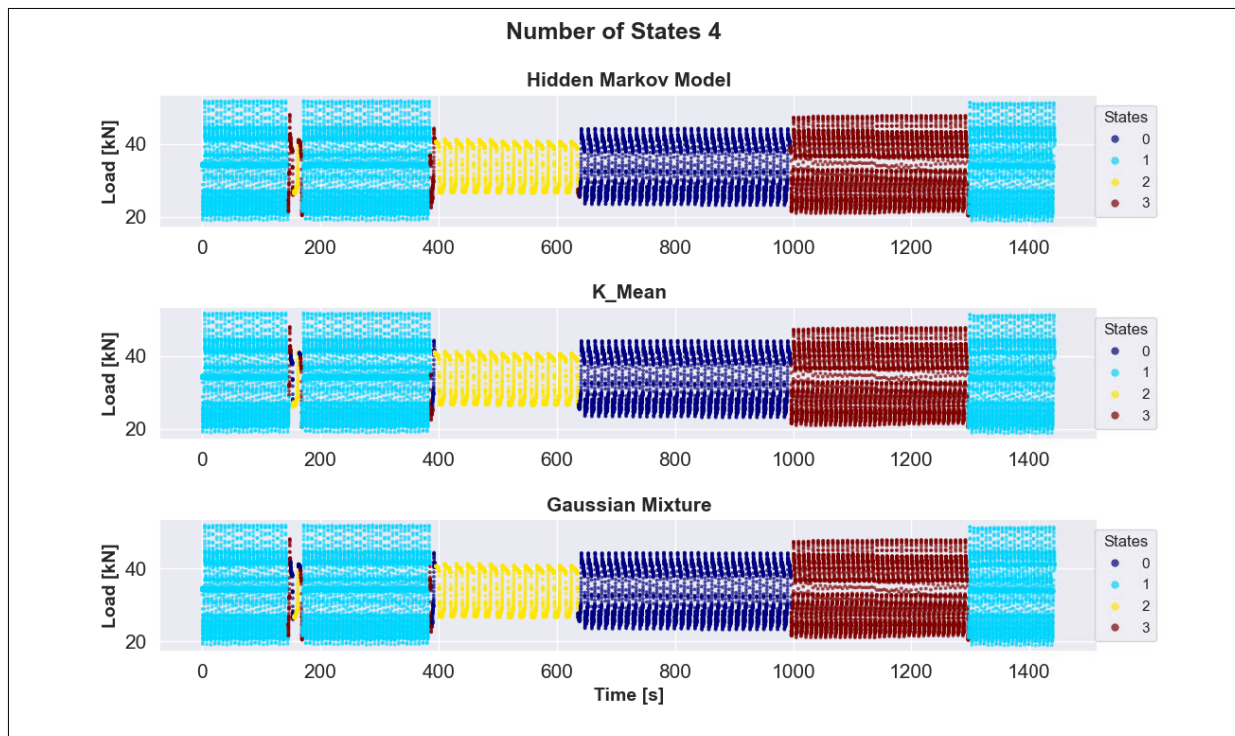


Figure 53: Comparison of all three models performed with four states (load vs. time)

In chapter 3, some methods such as `decode`, `score` etc. included in the HMM model in Python and their applications were mentioned. The method `Decode` represents the most likely sequence of (hidden states) based on an observation sequence as well as the probability (or rather Log-probability) of this state sequence (solution of the second problem). The method `Score` represents the probability of the observed sequence, marginalizing out all the possible hidden states, in other words, it gives the sum of probabilities of all (hidden) state sequences based on the given observation sequence (solution of the first problem). The results of both methods for each number of states are shown in figure 54. If we take a closer look at the probabilities, it can be seen that the results from both methods are more or less the same, especially for the model with four states, there is no large difference between probabilities from `decode` and `score`. This could mean that the probabilities of all other possible state sequences are so low – that the (hidden) state sequence represented by `decode` could be by far the most likely state sequence.

```

For n_states = 2 Decode LogProb is : -1084884.3219759762 Probability : 5.31450087340769e-471160
For n_states = 2 Score LogProb is : -1084883.4065224384 Probability : 1.32751336671884e-471159

For n_states = 3 Decode LogProb is : -995071.5063727576 Probability : 8.6234918030048e-432155
For n_states = 3 Score LogProb is : -995070.4685177765 Probability : 2.43454522299981e-432154

For n_states = 4 Decode LogProb is : -936579.7177767401 Probability : 3.95100024715302e-406752
For n_states = 4 Score LogProb is : -936577.4328040316 Probability : 3.88202292204472e-406751

For n_states = 5 Decode LogProb is : -898166.9656936205 Probability : 1.10400610077804e-390069
For n_states = 5 Score LogProb is : -897950.9325336892 Probability : 7.32792034064206e-389976

For n_states = 6 Decode LogProb is : -855183.2282390838 Probability : 4.395006085461853e-371402
For n_states = 6 Score LogProb is : -854939.550841678 Probability : 2.95602848430101e-371296

```

Figure 54: Log-Probabilities and probabilities from Python methods `decode` and `score`

Score and decode prefer calculating the log-probability because the results of multiplying a lot of numbers would be either too large to store in memory or too small that cannot be distinguished from zero, as was the case in the results of the example illustrated in figure 54.

Exactly the same approach is done with the selected part of the dataset, namely motor rotational speed, motor rotational acceleration, reducer rotational acceleration and polished rod acceleration. The silhouette analysis of all three models shows again that there exist four hidden states in the dataset, with a much better and higher silhouette coefficient than before when we did this analysis for the entire dataset (from $\sim 0,75$ to $\sim 0,90$). But the decisive difference is in models with 5 and 6 states. While K-Mean and GM only consider a higher chance for a model with four states, GMMHMM also considers 5- and 6-state models possible.

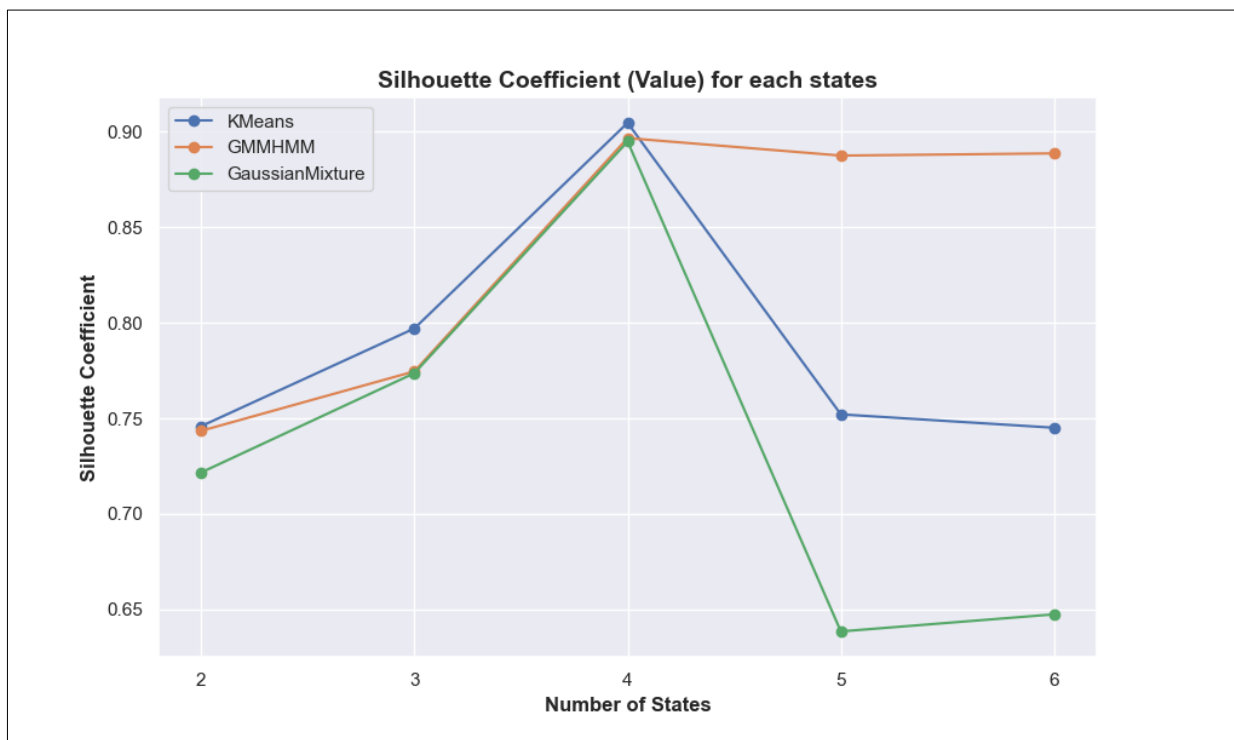


Figure 55: Silhouette coefficients for each model at each state (for selected dataset)

As shown in figures 56 and 57, all three models with four states show more or less the same results where they correctly recognized the states (especially visible in figure 55), with small differences in the transition part (especially visible in figure 56). There is no over- or underfitting found on any model. The illustrations for individual models at each state are shown in appendices.

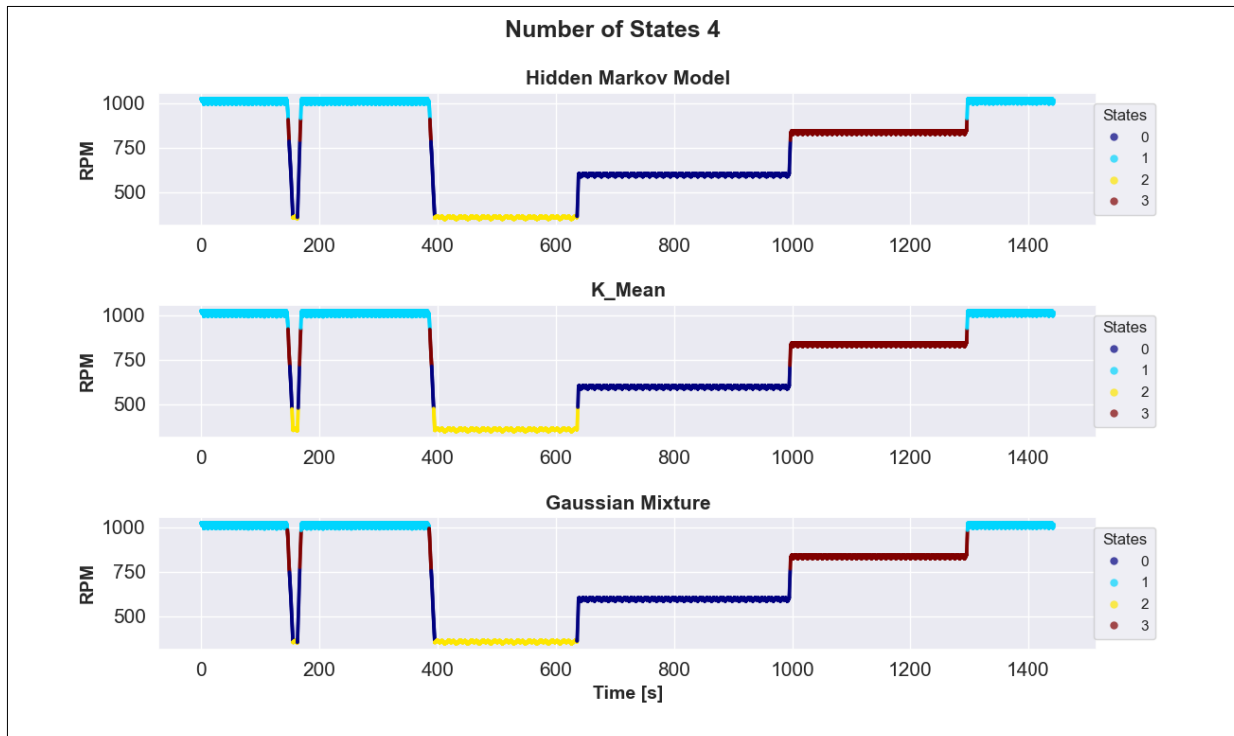


Figure 56: Comparison of all three models performed with 4 states (RPM vs. time)

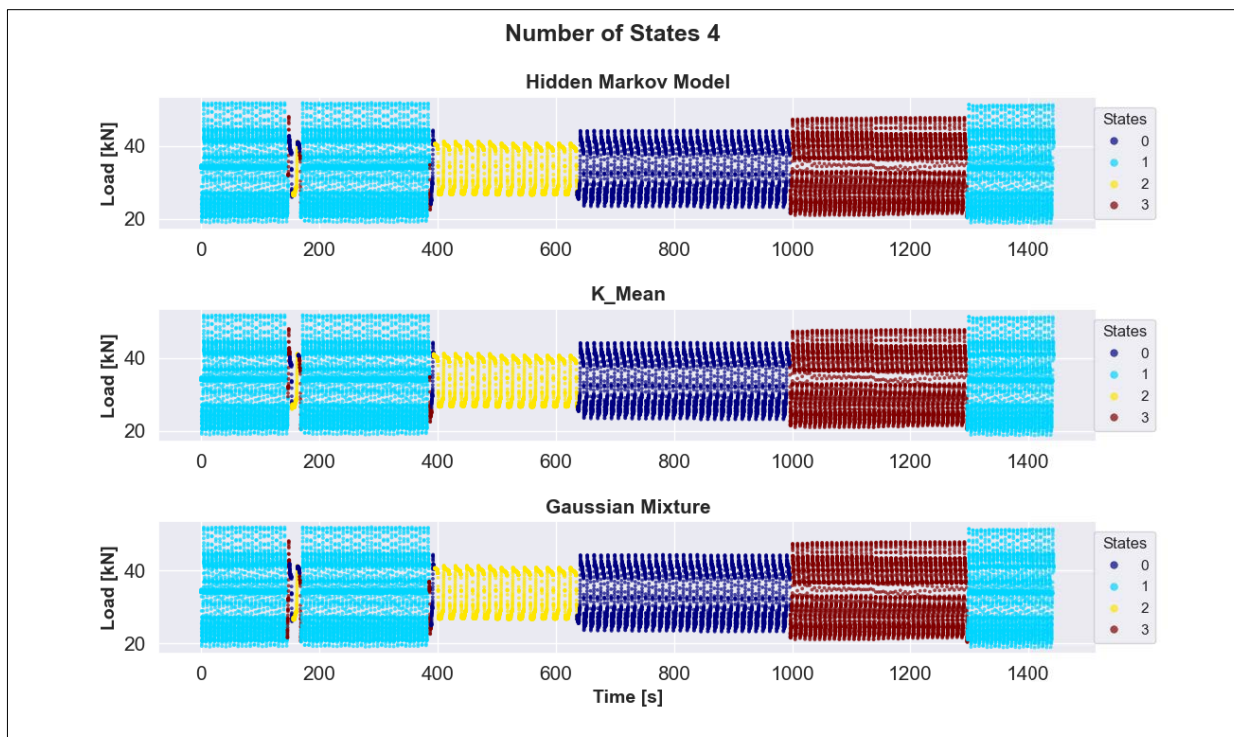


Figure 57: Comparison of all three models performed with 4 states (load vs. time)

The decisive difference between the three models is very clear when we run them with 5 and 6 states. In contrast to 4-state models, which provide more or less the same results for all three models, 5- and 6-state models show a clear difference.

Figures 58 and 59 show results from models run with five states. If we take a closer look at these figures, it can be seen clearly, while other two models still try to divide the samples into five groups, which leads to an overfitting, which is visible at the beginning (first 400 seconds)

and end (<1300 seconds) on an RPM of 1000, HMM has clearly recognized that in addition to the four main states, there is a transitional part (light blue part), and assigned these samples to a new state. Only GMMHMM recognized correctly that there are samples in the data which have other behavior than the other four main states.

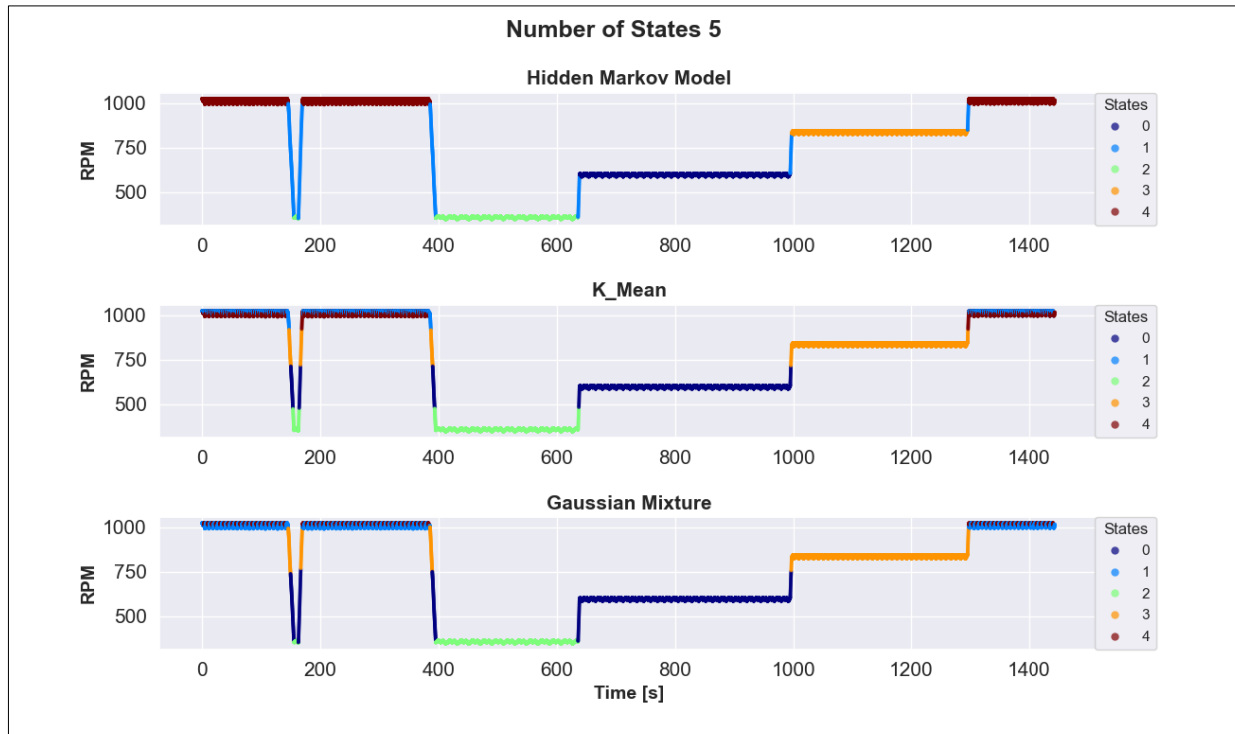


Figure 58: Comparison of all three models performed with five states (RPM vs. time)

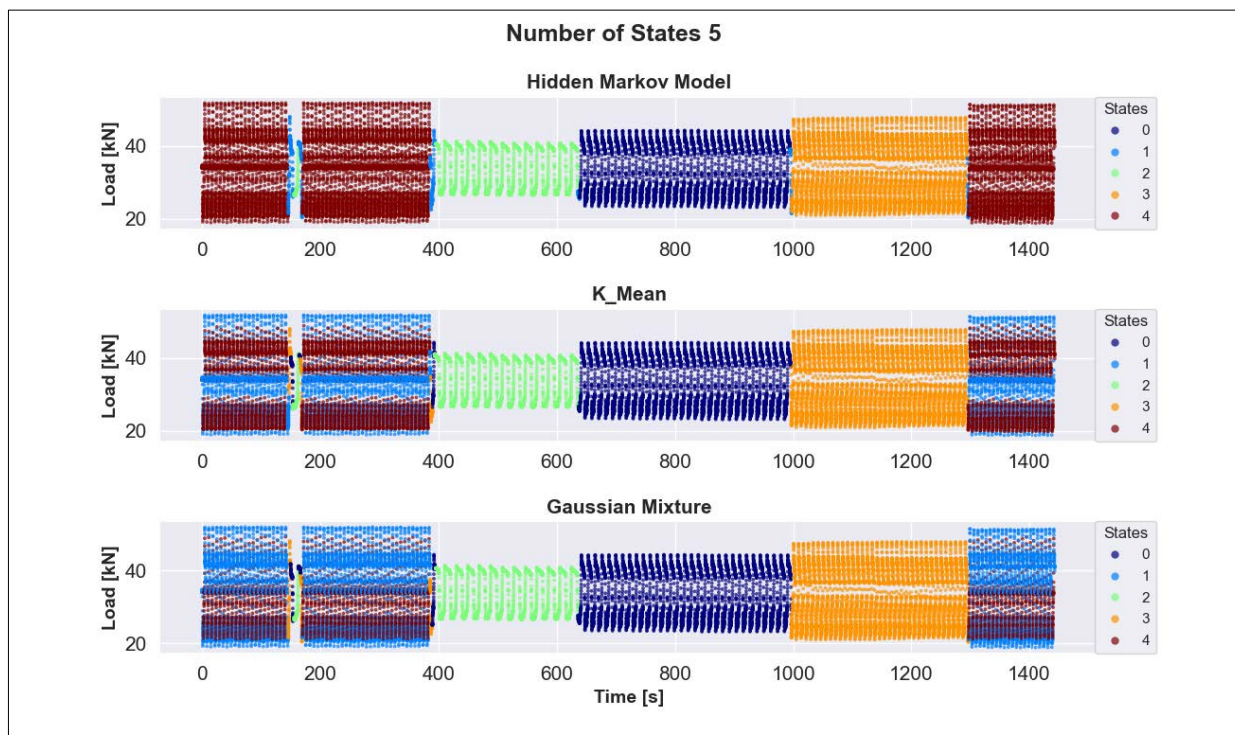


Figure 59: Comparison of all three models performed with five states (load vs. time)

Figures 60 and 61 show results of 6-states models. Here with six states also the same, while the other two models still try to divide the samples into six groups, which again leads to overfitting, HMM recognized that the transition part itself has two different behavior, one is: transition from a state with the higher value (in this case RPM and load) to a state with lower value, shown with the color dark blue, and second one is: transition from a state with the lower value to a state with higher value, shown with the color dark red.

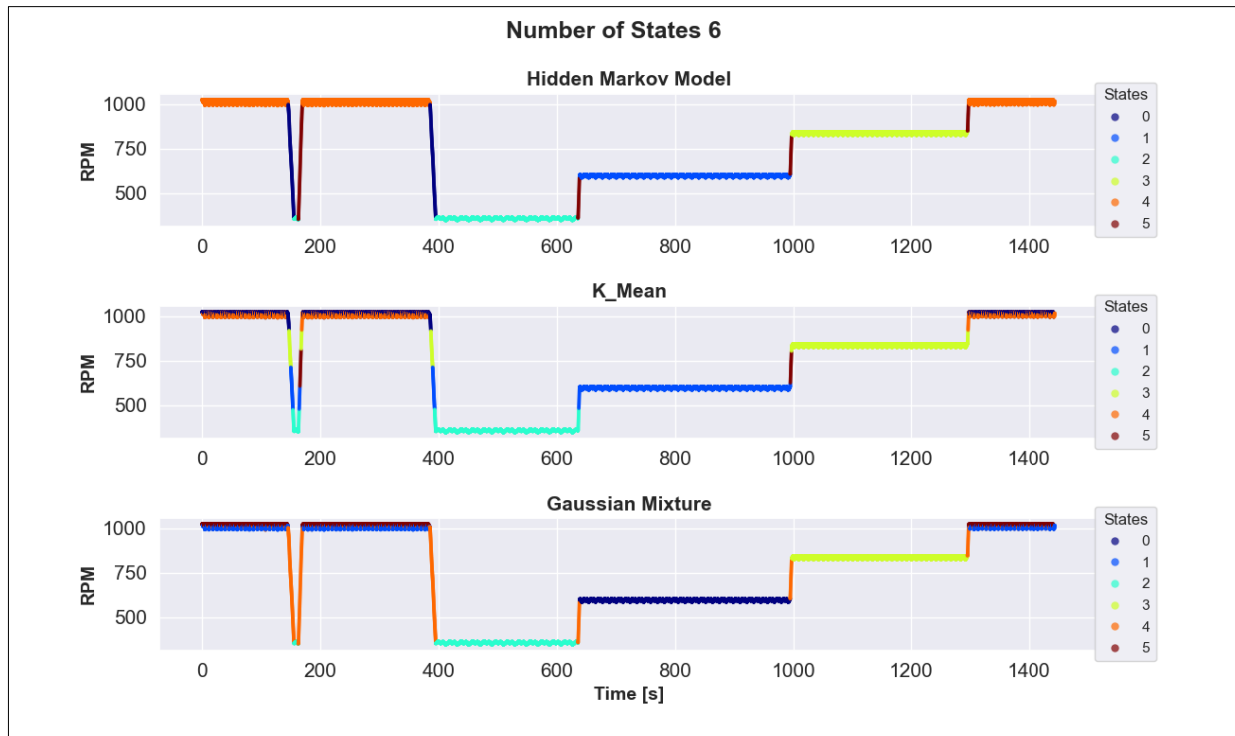


Figure 60: Comparison of all three models performed with six states (RPM vs. time)

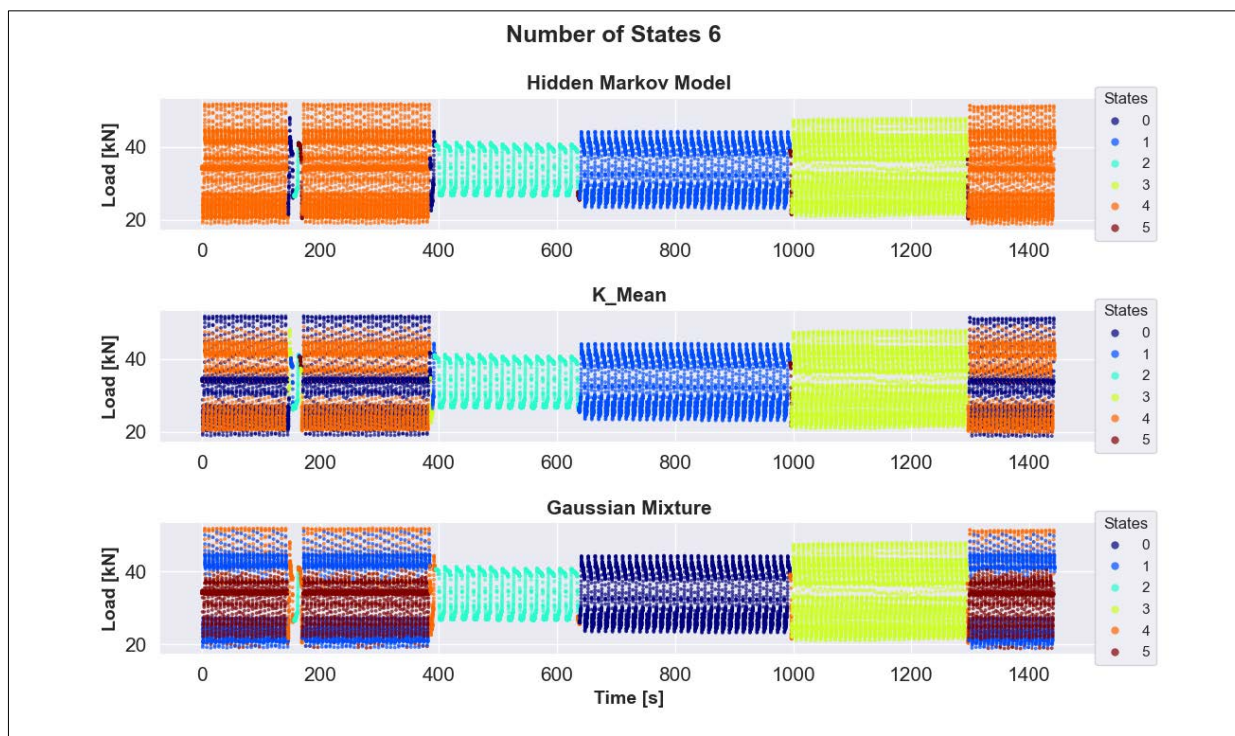


Figure 61: Comparison of all three models performed with six states (load vs. time)

These small but very precise differences will be very helpful to us in making predictions. The more details we know about our data, the more accurate the predictions are. That is why HMM can be of great and reliable help to us.

7 Conclusion

To avoid future problems, we have to recognize them earlier, therefore, we need to predict. To be able to predict, we need to know the states hiding in our data. If we are trying to find a method for relating an observation sequence to a sequence of hidden states, which explains the observations, HMM is a very good and reliable tool for this purpose.

Due to the efficient learning algorithms and the strong statistical foundation, HMM is able to learn directly from raw sequence data. This property and all other properties and advantages of HMM, which we discussed earlier, makes it possible to apply HMM in many fields, like data mining and classification, pattern recognition and so on.

HMMs are easy to implement but are a powerful model at extracting important information from the dataset. It is also easy to combine HMMs into libraries.

Furthermore, it is the case with almost all models that each iteration must access every point in the dataset, so the algorithm can be relatively slow as the number of samples increases. This is not the case with the HMMs, they can be performed for large number of samples.

All in all, the results presented in this thesis show clearly, that although HMM and Gaussian Mixture can be seen as an extension of the ideas behind K-Means, and although GMMHMM and Gaussian Mixture are the same in many respects, they nevertheless provide different results, where HMM supplies the best and accurate results.

In addition, as mentioned earlier in chapter 5, GMMs and HMMs are probabilistic models, K-Means non-probabilistic. GMM and HMM are both probabilistic models, nevertheless, it seems that HMM was the only one of the two models that not only considered the distances of samples to the neighbor but also the behavior of samples. The algorithms of the HMM make this decisive difference.

8 Publication bibliography

1. A Developer Diary (o. J.): Introduction to Hidden Markov Model. Edited by Abhisek Jana. Available online at <http://www.adeveloperdiary.com/data-science/machine-learning/introduction-to-hidden-markov-model/>, checked on 8/11/2021.
2. Analytics Vidhya (o. J.): Complete Guide to Expectation-Maximization Algorithm. Edited by Chirag Goyal. Available online at <https://www.analyticsvidhya.com/blog/2021/05/complete-guide-to-expectation-maximization-algorithm/>, checked on 9/28/2021.
3. Ankan, Ankur; Panda, Abinash (2018): Hands-On Markov Models with Python. Implement Probabilistic Models for Learning Complex Data Sequences Using the Python Ecosystem. Birmingham: Packt Publishing Ltd. Available online at <https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=5529464>.
4. Fink, Gernot A. (2014): Markov models for pattern recognition. From theory to applications. Second edition. London, Heidelberg, New York, Dordrecht: Springer (Advances in computer vision and pattern recognition).
5. hmmlearn (2010): Tutorial. Available online at <https://hmmlearn.readthedocs.io/en/latest/tutorial.html#available-models>, checked on 5/7/2022.
6. Influxdata Inc. (o. J.): Time series data. Available online at <https://www.influxdata.com/what-is-time-series-data/>, checked on 5/7/2022.
7. Langbauer, Clemens; Langbauer, Thomas; Fruhwirth, Rudolf; Mastobaev, B. (o. J.): Sucker rod pump frequency-elastic drive mode development – from the numerical model to the field test. Edited by JVE Journals. Available online at <https://www.extrica.com/article/22074>, checked on 5/7/2022.
8. Rabiner, Lawrence R. (1989): A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE 77 (2)*, pp. 257–286.
9. ScienceDirect (o. J.): Polished Rod. Available online at <https://www.sciencedirect.com/topics/engineering/polished-rod>, checked on 5/7/2022.
10. scikit-learn (2007): Selecting the number of clusters with silhouette analysis on KMeans clustering. Available online at https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html, checked on 5/7/2022.
11. Takacs, Gabor (2015): Sucker-Rod pumping handbook. Production engineering fundamentals and long-stroke Rod Pumping. Waltham, MA: Gulf Professional Publishing. Available online at <http://www.sciencedirect.com/science/book/9780124172043>.

12. VanderPlas, Jake (2017): Python data science handbook. Essential tools for working with data. First edition. Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly. Available online at <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=4746657>.
13. Visser, Ingmar; Speekenbrink, Maarten (o. J.): Dependent Mixture Models - Hidden Markov Models of GLMs and Other Distributions in S4. Package 'depmixS4'. Available online at <https://cran.r-project.org/web/packages/depmixS4/depmixS4.pdf>, checked on 5/7/2022.
14. Wikipedia (o. J.a): Bayes' Theorem. Available online at https://en.wikipedia.org/wiki/Bayes%27_theorem, checked on 9/28/2021.
15. Wikipedia (o. J.b): Conditional Probability. Available online at https://en.wikipedia.org/wiki/Conditional_probability, checked on 9/28/2021.

List of tables

Table 1: Conditional probability table (CPT)	15
Table 2: Transition Probability Matrix	21
Table 3: Emission Probability Matrix	23

List of figures

Figure 1: A simple Markov chain.....	14
Figure 2: Flow chart of an EM algorithm	18
Figure 3: States and observations diagram of the weather.....	20
Figure 4: States and the transition probabilities (A Developer Diary o. J.).....	21
Figure 5: States and the associated transition and emission probabilities (A Developer Diary o. J.).....	22
Figure 6: A summary of the hidden Markov model and its components	23
Figure 7: Observations vs states in form of a lattice (Rabiner 1989)	27
Figure 8: Trellis diagram for forward algorithm	27
Figure 9: Model parameters, states, observations and observed sequence.....	28
Figure 10: Assumptions, inputs and outputs of forward algorithm	29
Figure 11: Python code for forward algorithm (A Developer Diary o. J.).....	29
Figure 12: Alpha for each state at each time step t , and probability of obs. sequence $P(V \lambda)$	29
Figure 13: Trellis diagram for backward algorithm	31
Figure 14: Assumptions, inputs and outputs of Backward algorithm	32
Figure 15: Python code for Backward algorithm (A Developer Diary o. J.).....	32
Figure 16: Beta for each state and each time step t (7 days)	32
Figure 17: Assumptions, inputs and outputs of the Viterbi algorithm.....	36
Figure 18: Best state seq. and its probability predicted, using the Viterbi algorithm.....	36
Figure 19: Operations sequence needed to calculate joint event by a transition from S_i to S_j	37
Figure 20: States, observations and model parameters a , b and π	40
Figure 21: Assumptions, inputs and outputs of the Baum-Welch algorithm.....	40
Figure 22: a & b estimated by Baum-Welch.....	41
Figure 23: a & b initial	41
Figure 24: Predicted (hidden) states using a & b from Baum-Welch algorithm	41
Figure 25: Predicted (hidden) states using a & b initial.....	41
Figure 26: Sucker rod pumping system (Langbauer et al. o. J.).....	44
Figure 27: Posterior probabilities of first 10 seconds of polished rod position	48

Figure 28: Posterior probabilities of samples 40 to 60 of polished rod position	48
Figure 29: Values of posterior probabilities for the samples 40 to 60	48
Figure 30: Motor power and strokes per minute over time.....	49
Figure 31: Features recorded and collected from the sucker rod pump	49
Figure 32: Info about dataset after removing incomplete columns	50
Figure 33: Important information about dataset	50
Figure 34: Development of three most decisive features over time	51
Figure 35: Transition matrix using feature polished rod position.....	54
Figure 36: Polished rod position and the associated histogram and density function.....	55
Figure 37: Stroke duration and strokes per minute	55
Figure 38: Polished rod kinematic parameters vs. time	56
Figure 39: Start and end of up- and downstroke of 2 different cycles.....	57
Figure 40: BIC for 6 diff. components (states) and 4 diff. covariance type	58
Figure 41: Silhouette coefficients for each model at each state (for the entire dataset)	59
Figure 42: Silhouette coefficients values.....	60
Figure 43: K-Mean with three states	61
Figure 44: Gaussian Mixture (GMM) with three states.....	61
Figure 45: GMMHMM with three states	62
Figure 46: K-Mean with five states.....	62
Figure 47: Gaussian Mixture (GMM) with five states	63
Figure 48: GMMHMM with five states	63
Figure 49: K-Mean with four states	64
Figure 50: Gaussian Mixture (GMM) with four states.....	64
Figure 51: GMMHMM with four states	65
Figure 52: Comparison of all three models performed with four states (RPM vs. time).....	65
Figure 53: Comparison of all three models performed with four states (load vs. time)	66
Figure 54: Log-Probabilities and probabilities from Python methods decode and score	66
Figure 55: Silhouette coefficients for each model at each state (for selected dataset).....	67
Figure 56: Comparison of all three models performed with 4 states (RPM vs. time).....	68
Figure 57: Comparison of all three models performed with 4 states (load vs. time)	68
Figure 58: Comparison of all three models performed with five states (RPM vs. time)	69

Figure 59: Comparison of all three models performed with five states (load vs. time).....	69
Figure 60: Comparison of all three models performed with six states (RPM vs. time).....	70
Figure 61: Comparison of all three models performed with six states (load vs. time).....	70
Figure 62: Viterbi algorithm	80
Figure 63: Baum-Welch algorithm.....	80
Figure 64: K-Mean with two states (for the entire dataset)	81
Figure 65: Gaussian Mixture (GMM) with two states (for the entire dataset).....	81
Figure 66: GMMHMM with two states (for the entire dataset).....	82
Figure 67: K-Mean with six states (for the entire dataset).....	82
Figure 68: Gaussian Mixture (GMM) with six states (for the entire dataset)	83
Figure 69: GMMHMM with six states (for the entire dataset).....	83
Figure 70: Gaussian Mixture (GMM) with four states (for selected dataset).....	84
Figure 71: GMMHMM with four states (for selected dataset).....	84
Figure 72: K-Mean with four states (for selected dataset)	85
Figure 73: Gaussian Mixture (GMM) with five states (for selected dataset)	85
Figure 74: GMMHMM with five states (for selected dataset)	86
Figure 75: K-Mean with five states (for selected dataset).....	86
Figure 76: Gaussian Mixture (GMM) with six states (for selected dataset).....	87
Figure 77: GMMHMM with six states (for selected dataset)	87
Figure 78: K-Mean with six states (for selected dataset)	88

Abbreviations

SRP	Sucker Rod Pump
HMM	Hidden Markov Model
GMM	Gaussian Mixture Model
GMMHMM	Gaussian Mixture Hidden Markov Model
VSD	Variable Speed Drive System
FC	Frequency Converters
CPT	Conditional Probability Table
BIC	Bayesian Information Criterion
AIC	Akaike Information Criterion
PR	Polished Rod
NLP	Natural Language Processing

Appendices

Appendix A

```

def viterbi(V, a, b, initial_distribution):
    T = V.shape[0]
    M = a.shape[0]

    omega = np.zeros((T, M))
    omega[0, :] = np.log(initial_distribution * b[:, V[0]])

    prev = np.zeros((T - 1, M))

    for t in range(1, T):
        for j in range(M):
            # Same as Forward Probability
            probability = omega[t - 1] + np.log(a[:, j]) + np.log(b[j, V[t]])

            # This is our most probable state given previous state at time t (1)
            prev[t - 1, j] = np.argmax(probability)

            # This is the probability of the most probable state (2)
            omega[t, j] = np.max(probability)

    # Path Array
    S = np.zeros(T)

    # Find the most probable last hidden state
    last_state = np.argmax(omega[T - 1, :])

    S[0] = last_state

    backtrack_index = 1
    for i in range(T - 2, -1, -1):
        S[backtrack_index] = prev[i, int(last_state)]
        last_state = prev[i, int(last_state)]
        backtrack_index += 1

    # Flip the path array since we were backtracking
    S = np.flip(S, axis=0)

    # Convert numeric values to actual hidden states
    result = []
    for s in S:
        s = int(s)
        result.append(States[s])
    return result

```

Figure 62: Viterbi algorithm

```

def baum_welch(V, a, b, initial_distribution, n_iter=20):
    M = a.shape[0]
    T = len(V)

    for n in range(n_iter):
        alpha = forward(V, a, b, initial_distribution)
        beta = backward(V, a, b)

        xi = np.zeros((M, M, T - 1))
        for t in range(T - 1):
            denominator = np.dot(np.dot(alpha[t, :].T, a) * b[:, V[t + 1]].T, beta[t + 1, :])
            for i in range(M):
                numerator = alpha[t, i] * a[i, :] * b[:, V[t + 1]].T * beta[t + 1, :].T
                xi[i, :, t] = numerator / denominator

        gamma = np.sum(xi, axis=1)
        a = np.sum(xi, 2) / np.sum(gamma, axis=1).reshape((-1, 1))

        # Add additional T'th element in gamma
        gamma = np.hstack((gamma, np.sum(xi[:, :, T - 2], axis=0).reshape((-1, 1))))

        K = b.shape[1]
        denominator = np.sum(gamma, axis=1)
        for l in range(K):
            b[:, l] = np.sum(gamma[:, V == l], axis=1)

        b = np.divide(b, denominator.reshape((-1, 1)))

    return a, b

```

Figure 63: Baum-Welch algorithm

Appendix B

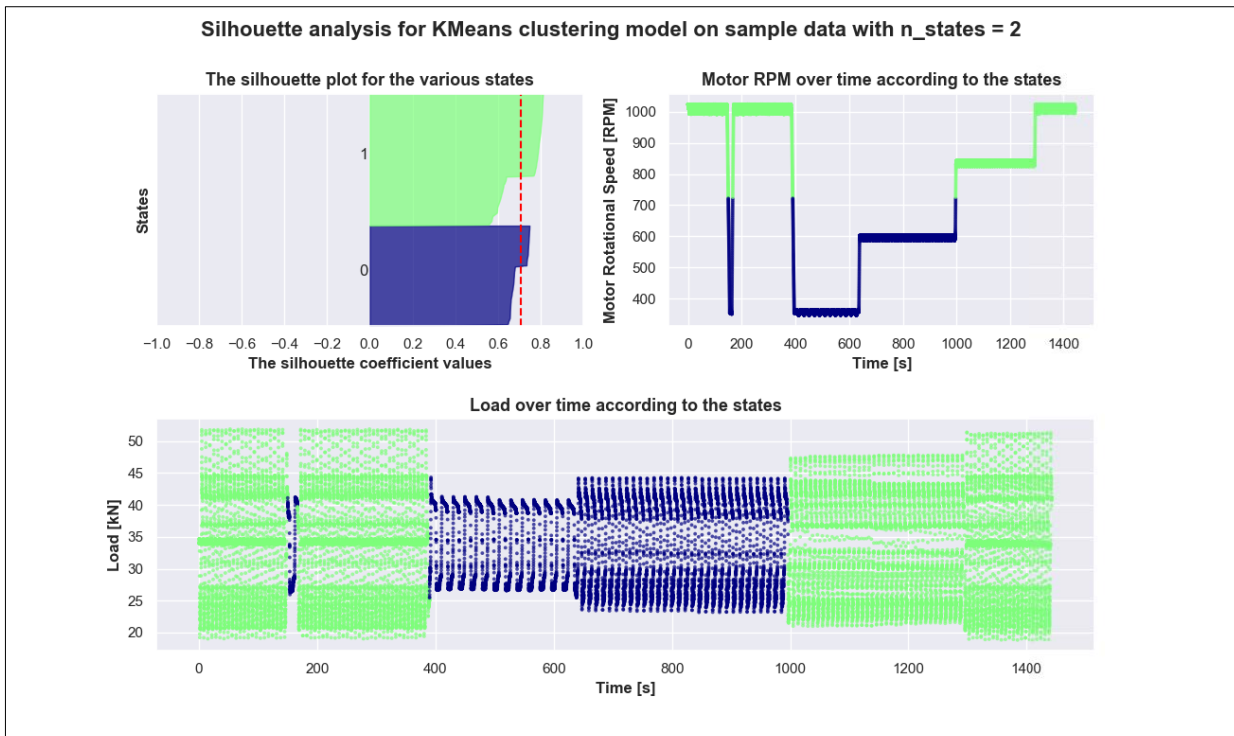


Figure 64: K-Mean with two states (for the entire dataset)

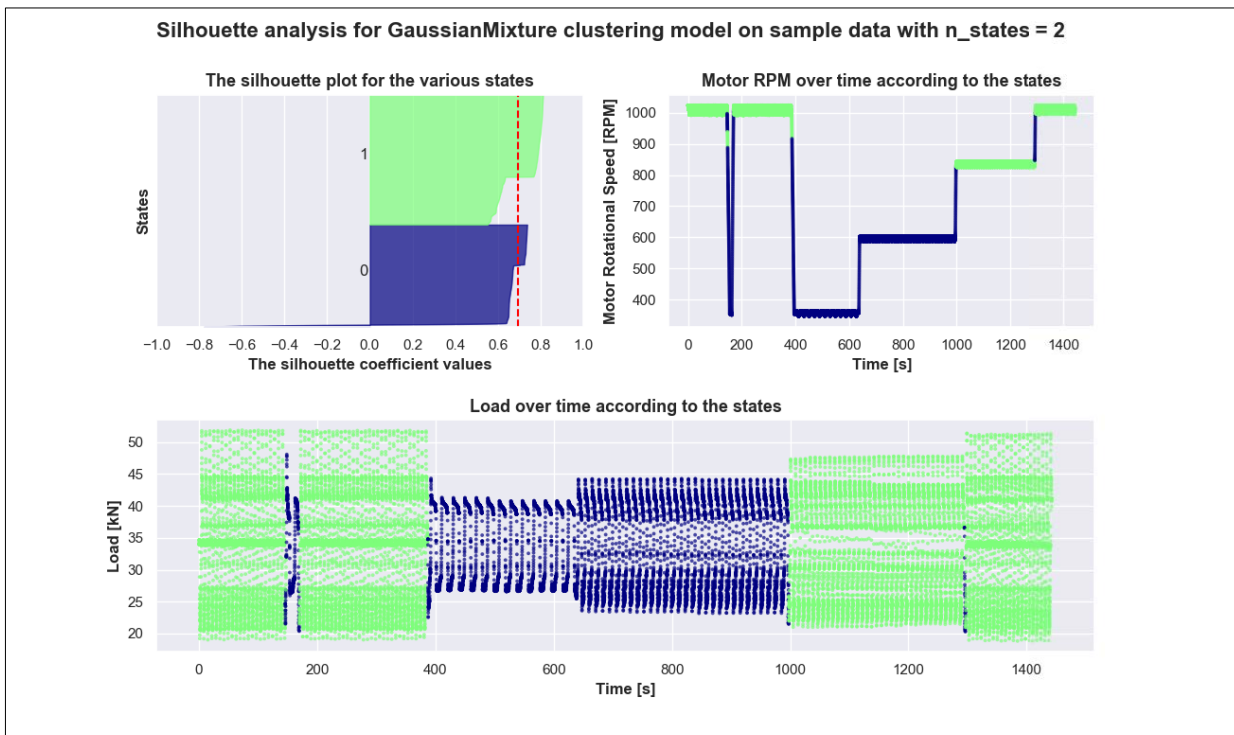


Figure 65: Gaussian Mixture (GMM) with two states (for the entire dataset)

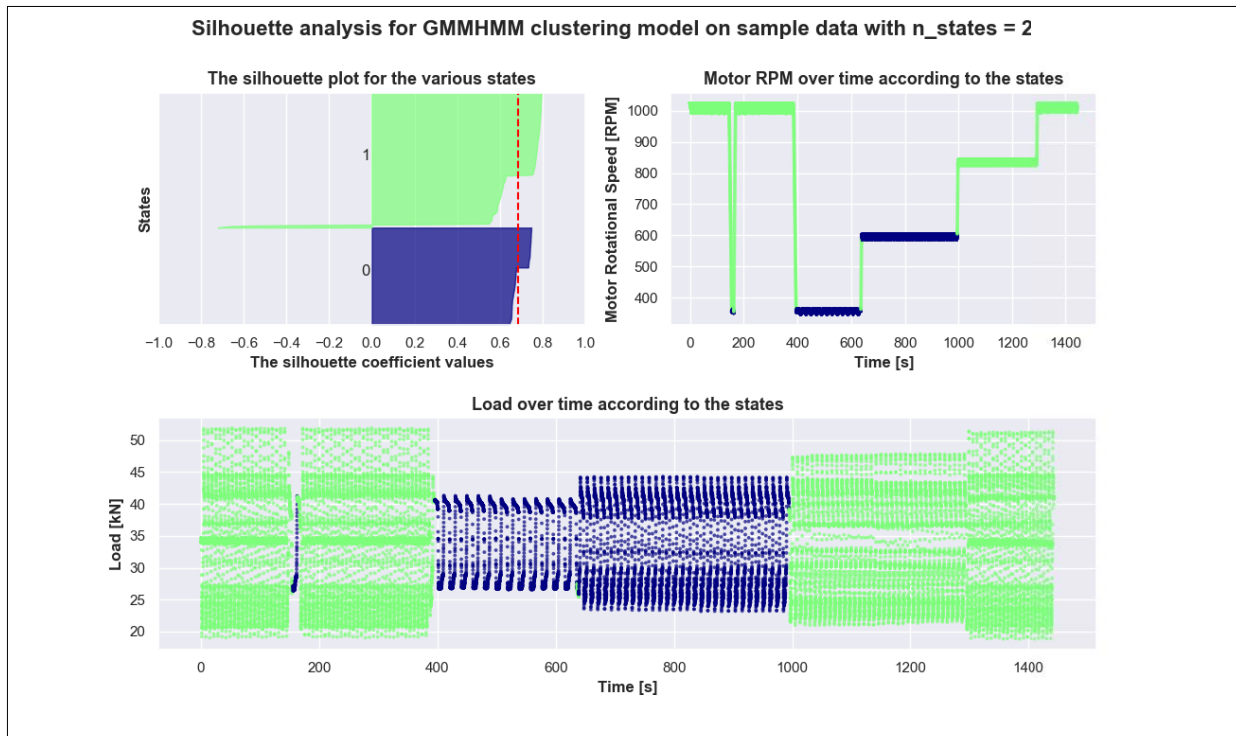


Figure 66: GMMHMM with two states (for the entire dataset)

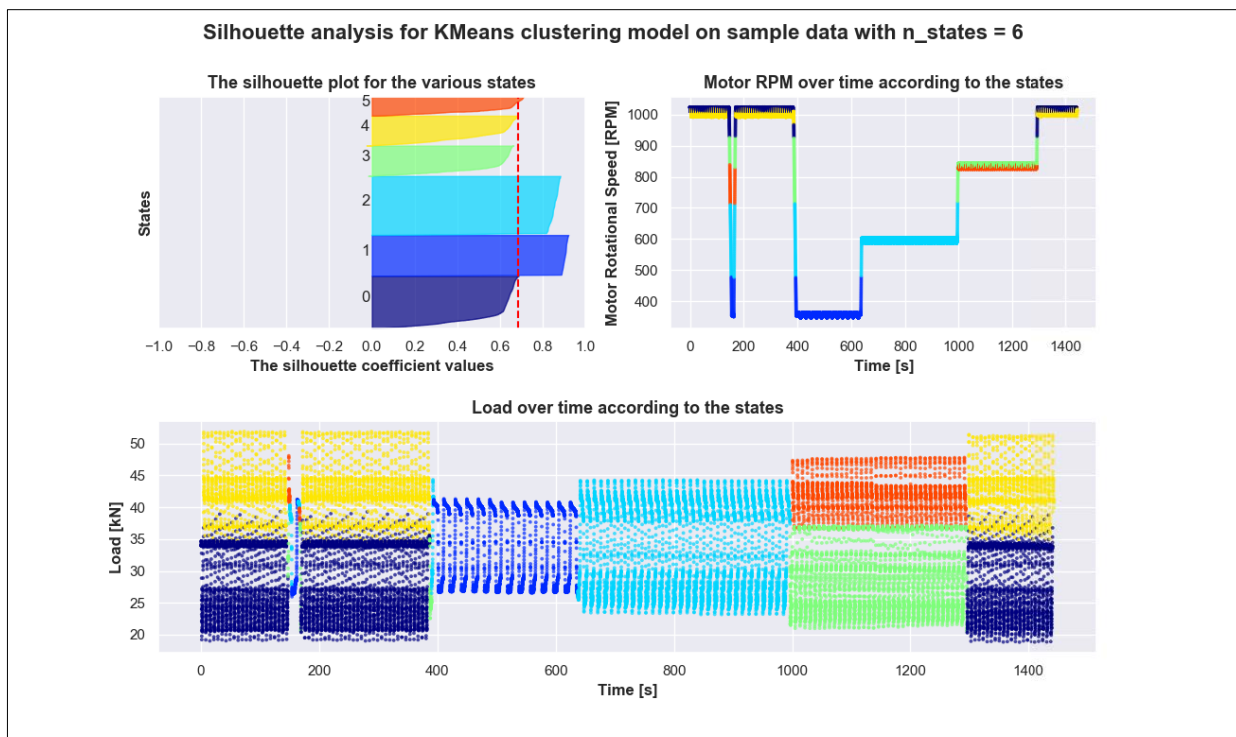


Figure 67: K-Mean with six states (for the entire dataset)

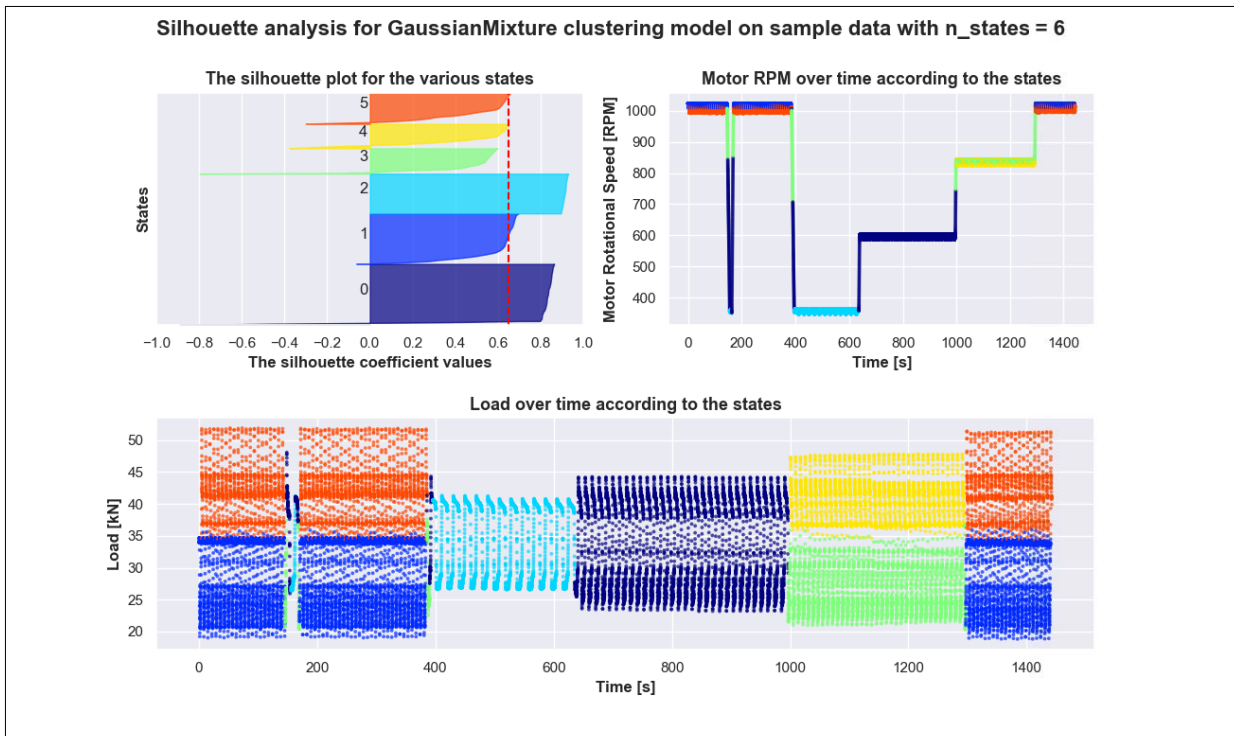


Figure 68: Gaussian Mixture (GMM) with six states (for the entire dataset)

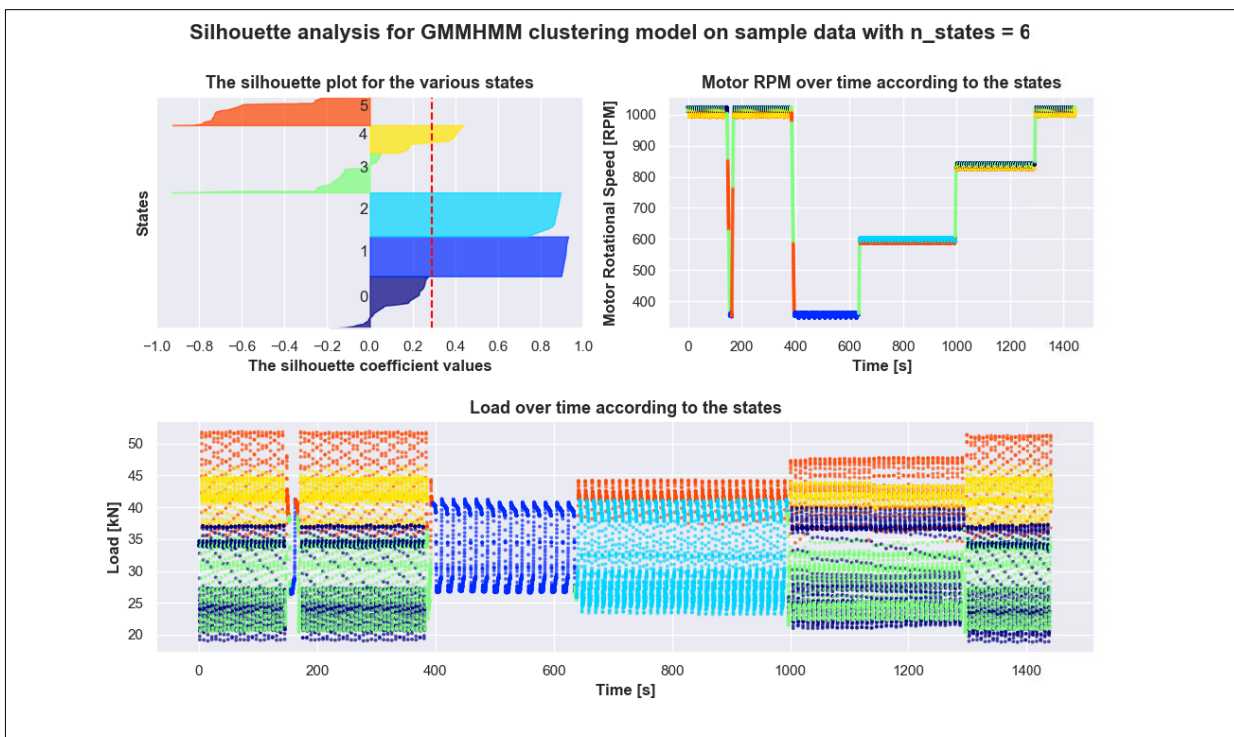


Figure 69: GMMHMM with six states (for the entire dataset)

Appendix C

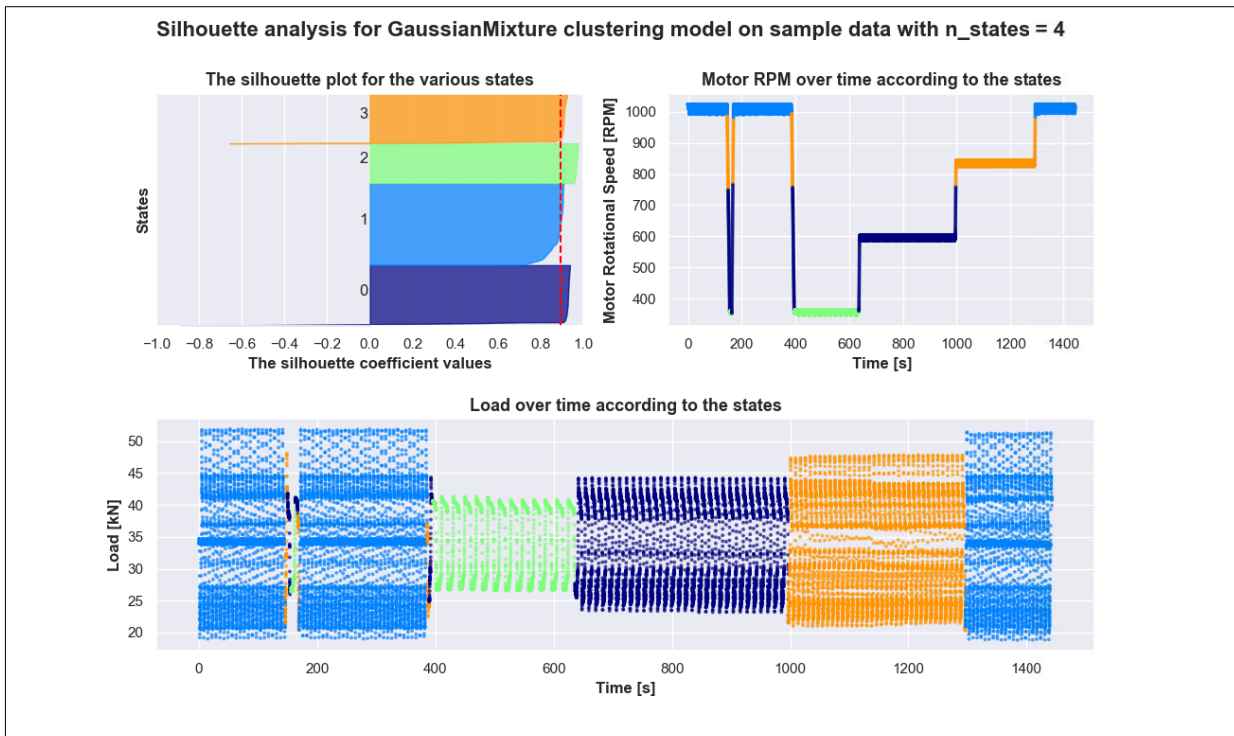


Figure 70: Gaussian Mixture (GMM) with four states (for selected dataset)

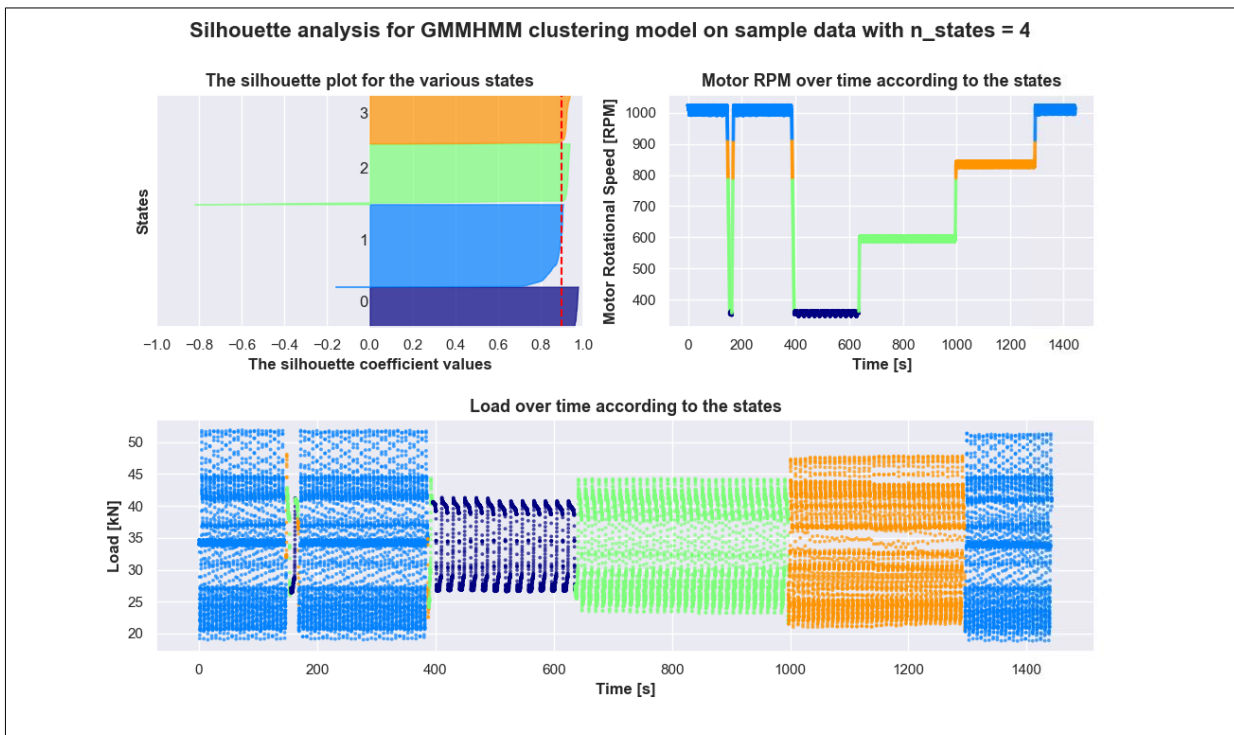


Figure 71: GMMHMM with four states (for selected dataset)

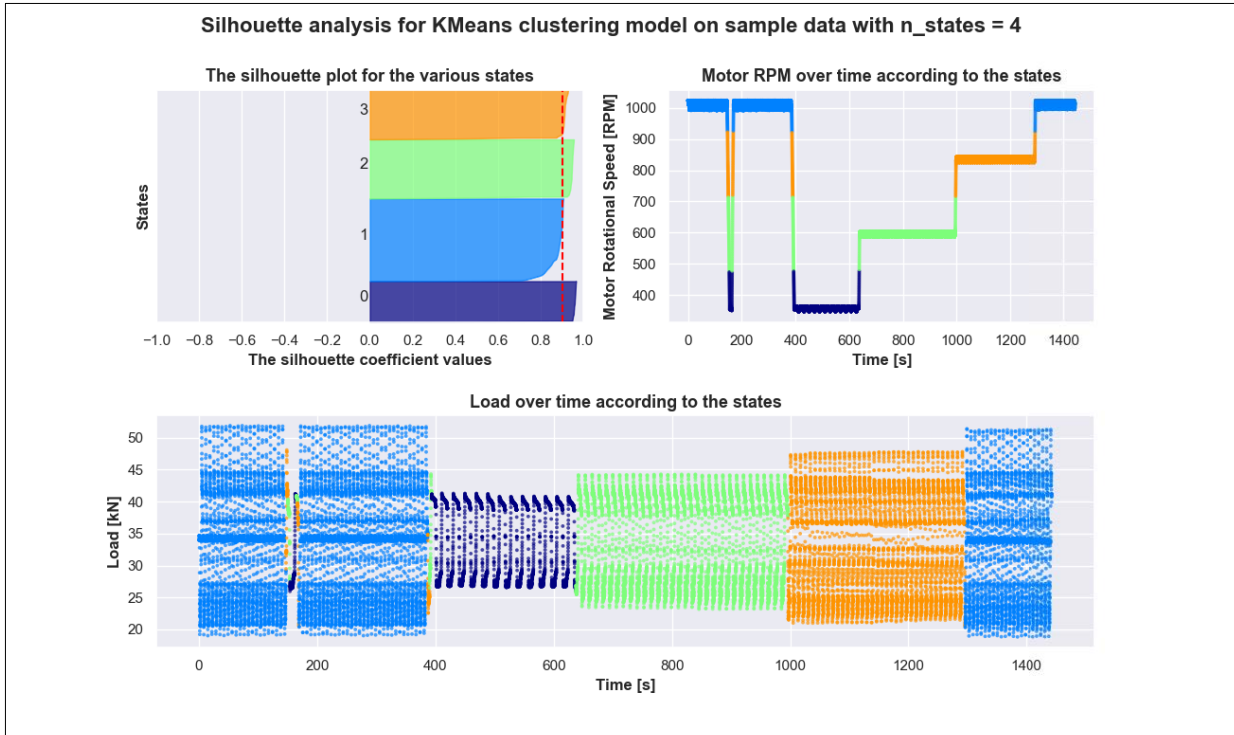


Figure 72: K-Mean with four states (for selected dataset)

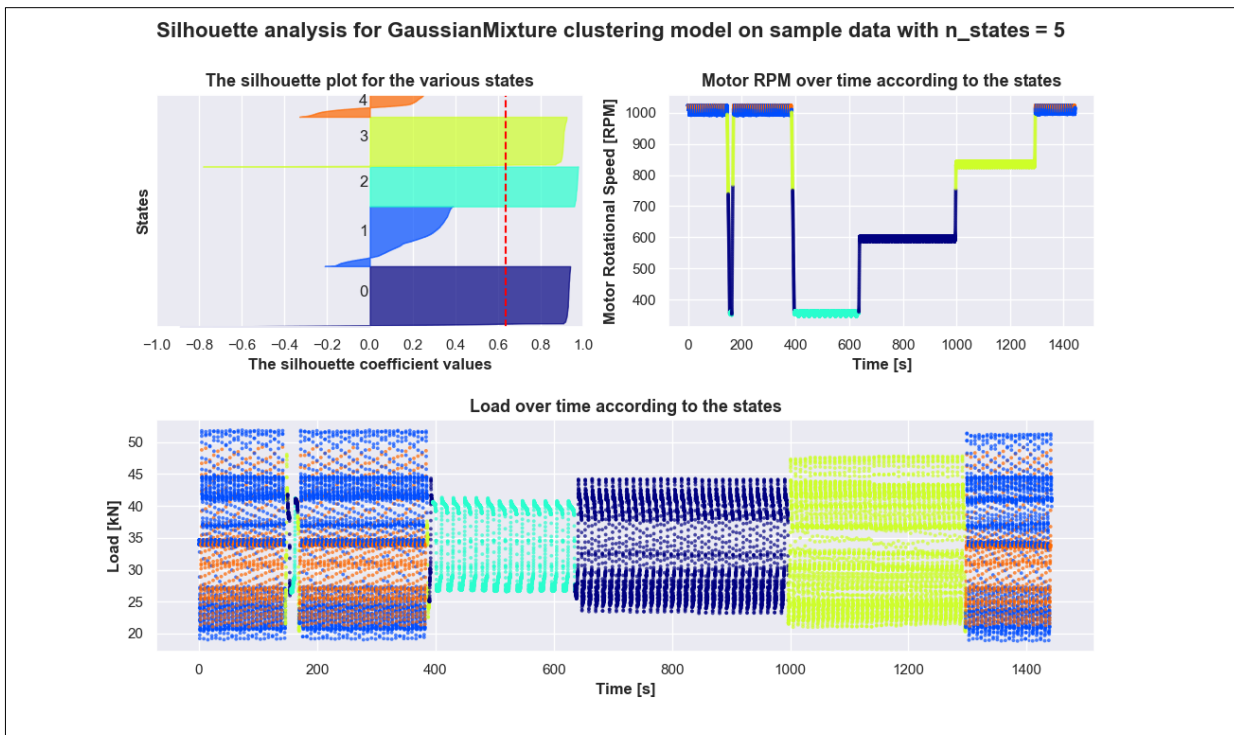


Figure 73: Gaussian Mixture (GMM) with five states (for selected dataset)

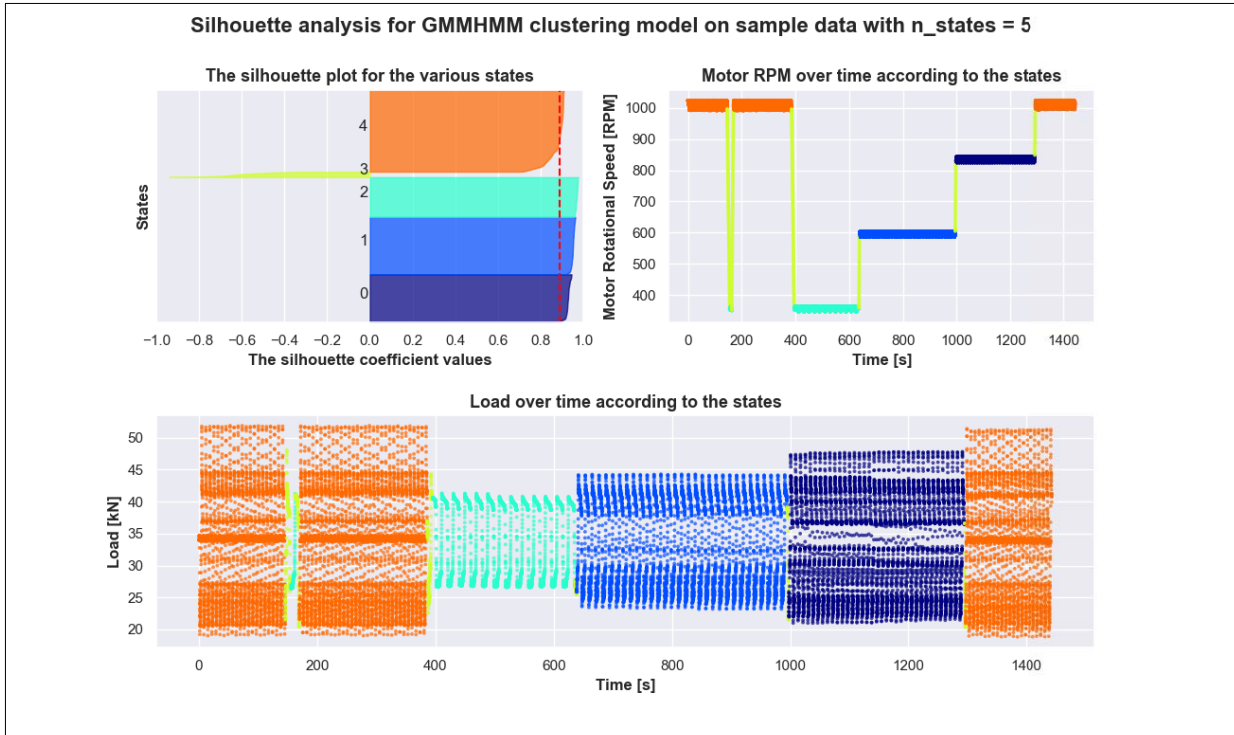


Figure 74: GMMHMM with five states (for selected dataset)

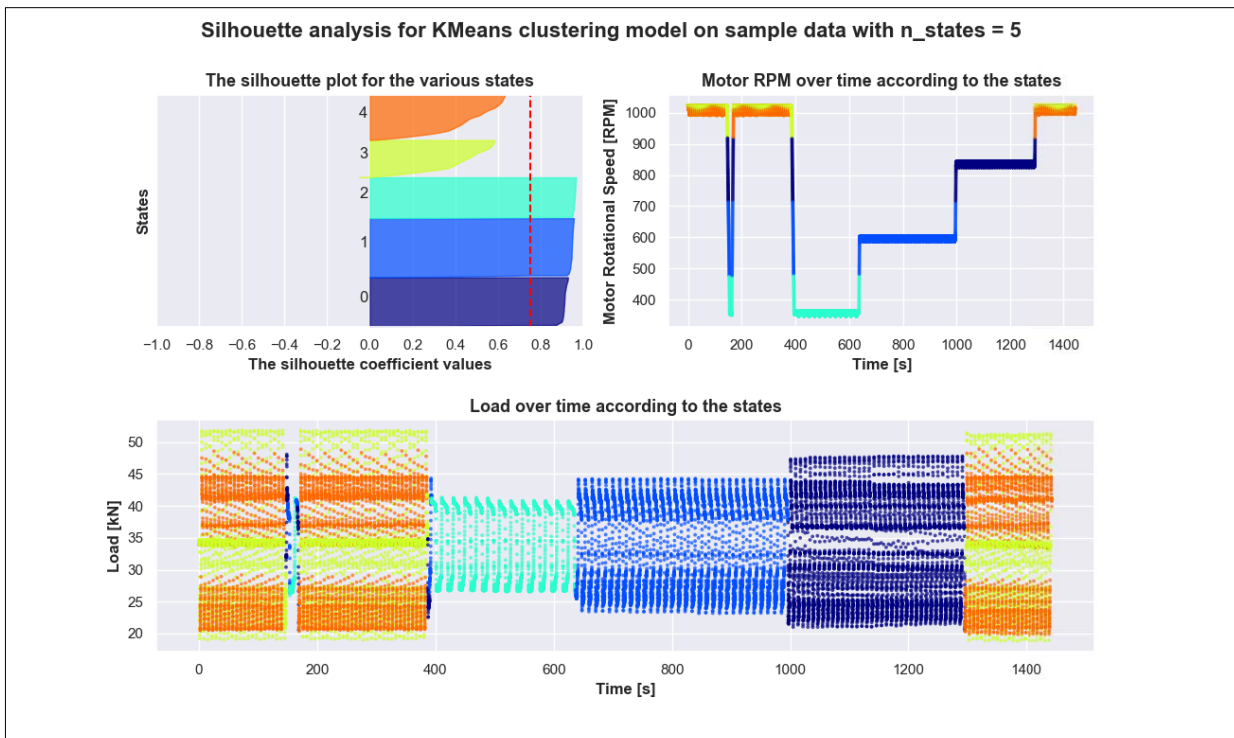


Figure 75: K-Mean with five states (for selected dataset)

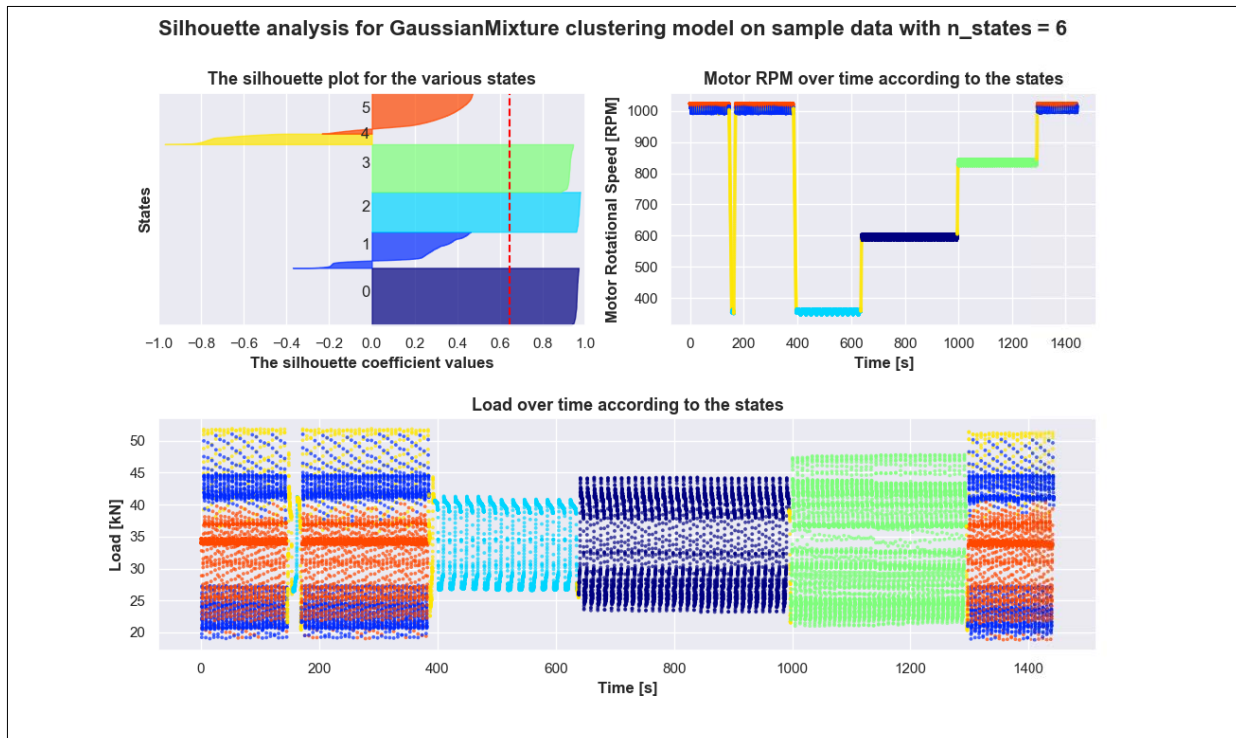


Figure 76: Gaussian Mixture (GMM) with six states (for selected dataset)

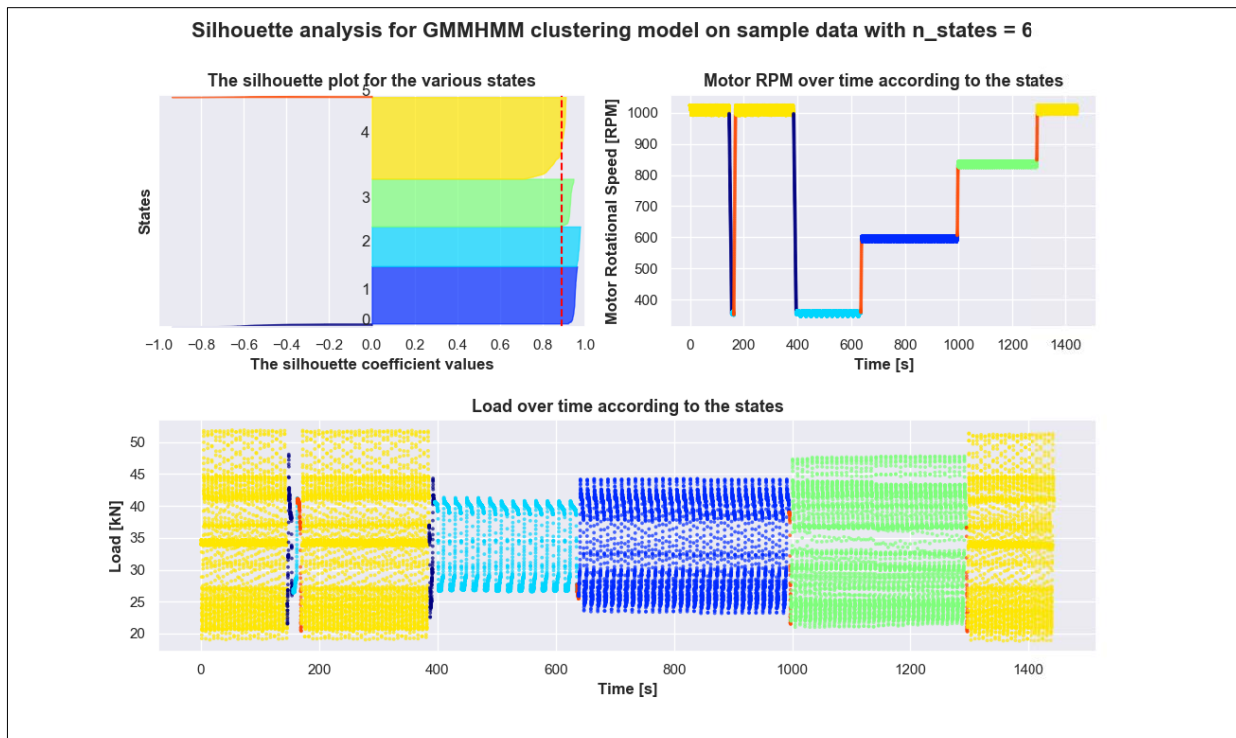


Figure 77: GMMHMM with six states (for selected dataset)

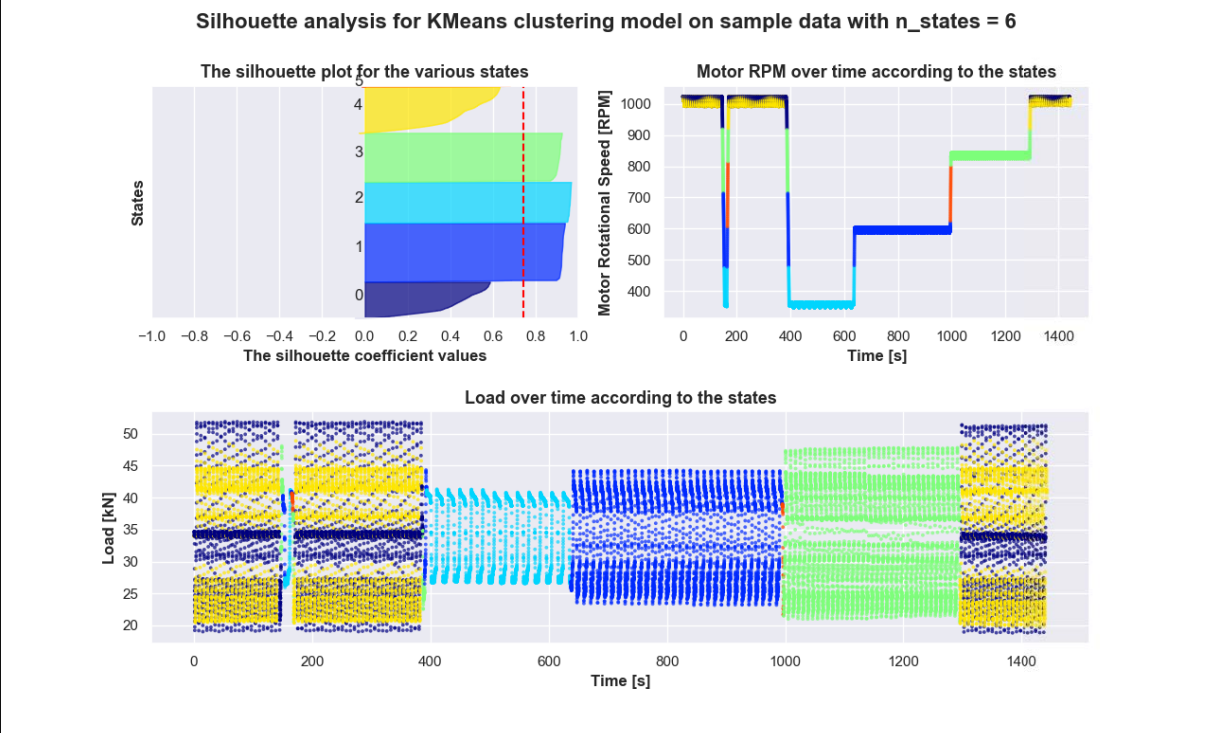


Figure 78: K-Mean with six states (for selected dataset)