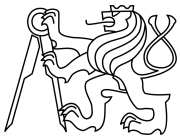
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Learning methods for continuous-time hidden Markov models
Student:	Bc. Lukáš Lopatovský
Supervisor:	Ing. Daniel Vašata, Ph.D.
Study Programme:	Informatics
Study Branch:	System Programming
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

Hidden Markov model (HMM) is a discrete-time random process, where the output variables depend on the states of an underlying discrete Markov chain that is usually unobserved. In many applications, however, discrete-time models are not ideal and continuous-time models can better describe the real process.

In continuous-time hidden Markov model (CT-HMM) it is assumed that the underlying Markov chain is continuous instead of discrete. It turns out that for CT-HMMs the task of parameter estimation is much more complicated than for discrete HMMs. A general EM framework for continuous-time dynamic Bayesian networks can be used but there is a need for efficient CT-HMM learning methods that can scale to large state spaces (hundreds or more of states).

The goal of the thesis is to:

1) describe existing CT-HMM learning methods,

2) compare them analytically from an efficiency point of view,

3) implement two most efficient methods and compare them numerically on a suitable toy model.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 29, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Master's thesis

Learning Methods for Continuous-Time Hidden Markov Models

Bc. Lukáš Lopatovský

Supervisor: Ing. Daniel Vašata, Ph.D.

9th May 2017

Acknowledgements

I would like to express my gratitude to my supervisor Ing. Daniel Vašata, Ph.D. for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore, I would like to thanks to Ing. Tomáš Šabata for introducing me to this interesting topic and to the Datamole Ltd. for support of open-source development.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on 9th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Lopatovský, Lukáš. *Learning Methods for Continuous-Time Hidden Markov Models*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017. Also available from: ⟨`https://github.com/lopatovsky/DP`⟩.

Abstrakt

Skrytý Markovův proces se spojitým časem je slibným modelem s využitím nejen pro biomedicínský výzkum. Nedostatek efektivních algoritmů pro jeho učení v minulosti výrazne omezoval jeho použití. Nedávno však byly prezentovány nové efektivní metody založené na EM algoritmu. V této diplomové práci zkoumáme a srovnáváme současné moderní metody, které jsou schopné vycvičit modely obsahující až stovky skrytých stavů. Jako součást práce jsme vyvinuli univerzální knihovnu pro skrytý Markovův proces se spojitým a diskrétním časem, která efektivně implementuje nejslibnější učební metody. Knihovna je snadno použitelná a dostupná všem uživatelům pod licencí open-source.

Klíčová slova skrytý Markovův proces, skrytý Markovův proces se spojitým časem, strojové učení.

Abstract

The continuous-time hidden Markov model is promising not only for the biomedical research. The lack of efficient learning algorithms has limited its use in the past. However, recently the new efficient EM approaches were presented. In this thesis we are examining and comparing current state-of-the-art methods that are able to train models containing hundreds of hidden states.

As the part of the work we have developed the general purpose continuous-time and discrete-time hidden Markov model library effectively implementing the best performing learning methods that is easy to use and available for everyone under open-source license.

Keywords HMM, CT-HMM, hidden Markov model, continuous-time hidden Markov model, EM, HMMs, machine learning.

Contents

Introduction **1**

1 Probabilistic Models **3**
1.1 Discrete-time Markov Process 3
1.2 Discrete-Time Hidden Markov Model 4
1.3 Continuous-Time Markov Process 5
1.4 Continuous-Time Hidden Markov Model 9

2 Statistics **11**
2.1 Expectation-Maximization Algorithm 11
2.2 Discrete-Time Hidden Markov Model 12
2.3 Continuous-Time Hidden Markov Model 17
2.4 Algorithms for CT-HMM parameters estimation 22
2.5 Complexity 24

3 Implementation **27**
3.1 About Library 27
3.2 Library Functionality 28
3.3 Implementation Details 29

4 Experiments **31**
4.1 Discrete-time vs. Continuous-time Hidden Markov Model . . . 32
4.2 Computational Complexity 36
4.3 Numerical Stability 40
4.4 Soft vs. Hard method 42
4.5 Hidden States Number 44

Conclusion **51**
Future Work 52

Bibliography **53**

A Acronyms **57**

B Contents of enclosed CD **59**

List of Figures

4.1 Models convergences 34
4.2 Average of the convergences from various random starting points . 34
4.3 Average of the convergences for various generated datasets 35
4.4 Average of the convergences for various generated datasets (zoom) 35
4.5 Time complexity of two algorithm variants and their subparts . . 38
4.6 Time-complexity of two algorithm variants and their subparts . . . 39
4.7 Relative distances of jump-rates matrices for data generated by exponential distribution with variable parameter λ 41
4.8 Probability of the testing dataset being generating by the models . 43
4.9 Performance ratio of the most probable state sequence 44
4.10 Schema of the model "birth and death" 45
4.11 Performance of the models with variable hidden states number trained by the *small* dataset 47
4.12 Performance of the models with variable hidden states number trained by the *big* dataset 48
4.13 Time demands of single iteration 49
4.14 Convergences for full and sparse matrices with variable number of hidden states 50

Introduction

A continuous-time hidden Markov model (CT-HMM) is a variant of hidden Markov model (HMM) where the output variables depend on the underlaying continuous-time Markov process. This differentiates it from the traditional HMM which is defined over discrete-time.

The CT-HMM model can be used on datasets with irregular observation times where the state transitions are unobserved and usually occur in between of two observations. This makes the model stronger and applicable to much bigger set of problems comparing to discrete-time HMM which use is restricted to equidistant time sequences. The most of the research is currently performed over medical data. Patients undergo the examinations irregularly, while their inner health state could change multiple times in between of two visits of medical center. Use of the discrete-time model was insufficient. Quantification of time leads to the emission sequence where most of the observations are void. It also does not track possible changes of the states in between of two observations. The CT-HMM can effectively solve these issues. It was used successfully for prediction of disease interaction [16], modeling of glaucoma progression [18, 19], and exploratory analyses of Alzheimer's Disease [19]. Besides the medical field it was also used to predict the read and write operations arriving at a flash memory [26].

The cost for the higher flexibility of the model is more complex inference procedure. There are not just unknown hidden states in observation points but also unknown moments of state transition. The model parameter estimation showed to be a hard problem to solve. First attempts based on the direct maximizing of data likelihood and summing the probability through all possible hidden states have only worked for restricted number of hidden states or small number of allowed states transitions [26, 13]. There was a need for more effective learning algorithm with less limitations [18].

The expectation maximization (EM) framework for the continuous-time Markov process and Bayesian networks in general was introduced in [22]. It was later extended to CT-HMM with the first proposed efficient EM algorithm

in [19]. The method is based on the computation of expected state transition and expected time spend in the state. It leads to the problem of solving integrals of matrix exponentials which is the most computationally demanding part of the method. Various approaches were used to solve it. Eigen decomposition (referred to as *Eigen*) was used in [25] . However according [19] it is not applicable for general jump-rate matrices which makes it unusable in practice. Two other methods were introduced in [19]: direct truncation of the infinite sum expansion of the exponential (referred to as *Unif*) and matrix exponentiation on an auxiliary matrix (referred to as *Expm*). The *Unif* method was marked as unstable later in the article, whereas *Expm* as being most robust and time efficient and able to compute models with hundreds of hidden states.

There was an effective theoretical solution, but there was no publicly available implementation that would spread the technology towards general public. Our intention was to create first of its kind library implementing the state-of-the-art methods for CT-HMM, that would be moreover easy and fast to understand, and licensed as open source so it can be used and improved by everyone.

As the result of this thesis, we have introduced the theory of the hidden Markov Models (Chapter 1), described and explained the HMMs algorithms (Chapter 2), implemented the general efficient HMM library for Python (details in Chapter 3). The library effectively implements both discrete-time and continuous-time HMM. It provides all useful methods for work with the models as well as all algorithms described in the second chapter. Its key feature is the EM algorithm for continuous-time HMM parameters estimation. We have implemented more variants of *Expm* algorithm. User can choose between *soft* and *hard* method 2.3.2, *float-interval* and *integer-interval* variant of algorithm 2.4 and train with both full and sparse jump-rate matrix. The behavior of all implemented variants is examined by conducting experiments (Chapter 4).

Chapter 1

Probabilistic Models

Before we start to talk about Continuous-time Hidden Markov Model (CT-HMM) we will briefly explain its discrete-time variant, usually just referred to as Hidden Markov Model (HMM or DT-HMM), starting by explaining the underlaying Markov process. This forms logical hierarchy as the CT-HMM is the natural extension of the discrete model sharing many ideas and subroutines, and Markov process is base building block of overall model.

1.1 Discrete-time Markov Process

Discrete-time Markov process, also referred to as Markov chain, is the stochastic process X_t in discrete equidistant time $\mathbf{T} = \{1, 2, 3 \dots\}$, that in every time-step occupies some state of the potentially infinite state set [8]. However, later in the text we restrict ourself on the finite state set $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$. The process being in time $t_i \in \mathbf{T}$ in the state s_j will be denoted as $X_{t_i} = s_j$. Alternatively, to make some explanation simpler, we can also denote it just as $X_{t_i} = j$. The state of the process can change in every time-step. Probability that the process changed from state s_i to state s_j is called transition probability. We will refer to it as a_{ij}. The transition probabilities for all pairs of states form square matrix of size n - the transition probabilities matrix $\mathbf{A}$. Note that elements on its diagonal, although referred to as transition probabilities, represent the probabilities of the process remaining in the same state.

The Markov process is memory-less, not holding information about the past states. So the probability of next transition depends only on the current state. This characteristic is called the Markov property and can be expressed by the following equation.

$$\begin{aligned} &\mathrm{P}(X_t = s_j \mid X_{t-1} = s_i, X_{t-2} = s_k, \dots) \\ &\quad = \mathrm{P}(X_t = s_j \mid X_{t-1} = s_i), \quad s_i, s_j, s_k \in \mathcal{S}, t \in \mathbf{T} \end{aligned} \tag{1.1}$$

If not stated differently, we will always refer to the time-homogeneous

Markov processes. Homogeneity of the Markov process, described in following equation, guarantees that one step transition probabilities are not changing throughout the time so the matrix **A** stays constant.

$$a_{ij} = \mathrm{P}(X_t = s_j \mid X_{t-1} = s_{j-1}), \qquad 1 \leq i, j \leq n, \forall t \tag{1.2}$$

Standard stochastic constrains are applied for the transition probabilities [23].

$$a_{ij} \geq 0 \tag{1.3}$$

$$\sum_{j=1}^{n} a_{ij} = 1 \tag{1.4}$$

1.2 Discrete-Time Hidden Markov Model

The Discrete-time Hidden Markov Model (HMM) [23] is doubly embedded stochastic process consisting of underlaying discrete-time Markov process X_t and another process O. The process O assigns to every state s_i the specific observation symbol from the set $\mathcal{V}$ that will be emitted by certain probability. The model is called "hidden", because the state sequence of Markov process is not directly visible. It can only be guessed from the measured observation symbols using statistical techniques. The set $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$ is a set of all observable (emission) symbols. The probability that state s_i emits symbol v_j or $\mathrm{P}(O_t = v_j \mid X_t = s_i)$ will be denoted as $b_i(j)$ and together with other elements form the matrix B, rectangular matrix of n rows and m columns. Comparing to the single Markov process, HMM is able to model more complex systems, that can be closer to describe the processes of real life problems.

The following parameters are needed to describe the HMM:

1. **Hidden States**
$$\mathcal{S} = \{s_1, s_2, \ldots, s_n\} \tag{1.5}$$

2. **State Transition Probability Matrix**
$$\mathbf{A} = \{a_{ij}\}, \quad 1 \leq i, j \leq n \tag{1.6}$$

3. **Observation Symbols**
$$\mathcal{V} = \{v_1, v_2, \ldots, v_m\} \tag{1.7}$$

4. **Observation Symbols Probability Matrix**
$\mathbf{B} = \{b_i(k)\}$, where $b_i(k)$ is the probability that the observation v_k will occur, if the system is currently in state i.
$$b_i(k) = \mathrm{P}(O_t = v_k \mid X_t = s_i), \qquad 1 \leq i \leq n, \quad 1 \leq k \leq m \tag{1.8}$$

5. **Initial state distribution**
   $\boldsymbol{\pi} = \{\boldsymbol{\pi}_i\}$, where $\boldsymbol{\pi}_i$ is the probability of the initial state being s_i.

$$\boldsymbol{\pi}_i = \mathrm{P}(X_1 = s_i), \qquad 1 \leq i \leq n \tag{1.9}$$

For the convenience we will declare parameter $\boldsymbol{\theta} = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ compactly denoting the set of all parameters of the model.

1.3 Continuous-Time Markov Process

In the discrete-time Markov process described in Section 1.1 change of the current state of the process could only occur once we have moved a step further in the discrete time t. Comparing to this, in the continuous-time Markov process (CTMP) can change of state occur at any moment (the occurrence holds exponential distribution).

A finite state continuous-time Markov process is a stochastic process $\{X_t \mid t > 0\}$ on the states $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ (for $n > 0$ and $\mathcal{S}$ being the finite state set), that in any time t occurs in corresponding state $x_t \in \mathcal{S}$.

It satisfies the following properties for times $0 \leq u_0 < u_1 < \cdots < u_r < u$:

Markov property: Probability of transition from state i to state j during time interval t is stationary i.e. independent on the states of process in the times $u^{'}$, $u^{'} < u$.

$$\begin{aligned} &\mathrm{P}(X_{t+u} = s_j \mid X_u = s_i, X_{u_r} = s_{i_r}, \ldots, X_{u_0} = s_{i_0}) = \\ &= \mathrm{P}(X_{t+u} = s_j \mid X_u = s_i), \qquad s_i, s_j, s_{i_0}, \ldots, s_{i_r} \in \mathcal{S} \end{aligned} \tag{1.10}$$

The property describes that the stochastic process is memoryless.

Homogeneity: Probability of transition from state s_i to s_j in any given time $u \geq 0$ depends only on the length of the time interval $t \geq 0$.

$$\begin{aligned} &\mathrm{P}(X_{t+u} = s_j \mid X_u = s_i) = \mathrm{P}(X_t = s_j \mid X_0 = s_i) = \\ &= \mathrm{p_{ij}}(t) \end{aligned} \tag{1.11}$$

The upper mentioned conditions assert that the transition probability $\mathrm{p_{ij}}(t), i, j = 1, \ldots, n$ satisfies following conditions [15]:

$$\mathrm{p_{ij}}(t) \geq 0, \tag{1.12}$$

$$\sum_{j=1}^{n} \mathrm{p_{ij}}(t) = 1, \tag{1.13}$$

$$\sum_{k=1}^{n} \mathrm{p_{ik}}(u)\mathrm{p_{kj}}(t) = \mathrm{p_{ij}}(u+t). \tag{1.14}$$

Equation (1.14) is known as *Chapman-Kolmogorov equation*. We can define the matrix $\mathbf{P_t}$, where the entry (i, j) is $\mathrm{p_{ij}}(t)$ to get the equation in matrix form:

$$\mathbf{P_u P_t} = \mathbf{P_{u+t}}, \qquad t, u > 0. \tag{1.15}$$

1.3.1 Jump Rates

Using the *Chapman-Kolmogorov equation* (1.14), and if we know the probability $\mathrm{p_{ij}}(t)$ for every states i, j and time $0 < t < t_0$, we are able to compute the values for any time $t > 0$ [15].

State transition probability $\mathrm{p_{ij}}(t)$ is continuous for $t = 0$ [8]. After assigning $t = 0$ we will get identity matrix $\mathbf{P_0} = \mathbf{I}$. Moreover from (1.15) follows that $\mathrm{p_{ij}}(t)$ is continuous for all $t > 0$ and so that there exists the right derivative in 0. This knowledge enables us to determine the $\mathrm{p_{ij}}(t)$ for any given time $t > 0$.

$$q_{ij} = \left.\frac{\mathrm{dp_{ij}}(t)}{\mathrm{dt}}\right|_{t=0} \tag{1.16}$$

We will call this derivative q_{ij} the jump rate from state s_i to some other state s_j. The jump rate q_{ii} can be derived from the equation of transition probabilities summing to one.

$$1 = \mathrm{p_{ii}}(t) + \sum_{j=1, j\neq i}^{n} \mathrm{p_{ij}}(t) \tag{1.17}$$

Dividing it by time interval t and letting t decrease close to zero, we obtain the following equation [5].

$$q_{ii} = -\sum_{j=1, j\neq i}^{n} q_{ij} \tag{1.18}$$

We assume only finite rates q_{ij}. Infinite rate would immediately leave the state, so it makes no sense for us to consider.

It can be shown that by construction of the CTMP from discrete time Markov process (DTMP) 1.3.2 and it's underlying Poisson process with rate λ the equation $q_{ij} = \lambda a_{ij}$ holds. Where a_{ij} is the transition probability from state s_i to state s_j in DTMP 1.6. This is why we call q_{ij} the jump rate [8].

We can also look at q_{ii} as at the rate in which the X_t is leaving state s_i (1.14). We define the jump-rates matrix as $\mathbf{Q}(i, j) = q_{ij}$.

To get transition probabilities for any time t from matrix $\mathbf{Q}$ we need to solve the *Kolmogorov's Backward* equation

$$\mathbf{P_t^{'}} = \mathbf{QP_t}, \tag{1.19}$$

Kolmogorov's Forward equation. They can be obtained from *Chapman-Kolmogorov equation* (1.15) by differentiation [8]. The solution of the differential equation with the initial condition $\mathbf{P_0} = \mathbf{I}$ leads to following matrix exponentiation formula.

$$\mathbf{P_t} = e^{\mathbf{Q}t} \tag{1.20}$$

1.3.2 Construction from Discrete-Time Markov Process with Poisson Process Timing

The useful way for understanding the continuous-time Markov process is to know its relation to the discrete process that can be shown by the following construction [8].

We will take a homogeneous Poisson process $N(t)$ with parameter λ and a discrete-time Markov process Y_ν with states transition probabilities a_{ij}. We assume the $N(t)$ and Y_ν being mutually independent. By changing equidistant timing of Y_ν to Poisson process timing, we will get the process $Y_{N(t)}$ in which transition from state s_i happens at each arrival in $N(t)$ with probability $a_i = \sum_{j=1}^{n,j\neq i} a_{ij}$. Notice that the transition will not occur if $i = j$. Similarly, the time spent in state s_i before the transition occurs can be described by probability density function of exponential distribution with the parameter λa_i as $\mathrm{f}(t) = \lambda a_i e^{-\lambda a_i t}$. Such a process $X_t = Y_{N(t)}$ fulfills the CTMP definition as described in Section 1.3.

Now we can derive the equation for probability of transition from state s_i to state s_j in the time t as the sum of all possible numbers of steps in which the transition can occur.

$$\begin{aligned}
\mathrm{p_t(i,j)} &= \sum_{n=0}^{\infty} \mathrm{P}(N(t) = n, X_t = s_j \mid X_0 = s_i) = \\
&= \sum_{n=0}^{\infty} \mathrm{P}(N(t) = n, Y_{N(t)} = s_j \mid Y_{N(0)} = s_i) = \\
&= \sum_{n=0}^{\infty} \mathrm{P}(N(t) = n)\mathrm{P}(Y_n = j \mid Y_0 = i) = \\
&= \sum_{n=0}^{\infty} e^{-\lambda t}\frac{(\lambda t)^n}{n!}a_{ij}^n
\end{aligned} \tag{1.21}$$

By differentiation of $\mathrm{p_t(i,j)}$ with respect to t at 0, we get the following equation for jump-rate matrix $\mathbf{Q}$.

$$\mathbf{Q} = \lambda(\mathbf{A} - \mathbf{I}) \tag{1.22}$$

1.3.3 Fully Observable Continuous-Time Markov Process

Let's have continuous time Markov process on the state space $\mathcal{S}$ with the jump rates in the matrix $\mathbf{Q}$, initial state probability distribution $\boldsymbol{\pi}$, and the known hidden state sequence $\mathbf{X}' = (x'_0, x'_1, \ldots, x'_{\nu'})$ occurring with transitions in times $\mathbf{T}' = (t'_0, t'_1, \ldots, t'_{\nu'})$. Such a system in which we know when and where the transition will happen is called fully observable [19]. We can count its complete likelihood $\mathcal{L}_{FO}$ what is the probability of state sequence $\mathbf{X}'$ and time sequence $\mathbf{T}'$ being generated by model with parameters $\mathbf{Q}$.

$$\begin{aligned}\mathcal{L}_{FO} &= \mathrm{P}(X'_{t'_0} = x'_0, \ldots, X'_{t'_\nu} = x'_\nu \mid \mathbf{Q}) = \\ &= \prod_{u=0}^{\nu'-1} (q_{x'_u x'_{u+1}} / q_{x'_u})(q_{x'_u} e^{-q_{x'_u}\tau'_u}) = \\ &= \prod_{u=0}^{\nu'-1} q_{x'_u x'_{u+1}} e^{-q_{x'_u}\tau'_u}\end{aligned} \tag{1.23}$$

where $q_i = \sum_{j=1}^{n, i\neq j} q_{ij}$ is the probability of transition from state i and $\tau'_u = t'_{u+1} - t'_u$ is the time interval among two consecutive steps.

The equation can be further rearranged into form that group together same state transition. Variable η_{ij} marks the number of transition q_{ij} that have occurred and τ_i is the total time spend in state s_i.

$$\mathcal{L}_{FO} = \prod_{i=1}^{n} \prod_{j=1, i\neq j}^{n} q_{ij}^{\eta_{ij}} e^{-q_i \tau_i} \tag{1.24}$$

1.3.4 General Continuous-Time Markov Process

In general we do not know states of the system during all the time. It is only known at some unevenly distributed times of observations $\mathbf{T} = (t_0, t_1, \ldots, t_\nu)$ as $\mathbf{X} = (x_0, x_1, \ldots, x_\nu)$. This add an amount of insecurity in the probability computation. We do not longer know, the number of transitions η_{ij} as well as the time spend in the specific state τ_i.

To count the likelihood of the process [19], we use earlier defined matrix $\mathbf{P}(t)$ and its elements $\mathrm{p_{ij}}(t)$ 1.3. The time interval among two observation is

marked as $\tau_u = t_{u+1} - t_u$.

$$\mathcal{L} = \mathrm{P}(X_{t_0} = x_0, \ldots, X_{t_\nu} = x_\nu \mid \mathbf{Q}) = \prod_{u=0}^{\nu-1} \mathrm{p}_{x_u x_{u+1}}(\tau_u) \tag{1.25}$$

It can be alternatively extended in the form

$$\mathcal{L} = \prod_{u=0}^{\nu-1} \prod_{i=0,j=0}^{n} \mathrm{p}_{ij}(\tau_u)^{\mathbb{1}(x_u=s_i, x_{u+1}=s_j)} \tag{1.26}$$

where function $\mathbb{1}(i,j)$ equals either 1, if both condition inside are true, or 0 if they are not. We define r as the number of all distinct time intervals from the set $\mathbf{T}_\Delta = \{\tau_1, \tau_2, \ldots, \tau_r\}$. In case of r being lower then the number of observations it can be beneficial to aggregate them as in formula

$$\mathcal{L} = \prod_{\Delta=1}^{r} \prod_{i=0,j=0}^{n} \mathrm{p}_{ij}(\tau_\Delta)^{\mathbb{C}(\tau=\tau_\Delta, x_u=s_i, x_{u+1}=s_j)} \tag{1.27}$$

where function $\mathbb{C}$ denotes to total number of intervals for which is the condition true.

1.4 Continuous-Time Hidden Markov Model

Continuous-time hidden Markov model is an extension of CTMP 1.3 where states in times $\mathbf{T} = (t_0, t_1, \ldots, t_\nu)$ are not directly observed but just seen as the observation symbols sequence $\mathbf{O} = (o_0, o_1, \ldots, o_\nu)$ emitted by the current state s_i with the probability $b_i(o)$.

Emissions of the observable symbol can occur at any time independently on the state transition times. For example it could be times when patient undergoes medical examination. The times can generally have highly irregular and unbalanced distribution.

The likelihood of completely observed system $\mathcal{L}_{FO}$ is similar to the $\mathcal{L}_{FO}$ of CTMP (1.23) (1.24) with the difference we need to take into account the probability of actual observation.

$$\begin{aligned}
\mathcal{L}_{FO} &= \mathrm{P}(X_{t_0} = x_0, \ldots, X_{t_\nu} = x_\nu, O_{t_0} = o_0, \ldots, O_{t_\nu} = o_\nu \mid \mathbf{Q}, \mathbf{B}) = \\
&= \prod_{u'}^{\nu'-1} q_{x_{u'} x_{u'+1}} e^{-q_{x_{u'}} \tau_{u'}} \prod_{u=0}^{v} b_{x_u}(o_u) = \\
&= \prod_{i=1}^{n} \prod_{j=1, i\neq j}^{n} q_{ij}^{\eta_{ij}} e^{-q_i \tau_i} \prod_{u=0}^{\nu} b_{x_u}(o_u)
\end{aligned} \tag{1.28}$$

There is much more latent information in such defined model. Not just the hidden states, but also the unknown transition times and unknown state

sojourn time (how long will the system remain in the state). Sometimes state can change, without emitting a single observation. The large number of hidden information make it to be more complex problem then the discrete time model.

Chapter 2

Statistics

2.1 Expectation-Maximization Algorithm

Expectation-maximization (EM) algorithm (first introduced in [6]) is the method for finding the maximum likelihood estimates(MLE) of parameters in probabilistic models over incomplete data-set, i.e. data-set containing unobserved (latent) variables. It is a natural generalization of maximum likelihood estimation. However, the latent variables make finding of the MLE more difficult. Direct count can be very computationally expensive. The EM algorithm uses iterative approach to approximate the solution by repeating sequence of simpler consecutive steps.

Let's have $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ containing the known (observed) variables and the vector $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ containing latent (unobserved) variables. We set $\boldsymbol{\theta}$ for unknown parameters we want to estimate. We denote notation $\mathcal{L}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) = \mathrm{P}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$ for likelihood function which estimates the probabilities for given parameters. We can count MLE directly by summing through all possible values of latent variables $\mathbf{z}$ as in following equation $\mathcal{L}(\mathbf{x}; \boldsymbol{\theta}) = \mathrm{P}(\mathbf{x}|\boldsymbol{\theta}) = \sum_{\mathbf{z}} \mathrm{P}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$ or get its approximation by EM algorithm in following steps [17].

1. **Initialization:** Make initial guess of parameters $\hat{\boldsymbol{\theta}}_\mathbf{0}$.

Continue by iterating through two following steps:

2. **Expectation:** Calculate expected value of the log-likelihood function $\log \mathcal{L}(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})$ conditioned by $\mathbf{z}$ and given $\mathbf{x}$ where $\hat{\boldsymbol{\theta}}_\boldsymbol{t}$ is the current parameters estimation.

$$\mathrm{M}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t) = \mathbf{E}_{\mathbf{z}|\mathbf{x},\hat{\boldsymbol{\theta}}_t}\left[\log \mathcal{L}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{z})\right] \tag{2.1}$$

3. **Maximization:** Find the values of parameters $\hat{\boldsymbol{\theta}}_{t+1}$ that maximize the function defined in expectation step.

$$\hat{\boldsymbol{\theta}}_{t+1} = \arg\max_{\boldsymbol{\theta}} \mathrm{M}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t) \tag{2.2}$$

As the value of $\mathrm{M}(\hat{\boldsymbol{\theta}}_t)$ matches the log-likelihood function at $\hat{\boldsymbol{\theta}}_t$, it follows that $\mathrm{Q}(\hat{\boldsymbol{\theta}}_t) = \log \mathrm{P}(\mathbf{x}, \mathbf{z}; \hat{\boldsymbol{\theta}}_t) \leq \mathrm{M}(\hat{\boldsymbol{\theta}}_{t+1}) = \log \mathrm{P}(\mathbf{x}, \mathbf{z}; \hat{\boldsymbol{\theta}}_{t+1})$ thus the function values in consecutive iterations form non-decreasing sequence. The computation can be stopped once the parameters converge to some values, or if their change is not more significant.

It is important to notice that this approach may lead to local optima. To get better results the algorithm can be launched more times starting always from different randomly initiated parameters. Possibly enhanced by the use of some more advance non-linear optimization technique.

Look at [7] for more detailed, easily understandable tutorial with examples.

2.2 Discrete-Time Hidden Markov Model

When using HMM in real-world application we need to deal with following problems. First we describe them and later in the successive subsections we explain the algorithms that can effectively solve them (explained in detail in Article [23] Sec. 3 A,B & C).

1. Compute the probability $\mathrm{P}(\mathbf{O}|\boldsymbol{\theta})$ of the observation sequence $\mathbf{O} = (o_1, o_2, \ldots, o_\nu)$, given the parameters $\boldsymbol{\theta} = (A, B, \boldsymbol{\pi})$ defined in Section 1.2. Elements of the observation sequence $\mathbf{O}$ are some specific measured data, taking values from the set $\mathcal{V}$.

2. Choose the optimal state sequence $\mathbf{X} = (x_1, x_2, \ldots, x_\nu)$ having the observation sequence $\mathbf{O}$ and parameters $\boldsymbol{\theta}$.

3. Adjust the model parameters $\boldsymbol{\theta}$ in the way it maximizes the probability of observation sequence $\mathrm{P}(\mathbf{O}|\boldsymbol{\theta})$.

2.2.1 Forward-Backward Algorithm

The *forward-backward* algorithm is actually pair of two separate algorithms (forward vs. backward). We will explain the *forward* one and at the end we will describe the modifications that are needed to do the *backward*. Both of them are sufficient to solve the first proposed problem 1 separately. We still need to define both, because of the use in the later text.

Forward algorithm is the dynamic programming algorithm benefiting from the Markov property (1.1) - the independence upon past events. We define forward variable $\alpha_t(i)$ as the probability of the partial observation sequence with

the last observation in time t emitted by state s_i given the model parameters $\boldsymbol{\theta}$.

$$\alpha_t(i) = \mathrm{P}(o_1, o_2, \ldots, o_t, X_t = s_i \mid \boldsymbol{\theta}) \tag{2.3}$$

The forward variables can be gradually counted using the bottom-up strategy and following equations for $t = 1$ and $t = t + 1$.

$$\alpha_1(i) = \boldsymbol{\pi}_i b_i(o_1), \qquad 1 \leq i \leq n \tag{2.4}$$

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{n} \alpha_t(j) a_{ji} \right) b_i(o_{t+1}), \qquad \begin{array}{l} 1 \leq t \leq \nu - 1, \\ \\ 1 \leq i \leq n \end{array} \tag{2.5}$$

Now we can obtain the solution of the first problem simply by summing through the all forward variables in the time ν.

$$\mathrm{P}(\mathbf{O}|\boldsymbol{\theta}) = \sum_{i=1}^{n} \alpha_\nu(i) \tag{2.6}$$

Similarly we can define backward variable $\beta_t(i)$ as

$$\beta_t(i) = \mathrm{P}(o_{t+1}, o_{t+2}, \ldots, o_\nu, X_t = s_i \mid \boldsymbol{\theta}). \tag{2.7}$$

The Dynamic programming (DP) algorithm consist from two following equations for times $t = \nu$ and $t < \nu$.

$$\beta_\nu(i) = 1, \qquad 1 \leq i \leq n \tag{2.8}$$

$$\beta_t(i) = \sum_{j=1}^{n} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \qquad \begin{array}{l} 1 \leq t \leq \nu - 1, \\ \\ 1 \leq i \leq n \end{array} \tag{2.9}$$

The solution for the first problem can be obtain by summing probabilities for all the beginning states. We need to be careful not to forget to multiply by probability of the first observation symbol being emitted as it is not part of the definition for backward variable in time $t = 1$.

$$\mathrm{P}(\mathbf{O}|\boldsymbol{\theta}) = \sum_{i=1}^{n} \beta_1(i) b_i(o_1) \tag{2.10}$$

We need to count n variables at each time-step, each of them takes exactly n steps to evaluate. It makes overall complexity $\mathcal{O}(n^2\nu)$.

2.2.2 Individually Most Likely States Sequence

There are multiple ways how we can look at the word "optimal" in the problem two statement 2. One of the possible approaches is to maximize the expected number of correctly assigned states. To solve it, we define the variable determining the probability of being in the specific state in the particular time. We will also referred to it as "single state probability".

$$\gamma_t(i) = \mathrm{P}(X_t = s_i \mid \mathbf{O}, \boldsymbol{\theta}) \tag{2.11}$$

Here we can use already defined forward and backward variables and count $\gamma_t(i)$ as

$$\begin{aligned} \gamma_t(i) &= \frac{\mathrm{P}(X_t = s_i, \mathbf{O} \mid \boldsymbol{\theta})}{\mathrm{P}(\mathbf{O} \mid \boldsymbol{\theta})} = \frac{\alpha_t(i)\beta_t(i)}{\mathrm{P}(\mathbf{O} \mid \boldsymbol{\theta})} = \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{n} \alpha_t(j)\beta_t(j)} \end{aligned} \tag{2.12}$$

To get the desired individual most likely state x_t, it is enough to find one of the highest probability.

$$x_t = \underset{1 \le i \le n}{\operatorname{argmax}}\, \gamma_t(i), \qquad 1 \le t \le \nu \tag{2.13}$$

Applying this algorithm to the whole sequence leads to the highest expected number of correctly assigned states. However, the sequence as a whole can in some cases have very low probability or even not be feasible. This would happen if probability of transition among two consecutive states in the sequence was zero.

2.2.3 Viterbi Algorithm

Another way how to look on the problem two 2 is to find single most probable state sequence. It means to maximize $\mathrm{P}(\mathbf{X}|\mathbf{O}, \boldsymbol{\theta})$ what is equivalent to maximizing of $\mathrm{P}(\mathbf{X}, \mathbf{O} \mid \boldsymbol{\theta})$.

Viterbi algorithm is dynamic programming algorithm that similarly to the forward-backward algorithm benefits from memorylessness of Markov Chain. In DP we can gradually count the variable $\delta_t(i)$ that represents the maximal probability of the state chain from its beginning till the state in time t with the state x_t being s_i.

$$\delta_t(i) = \max_{x_1, x_2, \ldots, x_{t-1}} \mathrm{P}(x_1, x_2, \ldots, x_t = s_i, o_1, o_2, \ldots, o_t \mid \boldsymbol{\theta}) \tag{2.14}$$

To get the actual state sequence we need to store information about the state in time $t-1$ which has maximized the previous equation. We store it in the array $\psi_t(i)$. Now we can define the initialization of the algorithm as

$$\delta_1(i) = \boldsymbol{\pi}_i b_i(o_1), \qquad 1 \leq i \leq n \tag{2.15}$$

$$\psi_1(i) = 0, \tag{2.16}$$

and consecutive bottom-up computation as

$$\delta_t(i) = (\max_{1 \leq j \leq n} \delta_{t-1}(j) a_{ji}) b_i(o_t), \qquad \begin{array}{l} 2 \leq t \leq \nu, \\ 1 \leq i \leq n \end{array} \tag{2.17}$$

$$\psi_t(i) = (\operatorname*{argmax}_{1 \leq j \leq n} \delta_{t-1}(j) a_{ji}), \qquad \begin{array}{l} 2 \leq t \leq \nu, \\ 1 \leq i \leq n. \end{array} \tag{2.18}$$

Now we can get the searched state sequence probability

$$\mathrm{P}^* = \max_{1 \leq i \leq n} (\delta_t(i)), \tag{2.19}$$

and the actual state path $\mathbf{X}^* = (x_1^*, x_2^*, \ldots, x_\nu^*)$ by backtracking

$$x_t^* = \operatorname*{argmax}_{1 \leq i \leq n} (\delta_t(i)), \qquad t = \nu \tag{2.20}$$

$$x_t^* = \psi_{t+1}(x_{t+1}^*), \qquad t = \nu - 1, \nu - 2, \ldots, 1. \tag{2.21}$$

The structure of the algorithm is very similar to the Forward-Backward algorithm so we can easily see its complexity is also $\mathcal{O}(n^2\nu)$.

2.2.4 Expectation-Maximization Algorithm

There is any known general analytic solution for the problem three 3. There exist more possible iterative algorithms from which we will describe the expectation-maximization approach 2.1 based on the classic work of Baum and his colleagues called *Baum-Welch* algorithm [3]. We will start by defining the variable $\xi_t(i, j)$ as the probability of being in state s_i in time t and in state s_j in time $t+1$. Later we will also refer to it as "double state probability".

$$\xi_t(i, j) = \mathrm{P}(X_t = s_i, X_{t+1} = s_j \mid \mathbf{O}, \boldsymbol{\theta}) \tag{2.22}$$

The probability can be computed using forward-backward variables as follows

$$\begin{aligned}\xi_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\mathrm{P}(\mathbf{O} \mid \boldsymbol{\theta})} \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}\end{aligned} \tag{2.23}$$

Obviously, it is in relation with already defined variable $\gamma_t(i)$ - the probability of being in state s_i in time t.

$$\gamma_t(i) = \sum_{j=1}^{n}\xi_t(i,j) \tag{2.24}$$

We can get expected number of transitions from s_i when summing $\gamma_t(i)$ over the time till $\nu - 1$. Similarly we would get expected number of visits of state s_i by summing $\gamma_t(i)$ from $t = 1$ till $t = \nu$.

$$\mathbf{E}(\text{transitions from } s_i) = \sum_{t=1}^{\nu-1}\gamma_t(i) \tag{2.25}$$

If summing $\xi_t(i,j)$ over the time we get expected number of transitions from s_i to s_j.

$$\mathbf{E}(\text{transitions from } s_i \text{ to } s_j) = \sum_{t=1}^{\nu-1}\xi_t(i,j) \tag{2.26}$$

Now we have all what is needed to define the reestimated model $\hat{\boldsymbol{\theta}} = (\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\boldsymbol{\pi}})$.

$$\hat{\boldsymbol{\pi}}_{\boldsymbol{i}} = \gamma_1(i) \tag{2.27}$$

$$\begin{aligned}\hat{a}_{ij} &= \frac{\mathbf{E}(\text{transitions from } s_i \text{ to } s_j)}{\mathbf{E}(\text{transitions from } s_i)} \\ &= \frac{\sum_{t=1}^{\nu-1}\xi_t(i,j)}{\sum_{t=1}^{\nu-1}\gamma_t(i)}\end{aligned} \tag{2.28}$$

$$\begin{aligned}\hat{b}_i(k) &= \frac{\mathbf{E}(\text{times of visiting } s_i \text{ and observing symbol } v_k\ )}{\mathbf{E}(\text{times of visiting } s_i)} \\ &= \frac{\sum_{t=1,\text{if } O_t=v_k}^{\nu}\gamma_t(i)}{\sum_{t=1}^{\nu}\gamma_t(i)}\end{aligned} \tag{2.29}$$

The formulas can be interpreted as the steps of EM-algorithm with the expectation step being the computation of auxiliary function $Q(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}})$ and maximization step the maximization over parameter $\hat{\boldsymbol{\theta}}$. It is important to notice, that the found solution is local optimum. The searched space can be very complicated containing many local optima. That's way it can be helpful to start the algorithm more times with different parameter initialization.

2.3 Continuous-Time Hidden Markov Model

We have introduced the basic problems of DT-HMM at the beginning of Section 2.2. They can be similarly defined for continuous-time model. Just not only the observation sequence $\mathbf{O}$ but also the time sequence $\mathbf{T}$ need to be included. The solution for two first problems as well as a considerate part of the third problem is pretty straight-forward. The discrete-time solutions just needs to be redefined with different transition probability matrix for different intervals. It is discussed more deeply in the following Subsection 2.3.1. The EM algorithm for jump-rate matrix estimation is not so straight-forward anymore. We need to care about what is happening in times between two observations as it has impact on the results of likelihood computation. The EM algorithm and its variants are explained in all the remaining subsections of this chapter.

2.3.1 The Posterior State Probabilities

To count the single and double states probability of the CT-HMM with specified parameters, we do not need to know about the state transition that have occurred in between of two observations. If we take just the observation points into the account, we can look on the model as it was the time inhomogeneous DT-HMM. It is the model in which the state transition probabilities can change in every time step. We have previously defined the transition probability from state s_i to state s_j in time interval τ as $\mathrm{p_{i,j}}(\tau) = e^{\mathbf{Q}\tau}[i,j]$. The $\mathbf{P}_t$ can play the role of the transition probabilities matrix $\mathbf{A}$ in the discrete time inhomogeneous model. For example the probability of the model with the known states in the observation times can be computed as following

$$\begin{aligned} &\mathrm{P}(X_{t_0} = x_0, \ldots, X_{t_\nu} = x_\nu, O_{t_0} = o_0, \ldots, O_{t_\nu} = o_\nu \mid \mathbf{Q}, \mathbf{B}) = \\ &= \prod_{u=1}^{\nu-1} \mathrm{p_{x_u, x_{u+1}}}(\tau_u) \prod_{u=1}^{\nu} b_{x_u}(o_u), \end{aligned} \tag{2.30}$$

where $\tau_u = t_{u+1} - t_u$ is the u-th time interval. It should be obvious now that for the continuous variants of the Viterbi and Forward-backward algorithm we only need to change the state transitions a_{ij} to the time variable $\mathrm{p_{i,j}}(\tau_u)$. Now, we can use the forward and backward variables to count posterior state

distribution for the single state $\gamma_t(i) = \mathrm{P}(X_{t_u} = s_i \mid \mathbf{O}, \mathbf{T}, \mathbf{Q_t})$ and the two consecutive states $\xi_t(i,j) = \mathrm{P}(X_{t_u} = s_i, X_{t_{u+1}} = s_j \mid \mathbf{O}, \mathbf{T}, \mathbf{Q_t})$ in the same way as we have done in the discrete model.

2.3.2 Soft vs. Hard Method

The described way for computing of posterior state distribution of hidden states is referred to as *Soft method*. Supplementary, there is the *Hard method*. The *Hard method* EM algorithm tries to maximize the probability of the most probable hidden states sequence. The sequence can be obtained by the Viterbi algorithm and than be used "hardly", just considering the single pair of end states for every time interval, to update the single and double states probability tables. Important to notice is that the *Hard method* tries to maximize the probability of most likely sequence instead of maximizing the most likely observed data [2]. So the optimization of the second mentioned is created just as the byproduct. Also there is no guarantee for the *Hard method* that the probability of the dataset being generated by the model will be non-decreasing throughout the iterations. Instead we can securely claim the non-decreasing character of maximal state sequence probability.

2.3.3 Continuous-Time Markov Process EM

There is no available analytic maximizer of likelihood function (1.27), however it can be estimated iteratively by the EM algorithm proposed in article [20]. For the expectation step we will use logarithm of expected complete likelihood function (1.24) with the estimated parameter of $\mathbf{Q}$ in step t noted as $\hat{\mathbf{Q}}_\mathbf{t}$.

$$\ln(\mathcal{L}_{FO}) = \sum_{i=0, i\neq j}^{n} \sum_{j=0}^{n} \ln(\hat{q}_{ij}) \mathbf{E}(\eta_{ij} \mid \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t}) - \hat{q}_i \mathbf{E}(\tau_i \mid \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t}) \tag{2.31}$$

In the maximization step we take the derivative with respect to q_{ij} and evaluate new values for $\hat{\mathbf{Q}}$ so that they maximize the previous equation.

$$\hat{\mathbf{Q}}_\mathbf{t}(i,j) = \begin{cases} \frac{\mathbf{E}(\eta_{ij} | \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t})}{\mathbf{E}(\tau_i | \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t})} & \text{if } i \neq j \\ -\sum\limits_{\substack{k=1 \\ n,k\neq i}} q_{ik}, & \text{if } i = j \end{cases} \tag{2.32}$$

The remaining non-trivial task is to evaluate expectations $\mathbf{E}(\eta_{ij} \mid \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t})$

and $\mathbf{E}(\tau_i \mid \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t})$. They can be expressed by following sums.

$$\begin{aligned}\mathbf{E}(\eta_{ij} \mid \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t}) &= \sum_{u=0}^{\nu-1} \mathbf{E}(\eta_{ij} \mid x_u, x_{u+1}, \hat{\mathbf{Q}}_\mathbf{t}) = \\ &= \sum_{u=0}^{\nu-1} \sum_{k=0,l=0}^{n} \mathbb{1}(x_u = k, x_{u+1} = l)\mathbf{E}(\eta_{ij} \mid x_u = k, x_{u+1} = l, \hat{\mathbf{Q}}_\mathbf{t})\end{aligned} \tag{2.33}$$

$$\begin{aligned}\mathbf{E}(\tau_i \mid \mathbf{X}, \hat{\mathbf{Q}}_\mathbf{t}) &= \sum_{u=0}^{\nu-1} \mathbf{E}(\tau_i \mid x_u, x_{u+1}, \hat{\mathbf{Q}}_\mathbf{t}) = \\ &= \sum_{u=0}^{\nu-1} \sum_{k=0,l=0}^{n} \mathbb{1}(x_u = k, x_{u+1} = l)\mathbf{E}(\tau_i \mid x_u = k, x_{u+1} = l, \hat{\mathbf{Q}}_\mathbf{t})\end{aligned} \tag{2.34}$$

The Markov property and homogeneity of the Markov process allows to reduce the computation of expected values to all $i, j, k, l \in S$ for all distinct time-intervals. The ways how to do it are shown in Section 2.3.5.

2.3.4 Continuous-Time Hidden Markov Model EM

For the EM parameters estimation in DT-HMM, we have used the well known Baum-Welch algorithm 2.2.4. We can use the similar approach for estimation of parameters $\hat{\boldsymbol{\pi}}$ (2.27) and $\hat{\mathbf{B}}$ (2.29). We just need to use different transition probabilities matrix for every different interval as described in Subsection 2.3.1.

The estimation of the $\hat{\mathbf{Q}}$ can't be inherited from the discrete model, because the state transitions do not depend on the observation times and there can even be more of them between two consecutive observations. The following method proposed in the article [19], was the first which could efficiently solve the issue.

In the expectation step we will calculate the expected value for logarithm of fully observable likelihood function (1.28) given the model parameters and observation sequences.

$$\begin{aligned}\ln(\mathcal{L}_{FO}) = &\sum_{i=1}^{n} \sum_{j=1,i\neq j}^{n} \left(\ln(\hat{q}_{ij})\mathbf{E}(\eta_{ij} \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t}) - \hat{q}_i \mathbf{E}(\tau_i \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t})\right) + \\ &+ \sum_{u=0}^{\nu} \mathbf{E}(\ln(b_{x_u}(o_u)) \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t})\end{aligned} \tag{2.35}$$

The last part of the equation, specific for the hidden Markov model, doesn't contain q, so the result of maximization will be the same as in equation (2.32)

for Markov process. Lack of the direct states observations makes the computation of the expected values η_{ij} and τ_i more challenging. We need to obtain it just from the knowledge about the emission sequence. First we will derive the formula for the expectation $\mathbf{E}(\eta_{ij} \mid O, T, \hat{Q}_t)$ which can be expressed as the sum through all the possible state sequences.

$$\mathbf{E}(\eta_{ij} \mid O, T, \hat{Q}_t) = \sum_{x_0,\ldots,x_\nu} \mathrm{P}(x_0, \ldots, x_\nu \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t}) \mathbf{E}(\eta_{ij} \mid x_0, \ldots, x_\nu, \hat{\mathbf{Q}}_\mathbf{t}) \tag{2.36}$$

The expected value formula can be rewritten as the sum of the partial intervals.

$$\sum_{x_0,\ldots,x_\nu} \mathrm{P}(x_0, \ldots, x_\nu \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t}) \sum_{u=1}^{\nu-1} \mathbf{E}(\eta_{ij} \mid x_u, x_{u+1}, \hat{\mathbf{Q}}_\mathbf{t}) \tag{2.37}$$

Because the Markov condition hold we can interval-wise divide the complex probability function and for every of the time intervals sum through all the possible states k, l at its edges.

$$\sum_{u=0}^{\nu-1} \sum_{k=0,l=0}^{n} \mathrm{P}(x_u = k, x_{u+1} = l \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t}) \mathbf{E}(\eta_{ij} \mid x_u = k, x_{u+1} = l, \hat{\mathbf{Q}}_\mathbf{t}) \tag{2.38}$$

The expected value of sojourn time $\mathbf{E}(\tau_i \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t})$ can be derived equivalently to get

$$\sum_{u=0}^{\nu-1} \sum_{k=0,l=0}^{n} \mathrm{P}(x_u = k, x_{u+1} = l \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t}) \mathbf{E}(\tau_i \mid x_u = k, x_{u+1} = l, \hat{\mathbf{Q}}_\mathbf{t}). \tag{2.39}$$

In Subsection 2.3.1 we have shown how to count $\mathrm{P}(x_u = k, x_{u+1} = l \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}_\mathbf{t})$. The remaining part, the computing of $\mathbf{E}(\eta_{ij} \mid x_u = k, x_{u+1} = l, \hat{\mathbf{Q}}_\mathbf{t})$ and $\mathbf{E}(\tau_i \mid x_u = k, x_{u+1}) = l, \hat{\mathbf{Q}}_\mathbf{t})$ is presented in following two subsections.

2.3.5 The End-State Conditioned Expectations

We have previously described how to use the end-state conditioned expectations η_{ij} and τ_i to compute estimation of the jump-rate matrix $\mathbf{Q}$ by maximizing of its resulting probability (2.32). In this section we will discuss them more deeply and show the possible ways how to compute their values.

The Markov process we are describing is time-homogeneous. So the time offset at the beginning of the interval is not important and we can anchor it toward zero. The two following expectations are equivalent: $\mathbf{E}(\eta_{ij} \mid x_{t_1} = k, x_{t_2} = l, \mathbf{Q}) = \mathbf{E}(\eta_{ij} \mid x_0 = k, x_{t_\Delta} = l, \mathbf{Q})$ where $t_\Delta = t_2 - t_1$. It can help

us to save some computations as we can store the values and reuse them if needed.

The expectations can be expressed in following way (see [12]).

$$\mathbf{E}(\eta_{ij} \mid x_0 = k, x_t = l, \mathbf{Q}) = \frac{q_{ij}}{\mathrm{p_{kl}}(t)} \int_0^t \mathrm{p_{ki}}(x)\mathrm{p_{jl}}(t-x)dx \tag{2.40}$$

$$\mathbf{E}(\tau_i \mid x_0 = k, x_t = l, \mathbf{Q}) = \frac{1}{\mathrm{p_{kl}}(t)} \int_0^t \mathrm{p_{ki}}(x)\mathrm{p_{il}}(t-x)dx \tag{2.41}$$

We already know the solution of differential equation for counting the jump probabilities $\mathrm{p}(t)$. So we can transform the previously stated integrals to the form as shown in equation (2.41). We will denote the integrals as τ_{kl}^{ij}. For the equation (2.41) it will be similarly τ_{kl}^{ii}.

$$\tau_{kl}^{ij} = \int_0^t \mathrm{p_{ki}}(x)\mathrm{p_{jl}}(t-x)dx = \int_0^t e^{\mathbf{Q}x}[k,i]e^{\mathbf{Q}(t-x)}[j,l]dx \tag{2.42}$$

2.3.6 Methods to Solve End-State CE

There are several methods for solving of the proposed integral (2.42).

If the matrix $\mathbf{Q}$ is diagonalizable, the closed form can be obtained by *Eigen decomposition* based algorithm as describe in [20]. However as claimed in [19], the matrices are not diagonalizable in general. That makes the method often inappropriate for use in Baum-Welch algorithm, because the jump-rate matrices can take general form during the computation process.

The another method of *Uniformization* is based on truncating of infinite sum defining the exponential as described in [11]. The main advantage of the method is that all matrices can be precomputed and later reused, so it doesn't need any additional matrix multiplication [19]. The use of hard or soft method 2.3.1 influence the time performance of the algorithm with hard method being faster, because the posterior state probability table will become sparse. The method's running time also highly depends on the data and values of $\mathbf{Q}$ matrix. Depending on it the sequence truncating point can differ significantly.

The method referred to as *Expm* is using *Matrix exponentiation* to count the proposed integrals. The method is described in article [24]. The advantage is that just by one exponentiation of auxiliary matrix, we can get results for all pairs of k, l end-states expectations. The auxiliary matrix $\mathbf{W}$ is in the form $\mathbf{W} = \begin{pmatrix} \mathbf{Q} & \mathbf{I}_{ij} \\ 0 & \mathbf{Q} \end{pmatrix}$ where $\mathbf{I}_{ij}$ is the square matrix of size $\mathbf{Q}$ composed by zeros and the only one at position i, j. Indices i, j refer to the states of intermediate visit as defined in the previous subsections. The following equality holds $\int_0^t e^{\mathbf{Q}x}[k,i]e^{\mathbf{Q}(t-x)}[j,l]dx = \int_0^t (e^{\mathbf{Q}x}\mathbf{I}e^{\mathbf{Q}(t-x)})[k,l]dx$. In the article [24] is shown that $\int_0^t e^{\mathbf{Q}x}\mathbf{I}e^{\mathbf{Q}(t-x)} = e^{\mathbf{Q}t}[0:n][n+1:2n]$ where n is the size of the matrix $\mathbf{Q}$.

We have got the matrix exponential form, for which conventional methods for matrix exponentiation can be used. For example scaling and squaring method using Padé approximants as described in [10].

We have decided to implement *Expm* method, because it is the most stable and the fastest from the mentioned methods [19]. We have implemented it with both soft and hard maximizer as we have found them interesting subject to exploring. We have also implemented two variants of algorithm dependent on the input data characteristic: float-interval and more time efficient, but less general integer-interval variant. They will be described more properly in next section.

2.4 Algorithms for CT-HMM parameters estimation

This section roughly describe the algorithm 2.1 for CT-HMM learning - expectation maximization approach estimating the jump-rate matrix $\mathbf{Q}$ also denoting as continuous-time Baum-Welch algorithm. We only show the parts specific for continuous-time model. The estimation of the other parameters $(\boldsymbol{\pi}, \mathbf{B})$, forward and backward algorithm so as the single and double state probabilities can be easily derived from DT-HMM algorithm explained in Section 2.2. It is also described in details in article [23].

The base of the algorithm is described in Algorithm 2.1 referring to its most important frame parts. The algorithm shows training at a single observation and time sequence. It is for the sake of simplicity. The training by more complex dataset can be made by iterating trough all the sequences. There are two methods affecting the way of convergence - **soft** and **hard** 2.3.2. The soft method uses the forward-backward algorithm and the hard uses the Viterbi algorithm to count posterior state probabilities (single and double-state probability tables).

The part that counts the end-state conditioned probabilities is described separately and it compounds two variants different by implementation and computational complexity. We refer to them as **float-interval** variant 2.2 (the name is derived from ability of the algorithm to process float time-intervals) and **integer-interval** variant 2.3 (it can only process the data with integer time intervals). As showed in the Chapter 4, the second mentioned variant is significantly faster 4.2 and it probably do not produce any critical numerical errors 4.3.

The both variants of the described algorithms iterate through all the pairs of hidden states. Namely at 3^{rd} line of **float-interval** algorithm 2.2, and at 2^{nd} and 5^{th} line of **integer-interval** algorithm 2.3. It can be optimized for sparse jump-rate matrices $\mathbf{Q}$. We can simply omit the pairs for which is the jump rate equal zero. As once being set to zero the value can't change by the run of the algorithm.

Algorithm 2.1 CT-HMM Parameters Estimation

Input data: observation sequence $\mathbf{O} = (o_1, o_2, \ldots, o_\nu)$, time sequence $\mathbf{T} = (t_1, t_2, \ldots, t_v)$
Input parameter: n - number of hidden states
Output: $\hat{\mathbf{Q}}$ - estimation of jump-rates matrix $\mathbf{Q}$
procedure BAUM-WELCH
    $\hat{\mathbf{Q}} \leftarrow$ randomly generated jump-rate matrix
    $\mathbf{T}_\Delta \leftarrow$ find all distinct time intervals from $\mathbf{T}$
    repeat
        $\mathbf{P}(t_\Delta) \leftarrow$ count state transition probabilitiese$^{\hat{\mathbf{Q}}t_\Delta}$ for each t_Δ from $\mathbf{T}_\Delta$
        $\boldsymbol{\Xi} \leftarrow \boldsymbol{\Xi}[u,k,l] = \mathrm{P}(X_{t_u} = k, X_{t_{u+1}} = l \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}), u \in 1, \ldots, v-1$
        ▷ The double state probability can be counted either from *forward* and *backward* probability tables - **soft** method, or from *viterbi* algorithm most probable state sequence - **hard** method. The $\mathbf{P}(t_\Delta)$ is used for counting of state transition probabilities.
        $\boldsymbol{\Xi}_\Delta \leftarrow$ sum together the matrices $\boldsymbol{\Xi}[u]$ with the same length of the time interval $t_{u+1} - t_u$ that equals $t_\Delta \in \mathbf{T}_\Delta, u \in 1, \ldots, v-1$
        Count end-state probabilities $\mathbf{E}(\eta_{ij} \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}})$ and $\mathbf{E}(\tau_i \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}})$
        ▷ It can be counted by one of the two *expm* based algorithm variants described later, **float-interval** algorithm 2.2 or **integer-interval** algorithm 2.3.
        $\hat{\mathbf{Q}} \leftarrow \hat{\mathbf{Q}}[i,j] = \frac{\mathbf{E}(\eta_{ij}|\mathbf{O}, matrT, \hat{\mathbf{Q}})}{\mathbf{E}(\tau_i|\mathbf{O}, matrT, \hat{\mathbf{Q}})}$, and $\hat{\mathbf{Q}}[i,i] = -\sum_{i \neq j} \hat{\mathbf{Q}}[i,j]$ [a]
    until probability is satisfiable

[a] (2.32)

Algorithm 2.2 The Expm Based Algorithm for Counting End-States Conditioned Probabilities (Float-Interval variant)

procedure FLOAT-INTERVAL
    for all $t_\Delta \in \mathbf{T}_\Delta$ **do**
        for all pair of states $i, j \in \mathcal{S}$ **do**
            $\mathbf{D}_{ij} \leftarrow (e^{t_\Delta \mathbf{W}})[0:n][n+1:2n]$, where $\mathbf{W} = \begin{pmatrix} \mathbf{Q} & \mathbf{I}_{ij} \\ 0 & \mathbf{Q} \end{pmatrix}$
            $\mathbf{D}_{ij} \leftarrow \frac{\mathbf{D}_{ij}}{\mathbf{P}(t_\Delta)}$
            if i is equal to j **then**
                $\mathbf{E}(\tau_i \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}) = \sum_{k,l \in \mathcal{S}} \boldsymbol{\Xi}_\Delta[t_\Delta, k, l] \mathbf{D}_{ii}[k,l]$
            else
                $\mathbf{D}_{ij} \leftarrow \mathbf{D}_{ij} \mathbf{Q}[i,j]$
                $\mathbf{E}(\eta_{ij} \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}) = \sum_{k,l \in \mathcal{S}} \boldsymbol{\Xi}_\Delta[t_\Delta, k, l] \mathbf{D}_{ij}[k,l]$

Algorithm 2.3 The Expm Based Algorithm for Counting End-States Conditioned Probabilities (Integer-Interval variant)

1: **procedure** Integer-Interval
2: **for all** pair of states $i, j \in \mathcal{S}$ **do**
3: $\mathbf{D}_{ij} \leftarrow (e^{\mathbf{W}})[0:n][n+1:2n]$, where $\mathbf{W} = \begin{pmatrix} \mathbf{Q} & \mathbf{I}_{ij} \\ 0 & \mathbf{Q} \end{pmatrix}$
4: **for all** $t_\Delta \in \mathbf{T}_\Delta$ **do**
5: **for all** pair of states $i, j \in \mathcal{S}$ **do**
6: $\mathbf{D}'_{ij} \leftarrow \mathbf{D}_{ij}^{t_\Delta}$ ▷ Use *square and multiply* algorithm to effectively compute power of $\mathbf{D}_{ij}$. Notice that every t_Δ needs to be integer.
7: $\mathbf{D}'_{ij} \leftarrow \frac{\mathbf{D}'_{ij}}{\mathbf{P}(t_\Delta)}$
8: **if** i is equal to j **then**
9: $\mathbf{E}(\tau_i \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}) = \sum_{k,l \in \mathcal{S}} \mathbf{\Xi}_\Delta[t_\Delta, k, l] \mathbf{D}'_{ii}[k, l]$
10: **else**
11: $\mathbf{D}'_{ij} \leftarrow \mathbf{D}'_{ij} \mathbf{Q}_{[i,j]}$
12: $\mathbf{E}(\eta_{ij} \mid \mathbf{O}, \mathbf{T}, \hat{\mathbf{Q}}) = \sum_{k,l \in \mathcal{S}} \mathbf{\Xi}_\Delta[t_\Delta, k, l] \mathbf{D}'_{ij}[k, l]$

2.5 Complexity

2.5.1 Time Complexity

By complexity we refer to the complexity of one iteration of EM-algorithm. All the iterations are equivalent so to get full runtime complexity you can multiply it by the iteration number.

The number of hidden states is denoted as n, the overall length of the training sequences as ν.

- **Discrete-Time Baum-Welch Algorithm**
  In one iteration of the discrete version of the algorithm the following main procedures take part:
  - Forward algorithm (Subsection 2.2.1)- complexity $\mathcal{O}(n^2\nu)$.
  - Backward algorithm (Subsection 2.2.1) - complexity $\mathcal{O}(n^2\nu)$.
  - Computation of single state probabilities (Subsection 2.2.2) - complexity $\mathcal{O}(n\nu)$. Summing and normalizing the forward and backward probabilities, that are of size $n\nu$.
  - Computation of double state probabilities (Subsection (2.23)) - complexity $\mathcal{O}(n^2\nu)$. Constant-time computation of probabilities for every pair of states in every time.
  - Various summing operation - complexity $\mathcal{O}(n^2\nu)$. Just linear pass trough the arrays, the biggest one has size $n^2\nu$.

As the operations are consecutive, the whole asymptotic complexity of the algorithm is the maximum - $\mathcal{O}(n^2\nu)$.

- **Continuous-Time Baum-Welch Algorithm**

The complexity of the continuous-time algorithm depends on the chosen variant of end-state conditioned probabilities counting. So we distinguish *integer-interval* 2.3 and *float-interval* 2.2 variants of the algorithm (later just algorithm). In the first one is the time difference among two consecutive observations integral, in the second it may be float. Float algorithm can be used for the integral intervals, but not vice-versa.

The *soft* and *hard* method of convergence 2.3.2 doesn't impact the asymptotic complexity, as both the forward-backward algorithm and the Viterbi algorithm have the same $\mathcal{O}(n^2\nu)$ complexity.

Let r be the number of different time intervals. In some cases it can be much more restrictive than $\nu - 1$ - the number of all intervals.

The first part of the algorithm is similar to the discrete-time version 2.5.1. With the difference, that we need to pre-compute the transition probabilities matrix for every time interval. Than we can simply use the discrete-time algorithms, as if dealing with heterogeneous DT-HMM.

- Pre-compute transition probability matrices - complexity $\mathcal{O}(rn^3)$. Call expm algorithm $\mathcal{O}(n^3)$ [21] for every distinct time interval.
- Subpart similar to discrete-time algorithm - complexity $\mathcal{O}(n^2\nu)$ 2.5.1.
- Count end-state conditioned expectations 2.3.5:
  1. **integer-interval algorithm**:
     - Preprocessing end-state conditioned expectation - complexity $\mathcal{O}(n^5)$. In this step we compute all the integrals(2.42) for unit time interval. It needs to call expm method $\mathcal{O}(n^3)$ at matrix $\mathbf{W}$ for every i and j 2.3.6.
     - Count matrices for every time interval - complexity $\mathcal{O}(rn^5 \log(t_{\max}))$, where $t_{\max}$ is the longest time interval among two consecutive observations.
       For every time interval, we go trough n^2 matrices, that we have already precomputed and we count its value for actual interval by using square and multiply approach.
  2. **float-interval algorithm**:
     - Count end-state conditioned expectation for every interval - complexity $\mathcal{O}(rn^5)$. We needs to call expm method $\mathcal{O}(n^3)$ at matrix $\mathbf{W}t$ for every i and j, where t is the length of the interval 2.3.6.

The overall time complexity of the iteration is $\mathcal{O}(rn^5 + n^2\nu)$, using the *float-interval* algorithm and $\mathcal{O}(rn^5 \log(t_{\max}) + n^2\nu)$ using the *integer-interval* algorithm. However, it doesn't necessary mean the float-algorithm runs faster. As showed in the time measurement experiment 4.2 the multiplicative constant for *expm* algorithm is very high. The experiment has shown that there also exists logarithmic pseudo-dependence on parameter $t_{\max}$. It is bounded to counting of Padé approximants. The matrices containing bigger numbers in average need to use the approximant of higher order, which is more complex to compute in the terms of multiplicative constant. We have shown, that in the real practice, it is generally much more computationally efficient to use the integer-interval algorithm.

Till now, we have assumed the full jump-rate matrix $\mathbf{Q}$. However, in some cases we may intentionally forbid some state-to-state transitions. Such a restriction also manifests itself in the term of complexity. Let's define m as the number of allowed state-to-state transition pairs. Than the new complexity shrinks to $\mathcal{O}(rn^3m+n^2\nu)$ or $\mathcal{O}(rn^3m \log(t_{\max})+n^2\nu)$ respectively. Notice that the possible value for m which makes sense can be in the interval (n, n^2), depending on the sparsity of the matrix $\mathbf{Q}$. So it can effectively divide the full matrix complexity by n.

2.5.2 Memory Complexity

In the discrete-time model algorithm the biggest array is the one for storing double state probabilities so the memory complexity is equal to its size - $\mathcal{O}(n^2\nu)$.

In the continuous-time model algorithm there are more arrays of size $n^2\nu$ and the array of size n^4 to store precomputed matrix exponentials. So the final memory complexity is $\mathcal{O}(n^2\nu + n^4)$. We can spare some memory with sparse jump-rate matrix $\mathbf{Q}$, because we only need to store the precomputed matrices for nonzero elements of $\mathbf{Q}$. The memory complexity for the sparse matrix is $\mathcal{O}(n^2\nu + n^2m)$ where m is the number of allowed state-to-state transition pairs.

Chapter 3

Implementation

The chapter discuss chosen technology and specific details of implementation. The code, tutorial and documentation to the library can be found in supplementary materials.

3.1 About Library

The goal was to create a library, that can be used by broad range of users. Such a library should be written in the well known programming language. It should be easy to install and also it should have introductory tutorial that would help to get orientation in the most of the library functionality.

To accomplish the established goal we have chosen to create *Python* library. Specifically, the development was done under *Python 3.5*. The Python is highly used among machine-learning community and the library is easily accessible by downloading from *Pip* [1] or *GitHub* [2]. The continuous-time hidden Markov model parameter estimation is computationally highly demanding. The using of the plain *Python* would make it cumbersome. To deal with the issue, we have used libraries *NumPy* - for effective work with array and matrices and *SciPy* [14] for the implementation of *expm* algorithm [9]. To make the computational performance even higher, we have decided to code all the most demanding parts in *Cython* [4]. Cython is a compiled language that generates *CPython* extension modules, so that the code-parts written in the language can get *C*-like performance. The advantage is its easy integration into the Python module.

The best way how to learn to use the library is follow the code examples. That's way we have decided to create *IPython* notebook with library usage examples, covering all the main use-cases of the library. Reading it, the fol-

[1] `https://pypi.python.org/pypi/hmms`
[2] `https://github.com/lopatovsky/HMMs`

lower can get the understanding how to correctly use the library and learn, what features are supported.

3.2 Library Functionality

The library implements discrete-time and continuous-time hidden Markov models. Most of the functions are available for both models, sometimes their input parameters or implementation details can differ depending on the model characteristic.

3.2.1 Models Initialization

New model can be created directly by passing its parameter. $\boldsymbol{\theta} = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ for the discrete-time model 1.2 and $\boldsymbol{\theta} = (\mathbf{Q}, \mathbf{B}, \boldsymbol{\pi})$ for the continuous-time model 1.4. The jump rate matrix $\mathbf{Q}$ can be defined as sparse by setting its elements to zero. The algorithm automatically recognizes such elements and spares the computational demands 2.4.

Other way to initialize the models is set the parameters randomly. The procedure just need to enter the number of hidden states and number of observation symbols. The random initialization is useful for training of the new models to match the dataset or for the experimental purposes. The random initialization of the continuous-time model is optional by uniform or exponential distribution.

It is also possible to read model parameters from file or write them to the file. We have used standard *.npz* format for storing multiple NumPy matrices or arrays.

We can also convert the continues-time parameters and use them for initialization of discrete-time model. The conversion is made by exponentiation of matrix $\mathbf{Q}$ for any given time interval t as $\mathbf{A} = e^{\mathbf{Q}t}$.

3.2.2 Probability and Statistical Functions

Giving the observation sequence (in the continuous-time model together with the time sequence) or even whole dataset of sequences we can use the function to count the probability that the data were generated by given model. The function use the forward algorithm described in Section 2.2.1. Similar function takes additionally to the observation and time also the hidden state sequence and count the probability for all the sequences given the model.

The library enables us to generate our own data sequences using the model parameters. We can even directly create whole artificial dataset. That may be useful for experimenting with the models. The continuous time model moreover needs to generate time sequences. We have chosen to generate them with the exponentially distributed time intervals. The parameter λ of the exponential distribution can be set as parameter.

We can get most likely state sequence and its probability by using the Viterbi algorithm described in Section 2.2.3. The resulting sequence can be plotted. It can be useful for the beginners to get the basic intuition about the algorithm functionality.

3.2.3 EM Algorithms

We use EM algorithms to estimate the models parameters. The Baum-welch algorithm for the discrete-time model described in section 2.2.4 and its continuous-time variant (Section 2.3.4). Some of the algorithms sub-function can be used separately - the forward and backward algorithm (2.2.1), the Viterbi algorithm (2.2.3), algorithms for counting single and double posterior state probabilities (described in Sections 2.2.2, 2.2.4 and 2.3.1). The Baum-welch algorithm function can optionally return the probability estimation sequence containing the measures for every iteration. It is useful for plotting algorithm convergences and deciding when is the best time to stop the algorithm. To use the algorithm for the training we need to pass the dataset of observation and for continuous-time model also time sequences. They can be passed in the form of NumPy matrices or as list of NumPy arrays. The second approach enable the training with sequences of different lengths.

The continuous-time algorithm is implemented in multiple variants. The soft or hard method can be used as described in Section 2.3.2. The optional boolean parameter can be set to enable fast convergence. The fast convergence use, if possible, the integer variant of the algorithm. If the option is set to false, algorithm always use the float variant. The integer and float variant of the algorithm are described in Section 2.4. Soft method and fast option are set to true by default, because they are given best expected results and time efficiency 4.

3.3 Implementation Details

3.3.1 Logarithmic Probabilities

The probability values obtained in the hidden Markov models can often get extremely small. Using the standard data-types, it could fast reach the underflow. Even the short sequences of few hidden states can underflow, if proper conditions are met.

The use of infinite data-type would be possible. However, it would make the algorithm unacceptably computationally and memory expensive, as the bit-length of the numbers can grow linearly with the number of probabilities to multiply.

More appropriate is to use log-probabilities (3.1). This makes the numbers grows linearly, thus their bite-length logarithmically. Under this conditions

is the underflow for common data types and the feasible size of the data-set almost impossible.

$$\log(\mathrm{p}_1\mathrm{p}_2) = \log(\mathrm{p}_1) + \log(\mathrm{p}_2) \tag{3.1}$$

3.3.2 "Log-sum-exp" Trick

The log-probabilities work great for multiplication and division, but the problem appears, once we need to sum the probabilities. The so called "log-sum-exp" trick can be used to deal with the issue.

Let's have probabilities $\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_n$. Now we want to make the sum of them. It is fairly simple, when they are usual float numbers: $\sum_{i=1}^{n} \mathrm{p}_i$. But it can get trickier, if they are stored in the form of log-probability. We denote log-probability $\log(\mathrm{p}_i)$ as a_i. Than we can write the sum in the following form. Notice that the output is also log-probability.

$$\log(\sum_{i=1}^{n} e^{a_i}) \tag{3.2}$$

We can't count it directly as the e^{a_i} can be very small and underflow. The idea of the trick is to take the biggest element out of the sum, as a multiplier and move all the other in relative position toward it. Let's $b = \max_i a_i$. Than we can write the previous equation (3.2) in the form of equation (3.3).

$$\begin{aligned} & \log(e^b \sum_{i=1}^{n} e^{a_i - b}) = \\ = \quad & b + \log(\sum_{i=1}^{n} e^{a_i - b}) \end{aligned} \tag{3.3}$$

This approach only dismiss the elements that are relatively for data-type maximal-range smaller than the biggest element b. Also the sum of more such diminished elements, under normal circumstances, never have an impact on the precision of computation - comparing the data-type imprecision.

3.3.3 The Sparse Jump-Rate Matrix

The time-complexity of the CT-HMM EM algorithm can shrink, when using the sparse matrix, as described in Section 2.5.1. The implementation take advantage of it, but not in the term of the theoretical complexity, since it is still cycling trough all the matrix elements. The algorithm just stop to perform the body of the cycle. Such solution may seems to be less effective in the terms of theoretical time-complexity. The number of states under the real conditions is not bigger than hundreds, so just cycling through all the matrix elements is utterly negligible, comparing the matrix exponentiations taking part in the body of the cycle.

Chapter 4

Experiments

The definition of the hidden Markov models may seem to be simple. However, in consequences it covers many non-trivial characteristics that needs some intuition to be understood correctly. This chapter is purposed to discuss such characteristic, and simultaneously try to support the allegations by presenting the experiments.

The later part of the chapter contains the experiments that are connected to the implemented methods or HMM theory.

1. **Discrete-time vs. Continuous-time Hidden Markov Model**
   Comparison of the discrete and continuous-time HMM convergences at the equivalent dataset.

2. **Computational Complexity**
   Examination of the time complexity and comparison with theoretical expectations.

3. **Numerical Stability**
   Comparison of the *float-interval* and *integer-interval* method from the point of numerical stability.

4. **Soft vs. Hard Method**
   Comparison of Soft and Hard method from two different points of view.

5. **Hidden States Number**
   Examination of the models behavior, when trained with variable numbers of hidden states.

Notice that the *performance ratio* used to measure quality of convergence (unless otherwise stated) refer to the ratio of the logarithmic probability estimations of the trained model toward the original model (the one, which was used to generate dataset). So the lower the ratio is, the better fit the model

the dataset. The performance ratio of the model before the first iteration is often cut of the graph, because it can take too extreme value.

All time measurement experiments were performed on processor Intel® Core™ i5-2520M CPU @ 2.50GHz.

4.1 Discrete-time vs. Continuous-time Hidden Markov Model

In the theoretical part we are describing two models: discrete-time and continuous-time hidden Markov model. The model convergence test compares how fast they can learn from generated dataset (converge to the local optima) and we have checked, if there is some significant difference among the converged optima values.

As the continuous-time model takes, except the observations, also the time vectors its domain range is obviously much broader. Still, it is possible to train it at dataset for discrete-time model simply by adding artificial equidistant time vectors.

Description

In the first part of the experiment we have used the dataset generated by the model which we have created artificially to suit its purpose well. The model contains three hidden states and three observation symbols. Comparing the general randomly generated model (model with the parameters generated by exponential distribution), it takes advantage of its more extreme position in the parameter space. So its distance to the majority of the random generated parameters is bigger. This makes the problem of convergence harder and in final produces nicer graphs, unfolding more of the process characteristics.

Training dataset consists of 50 vectors. Each of them compounding 50 observation. For the continuous-time model we have add the same number of time vectors, with unit time interval lengths.

For the experiment we have conducted ten runs of Baum-Welch algorithm always starting from different random position, and averaged the results. The starting positions were equivalent for both models. We have run 150 iterations for every convergence.

The convergences for the single runs of the first experiment can be seen in Figure 4.1 and then their average in Figure 4.2.

To make the results of the experiment more general we have continued with the second part in which we have repeated the procedure of the experiment with the same parameters five times on the models which parameters were created by hand in some specific real-life like manner, and five times on randomly generated models.

Observations

The convergences for the single runs of the first part of the experiment can be seen in Figure 4.1 and then their average in Figure 4.2. Figures 4.3 and 4.4 show the results of the second second part of experiment.

In the figures with the average convergences 4.2 and 4.3 we can see the continuous-time model converge slightly slower. However both models seem to converge to the same value, closing the gap by every iteration. The single run convergences 4.1 show the irregular speed of convergence of both models. The better performance at the beginning doesn't ensure it will not be over-performed by other instances later. The convergence slopes seem to be more regular towards the end of convergence.

The convergences to the datasets generated by the random models start mostly with visibly lower performance ratio than the convergence lines generated by specific hand-made models 4.3. We can also see that the models mostly over-performed the original model. For example, the average convergence from the first experiment 4.2 is converging to the performance ratio ~ 0.99481.

Conclusions

The slower convergence of continuous-time model is bounded with different character of the model. Since in the discrete-time model we always know the precise time when the state-change can happen (always at observation point), in continuous-time model it can happen in any time in between as well. Thus, computing of the new estimated parameters in maximization step needs to deal with more uncertainty. In continuous-time model is uncertainty not only caused by indirect observation of hidden states, but also by possible state changes in between of the observations. As we have any observations from that area, the probability estimation is counted by parameters from the last iteration, what is creating momentum in favor of old parameters and slow down the convergence process. The longer is the time interval, the stronger is the momentum.

In the second experiment we have shown, how characteristic of the searched model influenced training complexity. The randomly generated models usually generate datasets with higher entropy. Such datasets are closer to converge from most regions of the parameter space.

The over-performing of the original model is caused by the overfitting. The small size of dataset was not sufficient to cover characteristic of the model completely, instead the trained models were learned to fit its imprecision. It is not a problem for our experiment as the original models were chosen more-less randomly and it fills in its purpose. The over-performing would probably gradually disappear with growing size of the data-set. It is shown in experiment 4.5.1.

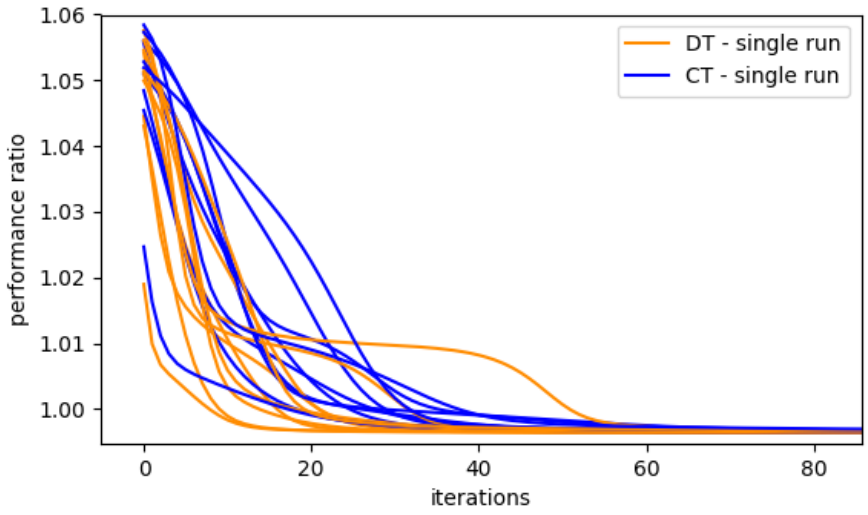


Figure 4.1: Models convergences

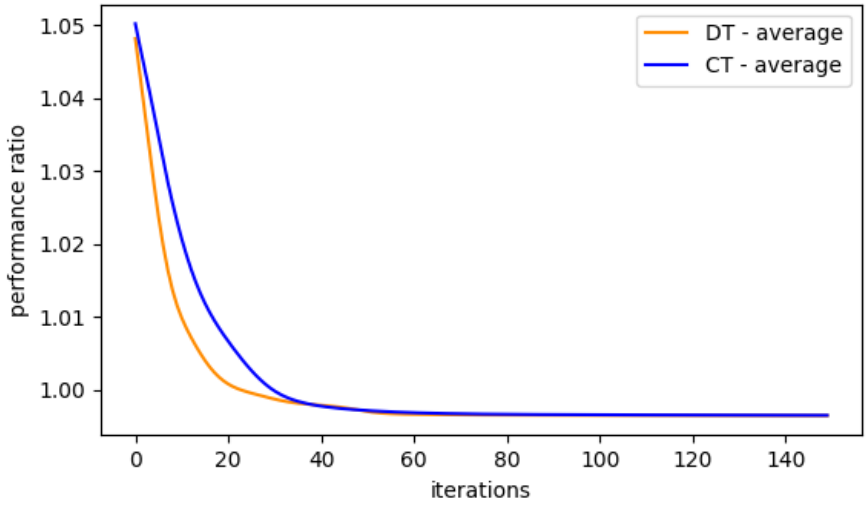


Figure 4.2: Average of the convergences from various random starting points

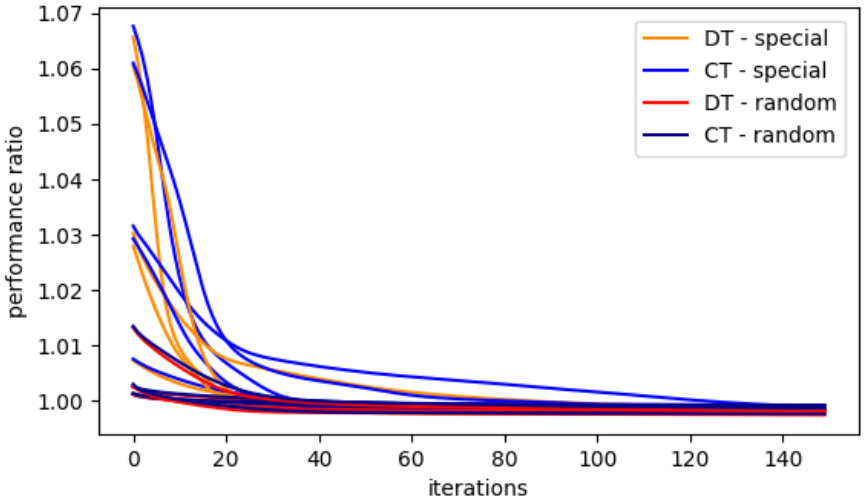


Figure 4.3: Average of the convergences for various generated datasets

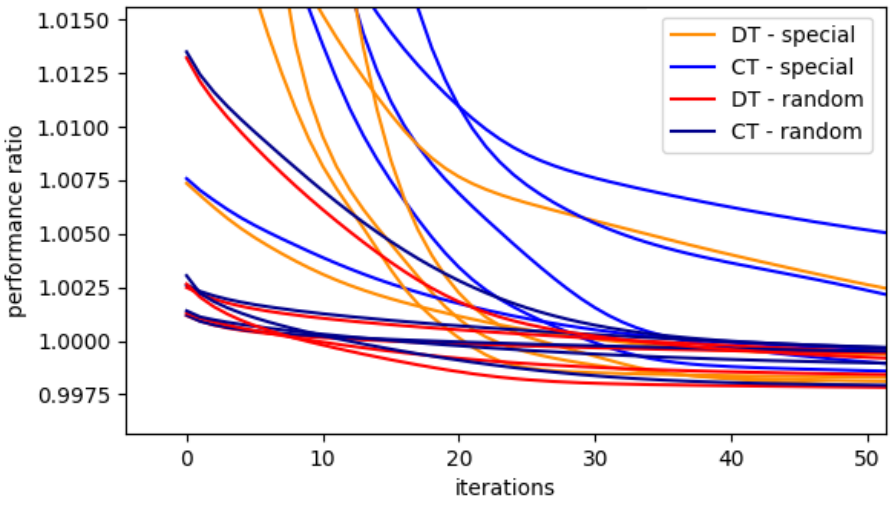


Figure 4.4: Average of the convergences for various generated datasets (zoom)

All of the training conducted during this experiment converged close to global optima. We have used simple three-states models. Behavior of more complex models will be examined later in Section 4.5.2.

4.2 Computational Complexity

In this section we are comparing time-performance of the float and integer-interval variants of CT-HMM as described in implementation part 2.4. In two following subsections we experiment with variable hidden states number and maximal time interval and examine, if the measured values correspond to their theoretical expectations described in Section 2.5.

4.2.1 Variable Hidden States Number

The hidden states number, referred to as n is key parameter in overall algorithm complexity. Due to computing of the matrix exponentials $\mathcal{O}(n^3)$ for every pair of end states, it influences the final time-complexity by its fifth power n^5. The upper bound theoretical complexity dependence on n is the same for both of the variants. However, the float-algorithm counts the matrix exponential by *expm* method separately for every time interval, while the integer-algorithm counts it only once and then use the matrix multiplication to get the individual results 2.5.1. In the experiment we measure and compare times of most costly algorithm parts, and notice how big is the portion of the overall computational complexity consumed by them.

Description

We have trained the models with the variable number of hidden states. To minimize the impact of the other factors we have let the number of output variables to be constantly 10 and we have used the same randomly generated dataset for both algorithm variants. The integral times intervals were generated by exponential distribution with parameter 0.5. To minimize the time measurements error we have always run 10 iterations of algorithm and repeated the overall procedure 5 times.

To see how the size of the dataset influence time demand, we have conducted all the described experiments twice. Once with the *small* dataset - compounding 10 sequences of 10 observations and once with the *big* dataset - compounding 10 sequences of 100 observations.

Observation

In Figure 4.5 we can observe that the float-interval algorithm is always slower. The prevalent source of complexity is the *expm* method. In the float-interval

algorithm over the small dataset are all other parts of the algorithm almost neglectable comparing to it.

Comparing to the *expm* are the time demands of *square and multiply* algorithm considerably smaller. It is called multiple times over different time intervals and still takes smaller portion of the time as the single *expm* call.

The increase of the dataset size caused the larger gap among the whole algorithm time complexity and its measured subparts - either *expm* or the sum of *expm* and *square and multiply* depending on the algorithm 4.5b.

Conclusion

We have only changed the number of states and let the dataset to be the same so the $t_{\max}$ parameter from the integer-interval complexity $\mathcal{O}(rn^5 \log(t_{\max}))$ is fixed as constant. This makes both algorithm variants to have same complexity $\mathcal{O}(rn^5)$. The measurements have shown the big multiplicative constant of *expm* method. That's way it is almost always better to use the integer-interval algorithm. (Its numerical stability is tested later in Section 4.3)

The most of the computational power is used to count matrix exponentials *expm*. In most cases it is the predominant cause of algorithm "slowness". It makes not much sense to edge-optimize other parts of the algorithm. Instead, the faster implementation of *expm*, its parallelization or replacement with other method could spare the significant portion of time.

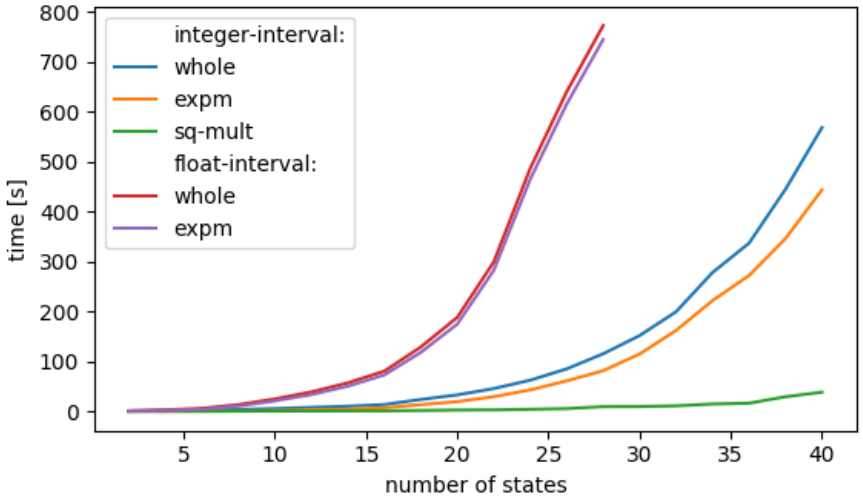
Training over the big dataset has decreased the percentage of the time spent by *expm* method. It is because the higher demand of other algorithm parts. Potentially, it can happen that this gap would overgrowth the *expm* part. The complexity of the algorithm part that call *expm* method only depend on r, the number of different time intervals, however there are parts of the algorithms with complexity $O(n^2T)$ 2.5.1 where T is number of all time intervals. So for the huge dataset with lower number of hidden states and many identical time intervals can this part be the predominant cause of the complexity.

4.2.2 Variable Maximal Time Interval

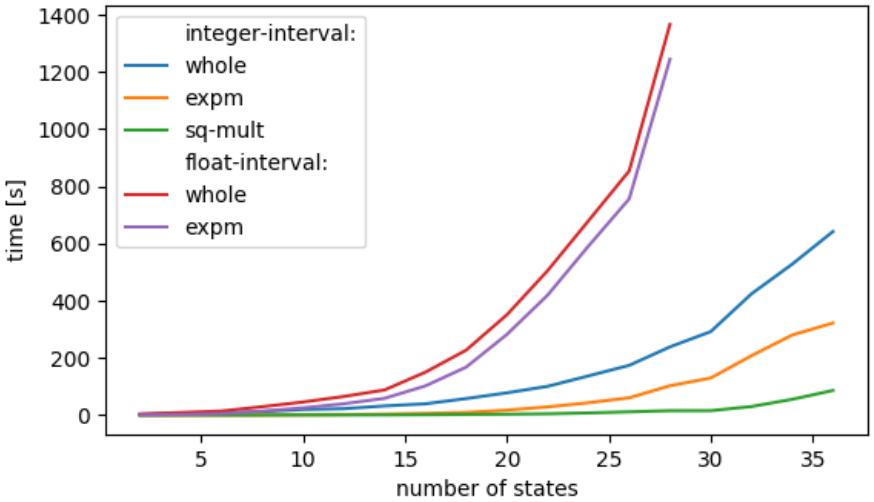
The other parameter presented in the theoretical time complexity of the integer-interval algorithm $\mathcal{O}(rn^5 \log(t_{\max}))$ is $t_{\max}$, maximal length of the time interval. On the oder side the parameter is not part of the float-interval algorithm complexity term $\mathcal{O}(rn^5)$ 2.5.1. The experiment aims to observe how the variable value of $t_{\max}$ changes the computational demand of the algorithm.

Description

In the experiment we have set $t_{\max}$ as the variable taking values of powers of two from 2^1 to 2^{56}. The use of higher values (even floats) was impossible, because of conversion to 64-bites integer used in *SciPy expm* method that



(a) Small dataset



(b) Big dataset

Figure 4.5: Time complexity of two algorithm variants and their subparts

eventually causes the integer overflow. We have generated the dataset of 10 sequences by 10 observations. There was always at least one time-interval of length $t_{\max}$ and all other were chosen uniformly from interval $(1, t_{\max})$. We have trained the model of 10 hidden states and 10 observation variables. To smooth the influence of the time measurement imprecision we have run 10 iterations and repeated the whole experiment 5 times.

Observation

Contrary to initial expectation both float and integer interval algorithm computational times are growing seemingly logarithmically with increasing $t_{\max}$. The cause of growth in float-interval algorithm is *expm* method, in integer-interval algorithm it is caused mainly by *square and multiply* method. There is a steep growth of the float-interval algorithm time demands present for small $t_{\max}$ values.

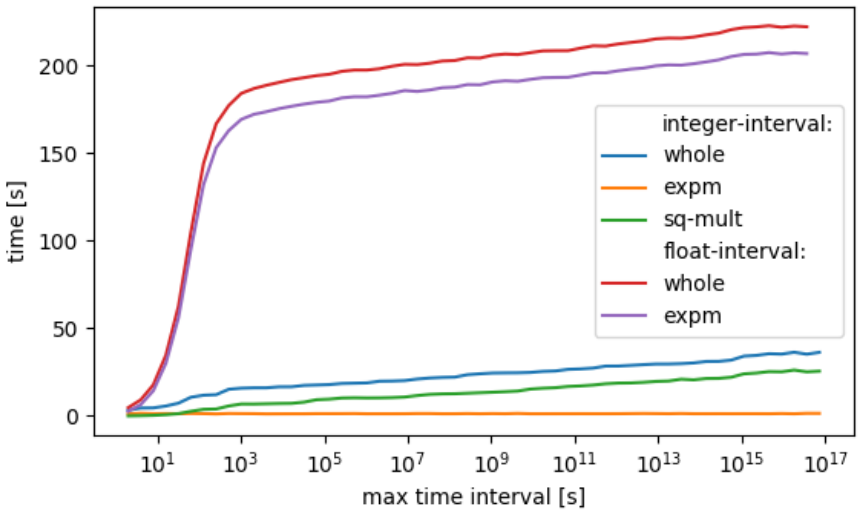


Figure 4.6: Time-complexity of two algorithm variants and their subparts

Conclusion

The steep growth at the beginning is caused by choice of integral interval lengths. For small values of $t_{\max}$ is higher probability of more same sized time intervals. That has direct impact at the complexity where variable r is directly presented. The chance of choosing two same interval length for bigger $t_{\max}$ values is extremely low.

The logarithmic grow of *square and multiply* method is obvious. But to explain similar behavior of the *expm* method we need to look deeper into

its implementation [1]. The method is using Padé approximants of different values from 3 to 13. The computation of the approximants of higher order is computationally more costly. The algorithm is choosing the smallest Padé approximant which securely not overgrow the wished threshold error. Our observation suggest that the bigger the numbers in the exponentiated matrix are, the bigger is the probability of the more complex Padé approximant being chosen.

4.3 Numerical Stability

The main advantage of integer-interval variant of algorithm is that it computes costly matrix exponential only once and derives all other exponentials of the matrix multiplications by computing its powers via *square and multiply* method. It arises the question of numerical stability of the method. Can inaccuracies acquired by *square and multiply* significantly change direction of convergence?

Description

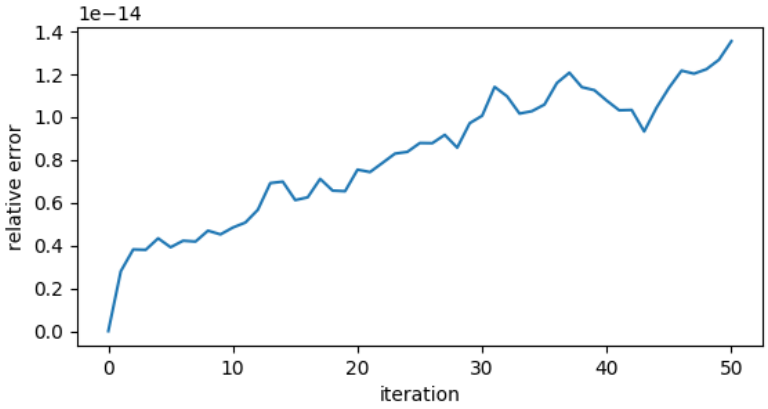
In the experiment we have run both integer-interval and float-interval algorithms with same initial parameters. As the value of interest we have measured the relative euclidean distance among both models jump-rates matrices $\mathbf{Q}$. We have used the models of 5 hidden states and 5 observable variables at the dataset of 100 observation points. Time intervals were generated by exponential distribution with parameter λ equals 0.1, 0.01 and 0.001 consecutively. The obtained plotted error is the average value of 10 runs of the experiment.

Observation

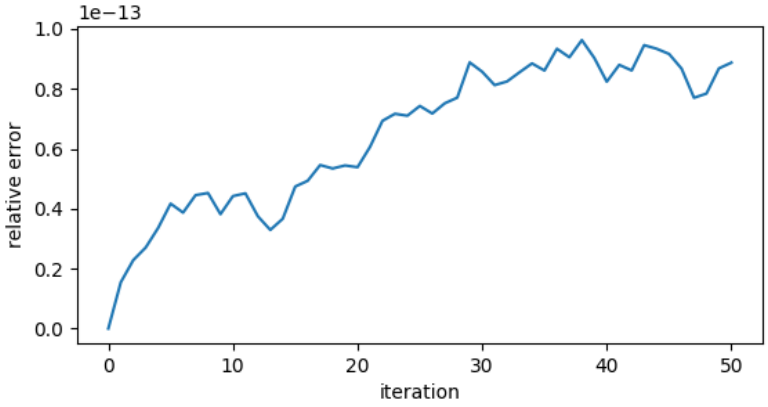
The measured relative error of jump-rate matrices seems to grow faster at the beginning and gradually slowing its pace by growing iterations. Values of the relative error are very small. However it seems that the variance magnitude grows with the decreasing magnitude of exponential parameter λ. (Figures 4.7a,4.7b,4.7c)

Conclusion

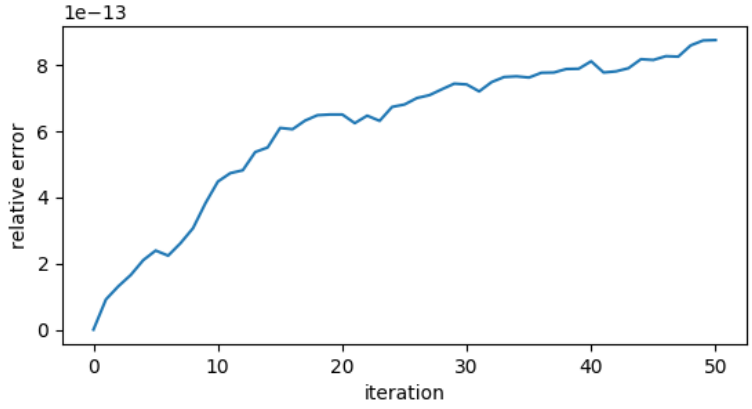
The experiment haven't shown any significant error propagation. It has been a bit more prevalent at the beginning of the convergence as during this phase are the changes in the jump-rate matrices most prominent. Later errors seems to be eliminated as the solutions converging to the local optima. Probably, under some more extreme edge conditions the matrices may diverge. But, when taking into the account the randomness of the initial configuration and assorted characteristic of the parameter space we do not see it as problem and



(a) $\lambda = 0.1$



(b) $\lambda = 0.01$



(c) $\lambda = 0.001$

Figure 4.7: Relative distances of jump-rates matrices for data generated by exponential distribution with variable parameter λ

we do not think it can, in general, negatively influence the performance of the algorithm.

4.4 Soft vs. Hard method

This experiment compare the soft and the hard method of EM algorithm described in Section 2.3.2. The methods differ in their maximization function. The soft method maximize the probability of the dataset being generated by the model, whereas the hard method maximize the probability of the most likely state sequence generating the dataset. We have examined and compared the results of both these functions for both models.

Description

There is any unified result of hard method trainings. Use of datasets generated by different models and also the various initial random parameters produce very different results. This makes the experimenting to be a complex task. In the observation we have tried to show the results uncovering most prevalent characteristics. First, we have measured the performance ratio on both training and testing dataset. We have chosen small tree hidden states model that was trained on the dataset of 100 sequences of 100 observation points as it manifests the methods characteristics sufficiently well.

The goal of the second part was to explore the possible strong feature of the hard method - maximizing the probability of most likely state sequence. We have taken the most probable state sequence of the trained model and compared its probability of being generated by both trained and original model. Performance ratio is this probability divided by probability of most probable sequence generated by the original model.

Observation

On Figure 4.8 we can see the resulting convergences of the first part of the experiment measured on the tasting dataset. Models trained by the soft method all converge relatively good. On the other side models trained by the hard method are unstable at the beginning and the final performance ratio depends heavily on the initial random generation of parameters.

Figures 4.9 show two different results of hard method convergence. While the probability of the most probable state sequence in both cases over-performed the original model, probability of the same sequence being generated by the original model is very low in the case on Figure 4.9b. The soft method behave more stable always producing intermediate results similar to the original model.

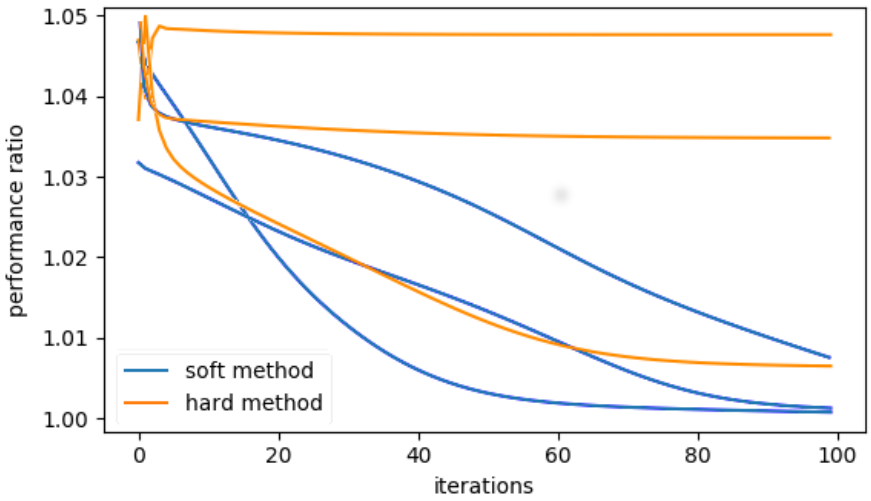
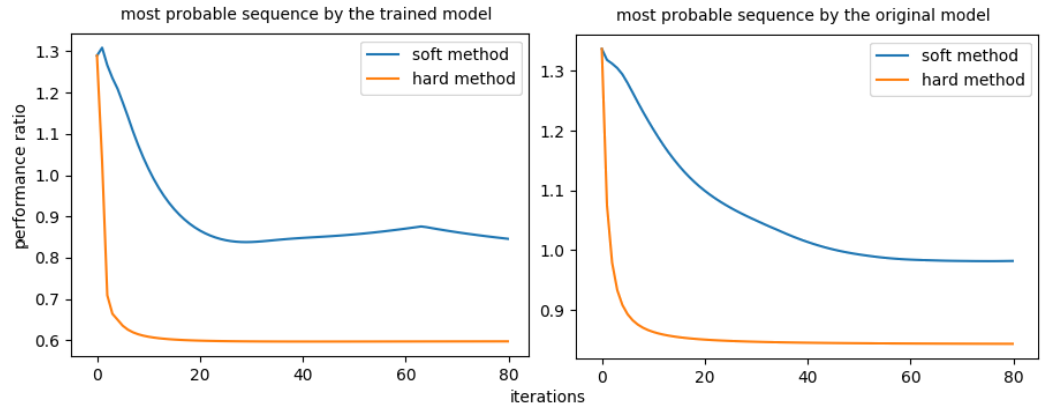


Figure 4.8: Probability of the testing dataset being generating by the models

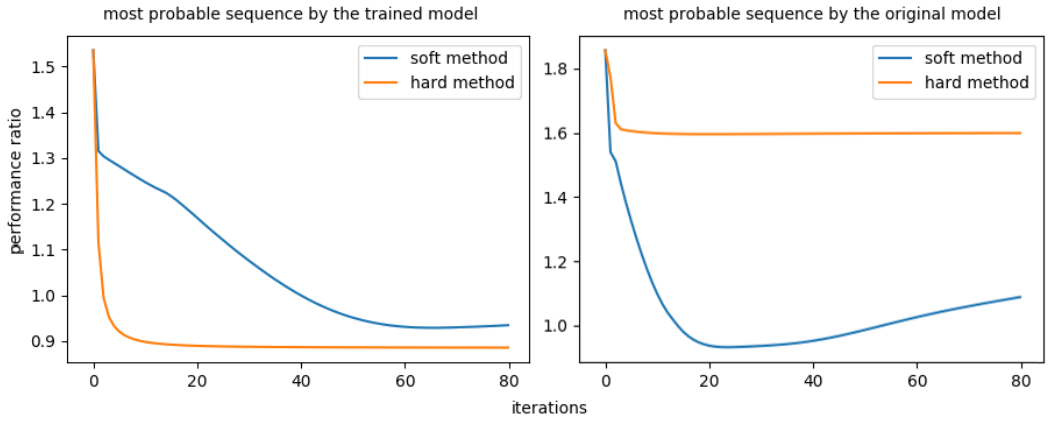
Conclusion

The hard method is maximizing the different likelihood function as described in Section 2.3.2. Its use for maximizing the probability of dataset being generated by the model is very unstable. Although sometimes it can produce satisfying results, it just appears as the side effect of its original purpose.

The hard method behaves considerably well in maximizing the probability of most probable state sequence. However, as the experiments uncovered the resulting sequence has often very low probability of being generated by the original model, thus the models parameters are different.



(a) example of the hard method successful convergence



(b) example of the hard method misleading convergence

Figure 4.9: Performance ratio of the most probable state sequence

4.5 Hidden States Number

The number of hidden states is an important parameter, influencing the characteristic of the model. The higher number means the higher plasticity of the model. It can be beneficial as the model can better fit the domain space, however sometimes it may not be useful at all and cause the over-training. The parameter also critically influence the time and memory complexity of the algorithm. In following two experiments we observe how variable number of states behave when matching the dataset generating by single model, and in the later one we try to find the limits of the algorithm regarding the number of states and examine how the increasing complexity of the parameter space influence the convergence ability.

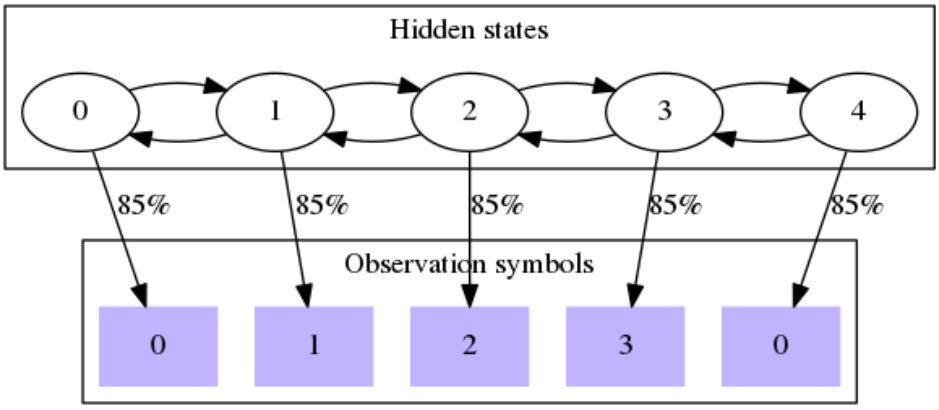


Figure 4.10: Schema of the model "birth and death"

4.5.1 Plasticity of the Models

Description

For this experiment we considered an artificial model consisting of five hidden states and four observation symbols arranged in the way of birth and death chain as in Figure 4.10. There is not allowed for the model to change states except in the way of arrows. To add the uncertainty into the dataset we have blurred the observation symbol emission by adding the 15% error to the emitted symbol. The model is deliberately built in the way so it uncovers the properties worthy to explore. However the properties are more or less visibly present in any model.

We have generated three random datasets. The *small* training dataset consisting of 15 sequences each by 15 observations and two *big* datasets of 100 sequences by 100 observations, one for training and other for testing purpose. We have used both training datasets to train models with variable number of hidden states (from 2 to 8) and marked their performance after every of one hundred iteration at the respective training and *big* testing dataset. We have repeated the experiments five times and plotted the average results.

Observation

The models, except the ones with 2 and 3 hidden states, were able to over-perform the original model at the *small* training dataset after couple of iterations 4.11a. The bigger number of hidden states makes the convergence faster and also converge to the lower value. The measure at the testing dataset shows, that the models were actually over-fitted after couple of iterations and later diverge from the actual model. 4.11b.

On the contrary the convergence lines plotting the training on the *big* data-

set 4.12a are almost the same as results on the testing data 4.12b. Models with the hidden states number higher or equal 5 converge well. After one hundred iteration they have reached performances in interval $(1.00025, 1.006)$ on the training dataset and $(1.0032, 1.01)$ on the testing dataset. Higher number of states make the convergence faster, but all of the models seems to converge to the similar value. Models with the 4 or lower number of hidden states converge visibly worse.

Conclusion

Choosing the correct number of hidden states is crucial as its under-valuating can negatively influence the performance of the algorithm. Insufficient number of the hidden states for models 2-4 does not allow them to converge to the optimum values. The waves on the convergence lines of models 2 and 3 reveal that the averaged convergence lines differ. That marks the instability, thus the model that is too weak to cover the problem space. Model 4 was probably not able to distinguish the states at the ends of the birth and death sequence. That's way it has not reached the peak performance.

We haven't proof that the over-valuated number of hidden states makes the models more vulnerable to overfitting as all models has overfitted similarly. The higher number of states than in the original model makes the model to converge in the smaller number of iterations, but not to the considerably better values. However, the cost of computationally more expensive single iteration makes their convergence slower in real time.

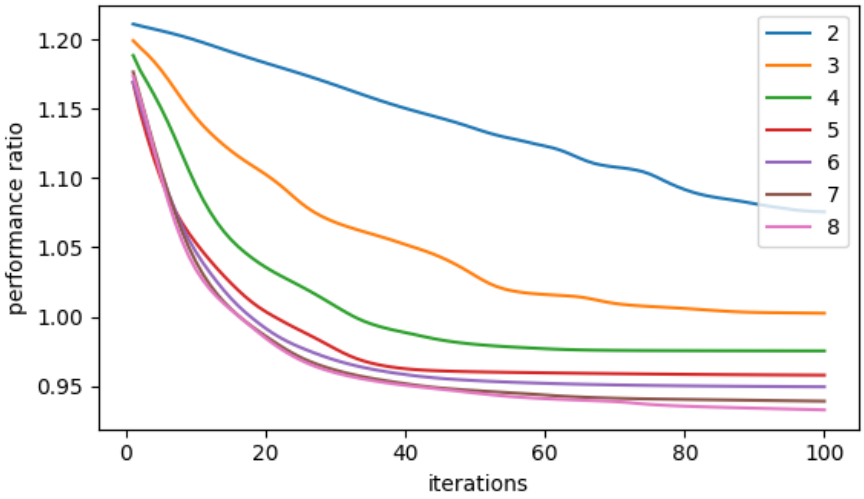
The experiment also stresses the importance of sufficiently sized dataset, as the small one emphasized the statistical error and lead to the over-fitting of the models. That is not the surprise, the weak dataset can't train powerful model.

4.5.2 High Number of Hidden States

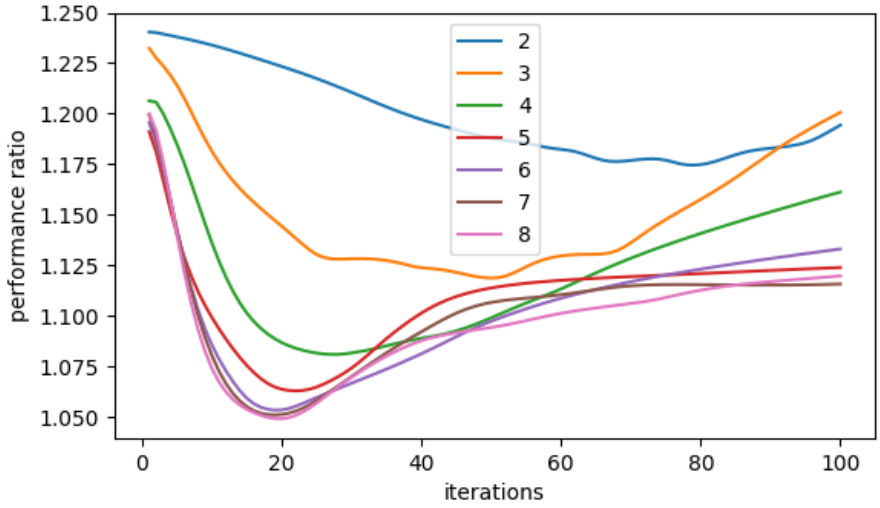
Training of models with high number of hidden states is computationally expensive. In this experiment we measure time demands of training for full and sparse jump-rate matrix as described in algorithm section 2.4. The second part of the experiment examines performance ratio and how it changes with the growing number of hidden states.

Description

All datasets used in this experiment were generated by the model with "birth and death" sequence of hidden states (similar as in Figure 4.10) always with the same number of hidden states and observation symbols as the trained models. The jump-rate matrix $\mathbf{Q}$ of such a model is three-diagonal. The actual jump rates in the nonzero positions were generated randomly.

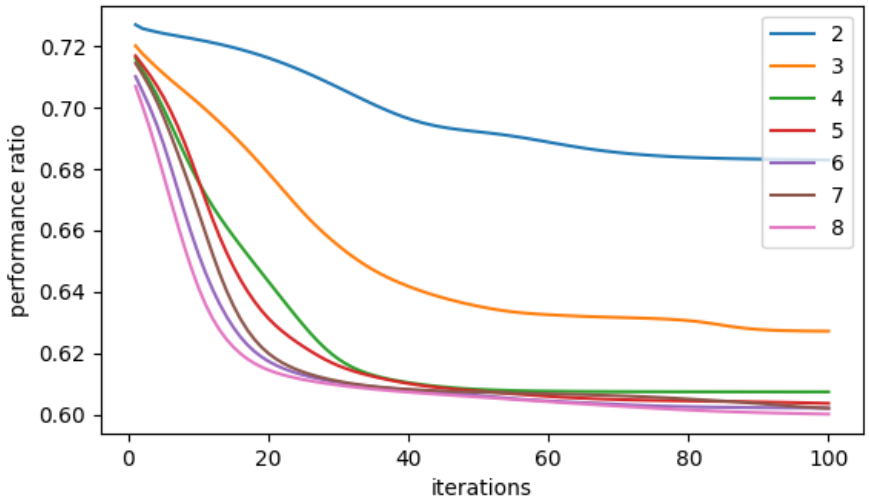


(a) Performance on the *small* training dataset

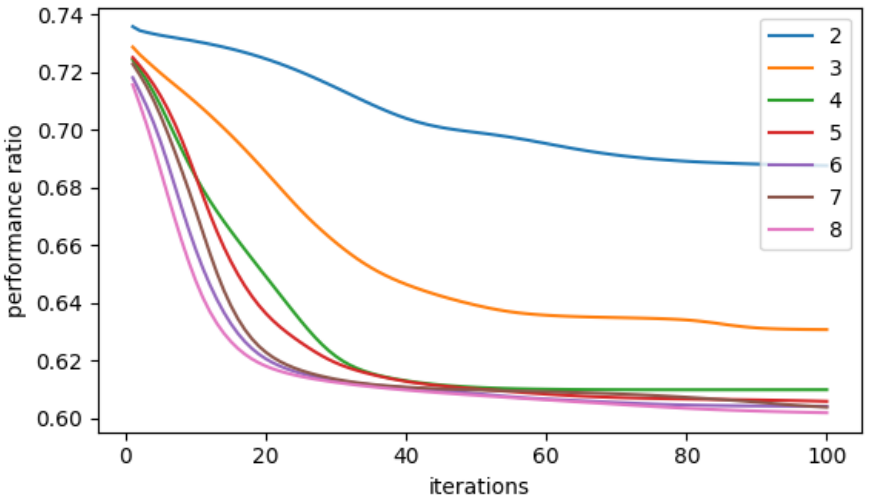


(b) Performance on the testing dataset

Figure 4.11: Performance of the models with variable hidden states number trained by the *small* dataset



(a) Performance on the *big* training dataset



(b) Performance on the testing dataset

Figure 4.12: Performance of the models with variable hidden states number trained by the *big* dataset

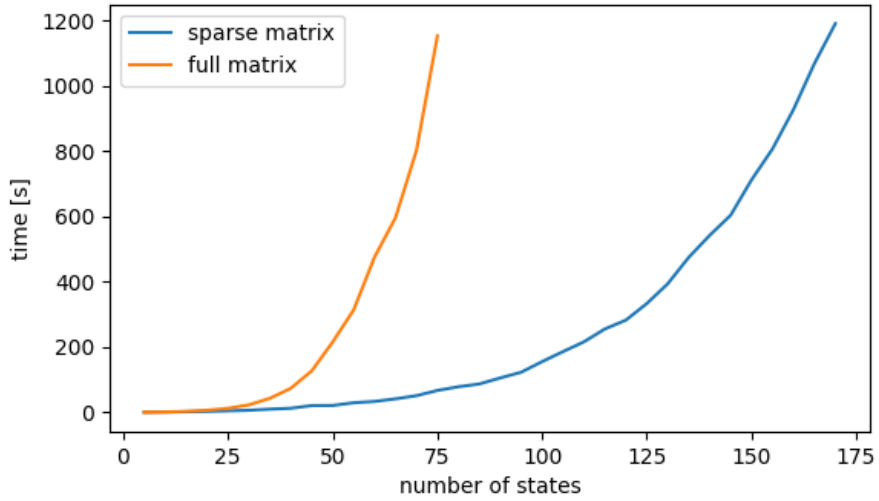


Figure 4.13: Time demands of single iteration

As the training models we have used *full* jump-rate matrix and *sparse* (three-diagonal) jump-rate matrix. All parameters were initially randomly generated. In the first part of the experiment we have measured time of single iteration for models with full and sparse matrix. For full matrix we have used models with 5 to 75 hidden states, for the sparse matrix models with 5 to 170 hidden states. We have used dataset of 1000 observation points. In the second part we have compared the convergence of models with full and sparse jump-rate matrix with variable number of hidden states. To overcome the overfitting we have used dataset of 10000 observation points.

Observation

In Figure 4.13 we can see that also the models with the high number of hidden states are able to converge in feasible time. Sparsity of the jump-rate matrix is crucial for the algorithm speed. It is possible to make three iteration in a hour by model with full matrix of 75 hidden states and the same number of iterations by model with sparse matrix of 170 hidden states.

The second part of the experiment in Figure 4.14 shows the results of the trained models on the testing dataset. The models with sparse jump-rate matrix converge significantly faster and reach the similar performance rate for all hidden states number. On the other hand the models with full jump-rate matrix are becoming a bit less plastic with growing number of hidden states.

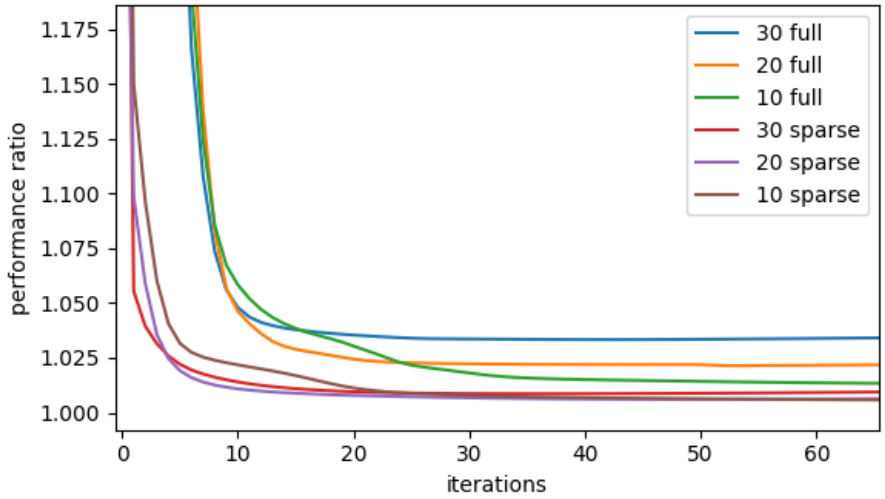


Figure 4.14: Convergences for full and sparse matrices with variable number of hidden states

Conclusion

We have shown that the implemented algorithm is fast enough to train models with high number of hidden states. The use of the sparse jump-rate matrix can significantly shrink the time-demands and also produce faster and better convergence. We recommend to use the sparse matrix always when the characteristic of the problem allows it, and if we are sure that the transition between two states can't occur.

The higher is the number of nonzero elements in the jump-rate matrix, the harder problem is need to be solved because convergence in multidimensional parameter space is more likely to fall in a local minima.

Conclusion

In this Master's thesis we have provided detailed theoretical description of discrete-time and continuous-time hidden Markov models. We have examined general characteristic of CTHMM and explained how it differs from discrete-time model. Most emphases was given to continuous-time model learning methods which various variants were described and implemented.

We have created the first of its kind general purpose continuous-time hidden Markov model library under open-source license [3]. The computationally effective and simple to use library provides broad functionality and implementation of current state-of-the-art algorithms for both discrete-time and continuous-time hidden Markov model. The most novel part are several variations of EM learning algorithm using method of matrix exponentiation for counting end-state conditioned expectations (*expm*).

The implemented algorithms are able to handle hundreds of hidden states. Fully connected model of 75 hidden states or sparse model of 170 hidden states can run one iteration of continuous-time learning algorithm in time under 20 minutes on 2.50GHz machine, when trained on the medium sized dataset.

We have implemented two variants of the *expm* algorithm different in the maximized likelihood function, so called soft and hard method. Soft method maximizes the probability of the training dataset being generated by the model, hard method uses the results of the Viterbi algorithm maximizing the most probable state sequence. We have shown the superiority of the soft method as the hard method is not directly optimizing the wished likelihood function and often produces misleading results.

The models trained on the dataset with integer lengths of time intervals can benefit from *intege-interval* variant of algorithm sparing the number of matrix exponential computations, which as we have shown is the most time demanding part of the algorithm. The experiments referring to its numer-

[3] accesible at pip `https://pypi.python.org/pypi/hmms` and GitHub `https://github.com/lopatovsky/HMMs`

ical stability demonstrated that the relative error is negligible to negatively influence the computation, whereas the time demands decreased radically.

Future Work

The effective algorithm for the continuous-time hidden Markov model parameters learning is relatively new. It has shown to be successful in biomedical field, but it certainly could be applied to solve problems of different domains. The open-source library may help to spread the method to the broader range of data scientist with different fields of interests.

The functionality of the library can be further increased. Either by looking for the new convergence methods or implementing the model extension (for example support of continuous or multiple observations).

It would be interesting to examine CT-HMM comparing to recurrent neural network as they share some common characteristics. Possibly some hybrid method could be developed.

Bibliography

[1] Al-Mohy, A. H.; Higham, N. J.: A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, volume 31, no. 3, 2009: pp. 970–989.

[2] Allahverdyan, A.; Galstyan, A.: Comparative analysis of viterbi training and maximum likelihood estimation for hmms. In *Advances in Neural Information Processing Systems*, 2011, pp. 1674–1682.

[3] Baum, L. E.; Petrie, T.; Soules, G.; etc.: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, volume 41, no. 1, 1970: pp. 164–171.

[4] Behnel, S.; Bradshaw, R.; Citro, C.; etc.: Cython: The Best of Both Worlds. *Computing in Science Engineering*, volume 13, no. 2, 2011: pp. 31 –39, ISSN 1521-9615, doi:10.1109/MCSE.2010.118.

[5] Cox, D. R.; Miller, H. D.: *The theory of stochastic processes*, volume 134. CRC Press, 1977.

[6] Dempster, A. P.; Laird, N. M.; Rubin, D. B.: Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1977: pp. 1–38.

[7] Do, C. B.; Batzoglou, S.: What is the expectation maximization algorithm? *Nature biotechnology*, volume 26, no. 8, 2008: p. 897.

[8] Grimmett, G.; Stirzaker, D.: *Probability and random processes*. Oxford university press, 2001.

[9] Higham, N. J.: The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, volume 26, no. 4, 2005: pp. 1179–1193.

[10] Higham, N. J.: *Functions of matrices: theory and computation*. SIAM, 2008.

[11] Hobolth, A.; Jensen, J. L.: Summary statistics for endpoint-conditioned continuous-time Markov chains. *Journal of applied probability*, volume 48, no. 04, 2011: pp. 911–924.

[12] Hobolth, A.; Jensen, J. L.; etc.: *Statistical inference in evolutionary models of DNA sequences via the EM algorithm*. University of Aarhus, 2005.

[13] Jackson, C. H.; etc.: Multi-state models for panel data: the msm package for R. *Journal of Statistical Software*, volume 38, no. 8, 2011: pp. 1–29.

[14] Jones, E.; Oliphant, T.; Peterson, P.; etc.: SciPy: Open source scientific tools for Python. 2001–, [Online; accessed 2017-05-03]. Available from: `http://www.scipy.org/`

[15] Karlin, S.; Taylor, H. E.: *A first course in stochastic processes, second edition*. Academic Press, 1975.

[16] Leiva-Murillo, J.; Rodrguez, A.; Baca-Garca, E.: Visualization and prediction of disease interactions with continuous-time hidden markov models. In *NIPS 2011 Workshop on Personalized Medicine*, 2011.

[17] Little, R. J.; Rubin, D. B.: *Statistical analysis with missing data*. John Wiley & Sons, 2014.

[18] Liu, Y.-Y.; Ishikawa, H.; Chen, M.; etc.: Longitudinal modeling of glaucoma progression using 2-dimensional continuous-time hidden markov model. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2013, pp. 444–451.

[19] Liu, Y.-Y.; Li, S.; Li, F.; etc.: Efficient learning of continuous-time hidden markov models for disease progression. In *Advances in neural information processing systems*, 2015, pp. 3600–3608.

[20] Metzner, P.; Horenko, I.; Schütte, C.: Generator estimation of Markov jump processes based on incomplete observations nonequidistant in time. *Physical Review E*, volume 76, no. 6, 2007: p. 066702.

[21] Moler, C.; Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, volume 45, no. 1, 2003: pp. 3–49.

[22] Nodelman, U.; Shelton, C. R.; Koller, D.: Expectation maximization and complex duration distributions for continuous time Bayesian networks. *arXiv preprint arXiv:1207.1402*, 2012.

[23] Rabiner, L. R.: A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, volume 77, no. 2, 1989: pp. 257–286.

[24] Van Loan, C.: Computing integrals involving the matrix exponential. *IEEE transactions on automatic control*, volume 23, no. 3, 1978: pp. 395–404.

[25] Wang, X.; Sontag, D.; Wang, F.: Unsupervised learning of disease progression models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 85–94.

[26] Zraiaa, M.: *Hidden Markov Models : A Continuous-Time Version of the Baum-Welch Algorithm*. Master's thesis, Imperial College London, Department of computing, United Kingdom, 2010.

Appendix A

Acronyms

CL Complete Likelihood

CT-HMM Continuous-time Hidden Markov Model

CTMP Continuous-time Markov Process

DP Dynamic Programming

DT-HMM Discrete-time Hidden Markov Model

DTMP Discrete-time Markov Process

EM Expectation-Maximization

HMM Hidden Markov Model

MLE Maximum Likelihood Estimation

Appendix B

Contents of enclosed CD

- `readme.txt` ..... brief description of the CD content
- `HMMs` ..... hidden Markov models library
  - `hmms` ..... source files
    - `cthmm.pyx` ..... continuous-time hidden Markov model class
    - `dthmm.pyx` ..... discrete-time hidden Markov model class
    - `art.py` ..... additional functions for visualization
    - `train.py` ..... additional functions for multi-training
  - `docs` ..... documentation files
  - `tests` ..... testing and experiment files
    - `experiments.py` ..... source code for experiments
  - `hmms.ipynb` ..... interactive tutorial
  - `setup.py` ..... installation script
  - `README.md` ..... description of the library
- `text` ..... text of the thesis
  - `DP_Lopatovsky_Lukas_2017.tex` ..... text of the thesis in latex
  - `DP_Lopatovsky_Lukas_2017.pdf` ..... text of the thesis in pdf
  - `img` ..... images and data files to experiments