TÉCNICO LISBOA

Time-series Forecasting using Markov Models

Filipa Robert de Oliveira Rente

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Mário Alexandre Teles de Figueiredo
Dr.ª Zita Alexandra Magalhães Marinho

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Dr.ª Zita Alexandra Magalhães Marinho
Member of the Committee: Prof.ª Cláudia Alexandra Magalhães Soares

September 2020

Para ser grande, sê inteiro

Para ser grande, sê inteiro: nada
Teu exagera ou exclui.
Sê todo em cada coisa. Põe quanto és
No mínimo que fazes.
Assim em cada lago a lua toda
Brilha, porque alta vive.

Odes de Ricardo Reis, heterónimo de Fernando Pessoa (1888–1935).

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First of all, I would like to thank the person who helped me the most during this dissertation, my co-supervisor Zita Marinho, for her kindness, great and honest insights and useful guidance during this work. Her patience in guiding me throughout the theoretical study of the subject, mathematical proofs and practical results was fundamental. I would like to extend my gratitude to my supervisor, Mário Figueiredo, for, despite his tight schedule, being always there for me, through a lot of face-to-face and virtual meetings or even by his quick response to email doubts. His great wisdom on the subject contributed to useful comments. Without the vision and help of both of them, none of this would have been possible. A special thank you to my boyfriend, Manuel Serra, for his endless support, even more through the hard times we are facing nowadays with the COVID-19 pandemic. Without you, I would have never made it this far. Last but not least, a special thank you to my friends and family who were always by my side during this journey. For the great friendships I have made during my journey throughout Técnico and all the good memories I will forever take with me, thank you. Even though Técnico and, in particular, the course, are very demanding and even if I do not remember everything that was taught to me, I will certainly never forget a lot of good moments and I will bring with me a tremendous willpower and, above all, a hard-work discipline. Thank you all for helping me becoming the person I am today.

Resumo

A modelação e previsão de séries temporais é um tópico que se tem tornado bastante popular nas últimas décadas, uma vez que pode ser aplicado a vários domínios práticos para resolver problemas complexos. Nesta dissertação, usamos modelos de Markov, tais como *hidden (semi-)Markov models* (*HMMs* e *HSMMs*) com observações geradas por processos de Poisson, e comparamo-los com redes neuronais, em particular com *LSTMs (long-short term memory)*, na tarefa de modelar e prever séries temporais.

A primeira contribuição é a formulação de um novo modelo HSMM que permite a introdução de características externas através de regressão logística. Realizamos várias experiências sintéticas onde testamos a robustez dos modelos de Markov face a erros de modelação. Testamos ainda com dados reais provenientes de redes sociais (Twitter), sobre *bitcoin*, onde avaliamos a precisão da previsão dos diferentes modelos. Os resultados indicam que os modelos Markov têm uma performance superior às redes neuronais no regime em que a quantidade de treino é menor. Outra vantagem deste tipo de modelos é que são mais fáceis de treinar e têm mais fácil interpretabilidade em comparação com redes neuronais. Esta última vantagem é muito procurada hoje em dia, no contexto de tomada de decisões automáticas com base em aprendizagem automática.

A segunda contribuição diz respeito à selecção do número óptimo de estados, assim como da duração máxima dos estados de um HSMM. Uma das principais contribuições é uma demonstração formal de equivalência entre modelos que nos permitem focar apenas na selecção do número de estados. Propomos um critério único de selecção baseado nessa demonstração. Além disso, o número óptimo de estados é encontrado através de uma estratégia de poda (*prunning*) sequencial aplicada ao critério de descrição mínima de misturas (*mixture minimum description length* – MMDL). Demonstramos a eficácia da abordagem na selecção do modelo usando experiências sintéticas e reais.

Keywords: Previsão de séries temporais; Modelos Markov; Regressão logística; Redes neuronais; Selecção da ordem do modelo.

Abstract

Time-series modeling and forecasting is a topic that has become quite popular over the last decades since it can be applied to a wide range of practical domains to solve complex problems. In this dissertation, we compare parametric stochastic Markov models, such as *hidden (semi-)Markov models* (*HMMs* and *HSMMs*), with artificial neural networks, in particular, *long-short term memory* (*LSTM*), for the task of time-series modeling and forecasting.

The first contribution regards the introduction of external features into the HSMM, through a logistic regression approach. We perform several synthetic experiments to assess the estimation accuracy of the Markov models. We also test with real social media data from Twitter, about *bitcoin*, where we evaluate the prediction accuracy of the different models. The results indicate that the Markov models can achieve a higher prediction accuracy than neural networks in the less training data regime. Moreover, these models are easier to train and less complex to interpret when compared to neural networks. The model interpretability is very sought after nowadays, in the context of automatic decision making based on machine learning.

The second contribution regards the model order selection problem in Markov models. We propose a novel approach to select the optimal number of states, as well as the state duration in HSMMs. One of the main contributions is a formal proof of equivalence between models that allows us to focus only on the selection of the number of states. We propose a unique order selection criterion based on that proof. Furthermore, the optimal number of states is found through a sequential pruning strategy using a so-called *mixture minimum description length* criterion. We demonstrate the effectiveness of the approach using synthetic and real experiments.

Keywords: Time-series forecasting; Markov models; Logistic regression; Artificial neural networks; Model order selection.

Contents

List of Tables

List of Figures

Nomenclature

α	Bernoulli parameter
$\alpha_t^*(j)$	Auxiliary forward variable
$\alpha_t(j)$	Forward variable
$\beta(j)$	Backward variable
$\triangle t$	Discretization unit (in hours, for the Twitter dataset)
$\boldsymbol{\beta}$	Logistic regression weights
$\boldsymbol{\lambda}$	Set of all k Poisson parameters
$\boldsymbol{\pi}$	Initial state distribution of the Markov chain
$\boldsymbol{\theta}$	Geometric distribution parameters
$\boldsymbol{A^{(t)}}$	Dynamic transition matrix
$\boldsymbol{B}$	Emission probability matrix
$\boldsymbol{c}_t$	Feature vector at time t
$\boldsymbol{D}$	Duration probability matrix
$\boldsymbol{H}$	Hessian matrix
$\boldsymbol{p}_\infty$	Stationary probability distribution
$\gamma(j)$	State posterior
$\hat{m}_i(n)$	Estimator of the mean sojourn time in state s_i
$\hat{v}_i(n)$	Estimator of the stationary probability distribution of the embedded Markov chain
Λ	Non-negative stochastic process to realize $\lambda(s)$ in Cox processes
λ	Rate of the arrival process; Poisson parameter; $\lambda(t)$ in time; $\lambda(s)$ in space
$\mathcal{D}$	Set of possible state durations
$\mathcal{E}_t(j)$	Probability of ending state s_j at time t in EDHMMs

$\mathcal{F}_t(i)$ Probability of entering state s_i at time $t+1$ in EDHMMs

$\mathcal{Q}$ Set of proposal distributions (Variational Bayes)

$\mathcal{S}$ State set of a Markov chain with k states, $\mathcal{S} = \{s_1, ..., s_k\}$

$\mathcal{V}$ Set of m observable values

μ Set of parameters

$\rho(\cdot)$ Squashing function

τ_t Residual time in state S_t

φ L-BFGS regularization parameter

$\xi_{\cdot}$ Lagrangian multipliers

$\zeta_t(i,j)$ Transition posterior $s_i \rightarrow s_j$ at time t

$\{f_i(\cdot)\}_{i=1}^k$ Set of k latent functions for the MMGCP model

$b_i^*(o_t)$ Filtered emission probability

$C(\cdot)$ LSTM cell state

c_j Probability of the j^{th} component in a mixture of distributions

C_k Value of the model order criterion

d_{il} Duration of the l^{th} visit of the embedded Markov chain to state s_i

d_{max} Maximum allowed state duration

$E(\cdot)$ Error function

E_i Random variable that represents the i^{th} arrival epoch, $E_{t_i} \equiv E_i$

$F(\cdot)$ LSTM forget gate

$f_{X_i}(x)$ Probability mass function of X_i

$G(\cdot)$ LSTM output gate

$g(\cdot)$ Divergence function between distributions

$I(\cdot)$ LSTM input gate

k Number of states of the Markov chain

$k^i(\cdot,\cdot)$ Variance of the Gaussian process prior of state s_i in the MMGCP model

k_{max} (k_{min}) Maximum (minimum) total number of states

m Number of observable values

$m^i(\cdot)$ Mean of the Gaussian process prior of state s_i in the MMGCP model

n Length of the state and observation sequences

$N(n)$ Number of jumps of the embedded Markov chain

$N(t)$ Temporal counting process of the arrivals

$N_i(n)$ Number of visits of the embedded Markov chain to state s_i

N_k Number of free model parameters

n_{0i} Number of times s_i is the first state in the state sequence S

n_{ij} Number of state transitions $s_i \rightarrow s_j$

n_{jv} Number of times observation takes value v in state s_j

O_t Random variable that represents the observation at time t

p Number of features

p_1 Probability of splitting Poisson process 1

p_2 Probability of splitting Poisson process 2

$p_i(d) \equiv D_{id}$ Probability of duration d in state s_i

$q(\cdot)$ Proposal distribution (Variational Bayes)

R Bounded region in space

s_i State instantiation

S_t Random variable that represents the state of the Markov chain at time t

t_i i^{th} time instant

$v_t(j)$ Viterbi variable

W, b LSTM weights and bias

w_{t_1} (w_{t_2}) Probability that sample o_t was generated by the second Poisson process

X_i Random variable that represents the i^{th} interarrival interval

y class label/indicator variable for logistic regression and LSTMs

Z Bernoulli random variable

D Duration random variable

Chapter 1

Introduction

1.1 Motivation

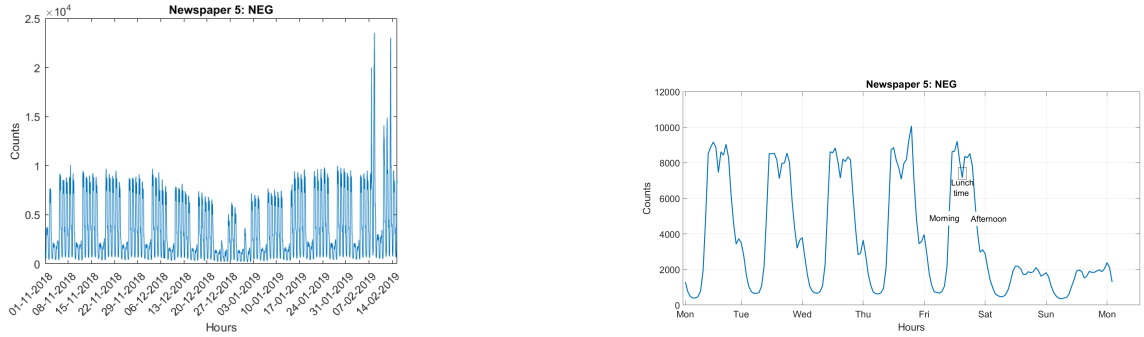
Many real phenomena produce data that can be represented as a *temporal point pattern*, i.e., a list of the time instants of occurring events. Earthquakes, road accidents, child births, and cancer diagnosis are some examples of such data. Common to these examples is that the number and exact time instant of future events is not known. Usually complex mechanisms are behind these seemingly random times, for example cancer diagnosis can cause new cancer diagnosis in the form of metastases.
Stochastic predictive models, also called *time-series prediction models* when applied to temporal point patterns, allow us to predict where in the future the next events are going to happen with a certain level of confidence. For example, in the business domain, *customer churn*, which refers to the loss of customers, has been widely studied in the past decade. It can greatly benefit from stochastic predictive models to identify, from behavioral patterns, potential clients at risk of churn with a certain probability [1]. Other examples of areas where time-series forecasting has been applied are medicine [2], finance [3] [4], meteorology [5] [6], retail [7], traffic [8], power engineering [9], and computer vision [10], to name a few.

1.2 Topic Overview

Building mathematical models of temporal point patterns is not trivial, because some models assume certain characteristics about the data that may not be verified, e.g., independence between events. There are cases (e.g. cancer metastases) where data inter-dependencies cannot be ignored. Furthermore, the time-series prediction model should be able to identify trends and seasonality patterns in the data and properly reflect those patterns in the predictions. As an example of such patterns, we present a dataset collected within the scope of this dissertation, in association with *Priberam*[1] and *Cofina*[2].
It consists of a time-series dataset that contains the number of hourly user accesses to the homepages

[1] *Priberam* is an international company that supplies natural language processing and machine-learning technologies to an extended portfolio of clients, including Amazon, Kobo, Microsoft, and leading media publishers in Portugal, Brazil and Spain.
[2] *Cofina* is one of the leading portuguese media conglomerates, founded in 1995.

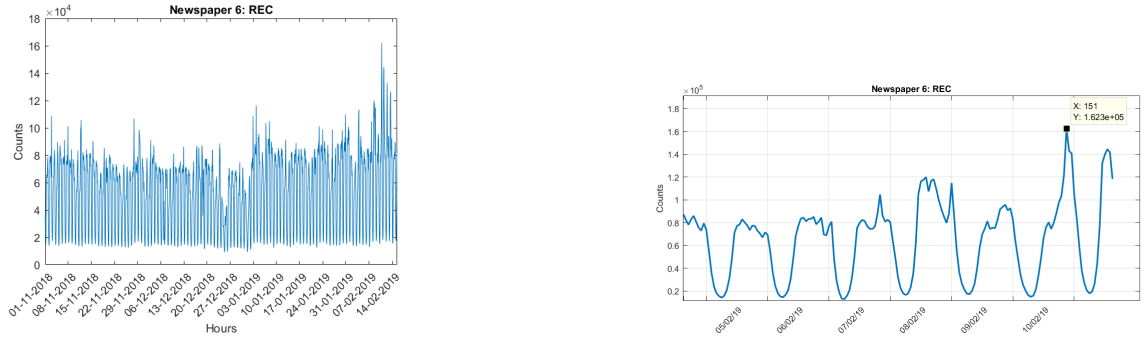
of Cofina newspapers' website, during the period from 1/11/2018 to 15/02/2019. Figure 1.1(a) shows the post-processed dataset of the *Jornal de Negócios* newspaper, from where interesting conclusions can be drawn. User access is quite regular, except during Christmas, where the maximum number of counts



(a) (b)

Figure 1.1: Number of counts of user access to the *Jornal de Negócios* newspaper webpage (a) during a period of three and a half months from 1/11/2018 to 15/02/2019 and (b) during the week of 05/11/2018.

reduces, and in February where some counts are surprisingly high. One can easily observe groups of 7 patterns that correspond to weeks, as shown in Fig.1.1(b). The higher peaks are workdays, whereas the lower peaks occur at weekends. The slope within one day is also very representative. However, this regular behavior is not present in all newspapers. Fig.1.2(a) shows the aperiodic and irregular nature



(a) (b)

Figure 1.2: Number of counts of user access to the *Record* newspaper webpage (a) during a period of three and a half months from 1/11/2018 to 15/02/2019 and (b) during the week of 05/02/2019.

of the *Record* newspaper, which presents sports news, mainly regarding portuguese football events[3]. A huge peak (zoomed-in in Fig.1.2(b)) is visible during the week of 05/02/2019. The peak, which corresponds to over 160 000 counts of user accesses in one hour, occurred in 10/02/2019 at around 21h, during the game played between *Benfica* and *Nacional da Madeira* teams, where *Benfica* scored an incredible 10-0 result. [4]

In this dissertation, we aim to study time-series and address their limitations in the most efficient way. To do so, we develop time-series predictive models that can properly capture trend and seasonality patterns and, together with implemented prediction algorithms, are applied to synthetic and real datasets. We compare the results of traditional models and neural networks in the task of time-series forecasting.

[3]The aperiodicity is due to the dependency of the counting data on external factors such as the date of football matches.
[4]The minimum value, which corresponds to night counts, is surprisingly stable at around 20 000 homepage accesses per hour.

Chapter 2

Methodological background

In this chapter, we begin by reviewing point processes, in particular, Poisson point processes. Then, we explore time-series modeling by presenting parametric models and their estimation procedure.

2.1 Temporal point processes

When dealing with problems with events that represent arrivals or departures spread over time, to predict future events, it is necessary to first model the events' data patterns in some way. The most common approach is to structure this type of data as a stochastic process, called a *temporal point process*. We can generalize these temporal point processes to any dimension, and the resulting processes are simply named *point (or arrival) processes* [11]. They are used in a variety of fields, such as neuroscience [12], telecommunications [13] and image deblurring [14]. For instance, in neuroscience [12], neural spike data collected from multiple electrodes can be modeled as point processes, providing statistical tools that measure important patterns to build a neurophysiological structure of the human brain.

A temporal point process is a stochastic arrival process used to model the time instants at which random arrivals or departures from a system occur.

> **Example 1** As an example of an arrival process, consider the e-mails received on a mailing box. E-mail arrivals may occur at any $t > 0$[1], thus, the arrivals are random variables.

A temporal arrival process can be defined in three different ways using: **1)** A sequence of random variables E_i, called arrival epochs, representing the exact time of the i^{th} arrival; **2)** A sequence of random variables X_i, designated interarrival intervals, which represent the time interval between the $(i-1)^{th}$ and the i^{th} arrivals; or **3)** $N(t)$, denominated the counting process, which specifies how many arrivals have occurred from the start up to time t.
Fig. 2.1 shows how these variables relate to each other. A temporal arrival process is fully specified with the joint distribution (for n arrival times) of a sequence of any of these three random variables.
An interesting class of temporal arrival processes is the Poisson point process, which is used to model

[1] Where we defined $t = 0$ as the start time of the process.

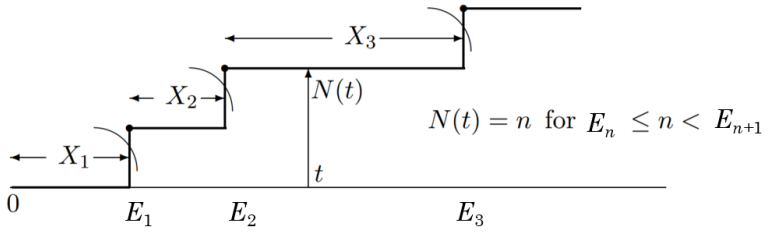


Figure 2.1: Structure of an arrival process, where the transitions correspond to event arrivals. Arrival epochs are denoted by E_i, interarrival intervals by X_i, and its counting process as $\{N(t); t > 0\}$ [15].

a wide variety of real event data. Examples of application fields of Poisson point processes include economics [16], earthquake analysis [17], biology [18] and industry, for modeling customer churn [19].

2.1.1 Poisson point processes

A Poisson process is a type of an arrival process that is commonly used for its simplicity. In a Poisson process, the interarrival times X_i are iid[2] random variables that follow an exponential distribution with pdf[3], illustrated in Fig. 2.2(a), given by $f_{X_i}(x_i)$ in eq. (2.1). The parameter $\lambda > 0$ in eq. (2.1) denotes the *rate* (or *intensity function*) of the arrival process, which corresponds to the expected number of arrivals in some bounded region per some unit of extent such as length, area, volume, or time. The rate is the only parameter of a Poisson process $X \sim \mathcal{P}(\lambda)$[4], and also its mean and variance $\mathbb{E}(X) = var(X) = \lambda$.

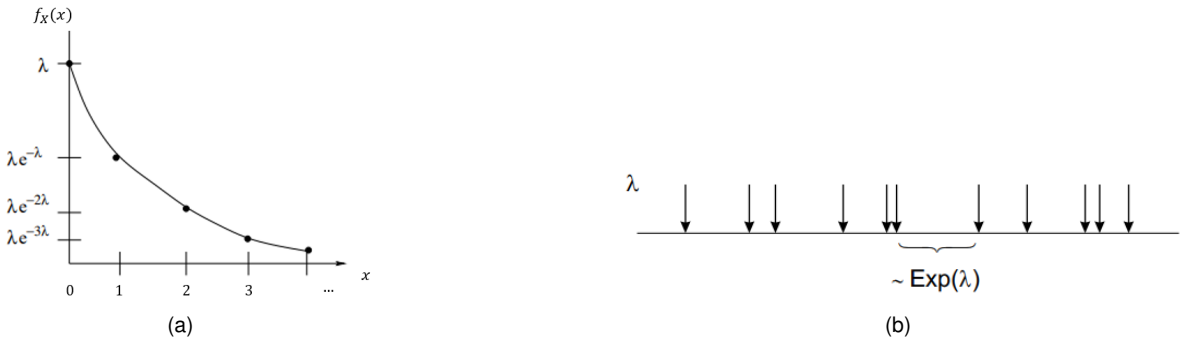


Figure 2.2: (a): Exponential pdf of the interarrival intervals in a Poisson process [20]; (b): Sample of a Poisson process where the exponential distribution of the interarrival times is visible [21].

> **Example 2** If we model example 1 as a Poisson process, the rate is the average number of incoming e-mails within a certain time interval. For example, $\lambda = 5$ e-mails per hour.

Poisson processes are indeed a powerful tool to model events over time because of their simplicity, which comes from the following properties:

[2]Independent and identically distributed.
[3]Probability density function.
[4]Where the notation $\mathcal{P}$ represents a Poisson distribution and $X \sim \mathcal{P}(\lambda)$ indicates that the random variable X follows a Poisson distribution with parameter λ.

***Property* 1: Independence.**

$$f_{(X_1,...,X_n)}(x_1,...,x_n) = \prod_{i=1}^{n} f_{X_i}(x_i) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}, \qquad x \geq 0. \tag{2.1}$$

The factorization in eq. (2.1) is due to the fact that the interarrival intervals are independent. However, there are situations in which this property is a limitation considering that some problems require a dependency between event arrivals.

***Property* 2: Constant rate** $\lambda(t) = \lambda$.

In any two given intervals of the same length, the same number of arrivals occur. Poisson processes that follow this property are designated *homogeneous*.

***Property* 3: Memoryless.**

The exponential distribution of interarrival intervals induces a *memoryless* behavior [15]

$$\mathbb{P}[X_i > t + \epsilon | \quad X_i > t] = \mathbb{P}(X_i > \epsilon), \qquad \forall \epsilon > 0, \quad \forall t \geq 0\,. \tag{2.2}$$

The probability equivalence in eq. (2.2) indicates that the memory of previous interarrival intervals is discarded.

***Property* 4: Stationary and independent increments.**

A Poisson process has stationary increments if the probability distribution of the arrivals does not change when shifted in time. The counting process $N(t_2) - N(t_1)$, for any $t_2 > t_1 \geq 0$, has the same distribution as $N(t_2 - t_1)$. The increments are independent if the counting process in one interval is independent of the counting process in any other disjoint interval.

***Property* 5: Merging and splitting.**

Merging (Fig. 2.3(a)) consists of combining independent Poisson processes. The merged process is also a Poisson process parameterized by the sum of the individual rates. Splitting (Fig. 2.3(b)) consists of subdividing Poisson processes by "re-directing" arrivals through a probabilistic choosing mechanism.[5] Each arrival is redirected to the first process with probability $p_1 = \lambda_1/(\lambda_1 + \lambda_2)$, and to the second with probability $p_2 = 1 - p_1$.

Despite the limitations of having constant rate and no memory, these processes are still one of the simplest and preferred processes for statistical time-series modeling.

As long as property 4 is verified, the Poisson process can be defined by the PMF[6,7] of the counting process $N(t)$ with constant rate λ. That is, [15]

$$\mathbb{P}(N(t) = n) = \frac{(\lambda t)^n e^{-\lambda t}}{n!}, \quad n \in \mathbb{N}^0. \tag{2.3}$$

The merging property suggests that more flexible Poisson processes can be built other than single

[5] Note that both processes are completely independent of each other, despite the dependence of the choosing mechanism.
[6] Probability Mass Function, the counterpart of the pdf for discrete distributions.
[7] Because the number of arrivals in the interval $[0, t]$ is a discrete variable.

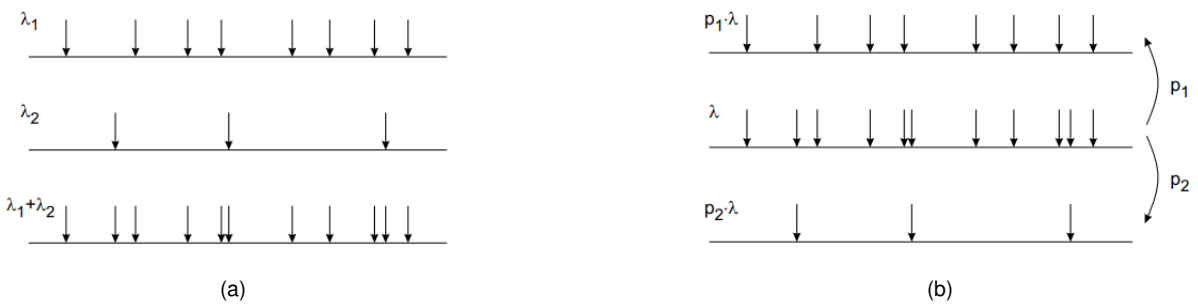


Figure 2.3: (a): Sample of a combination of two Poisson processes; (b): Sample of a subdivision of one Poisson process into two Poisson processes [21].

constant rate homogeneous Poisson processes. Nevertheless, the specification of the intensity function as the sum of different intensity functions is limiting for most real-life problems, which are possibly better modeled by a time-varying arrival rate $\lambda(t)$. This sets the distinction between homogeneous and *non-homogeneous Poisson processes*. Non-homogeneous Poisson processes, detailed in the following section, belong to the class of non-homogeneous point processes, used to model varying intensity functions. By definition, general non-homogeneous point processes have independent but not stationary increments. Moreover, the time intervals between points (X_i) are no longer iid.

2.1.2 Non-homogeneous Poisson processes

In a non-homogeneous Poisson process, the number of points in any interval, $N(t)$, is modulated by a Poisson distribution. Due to the non stationary increments, the arrival rate is not constant but a function of time ($\lambda \rightarrow \lambda(t)$), as illustrated in Fig. 2.4 (Left). The number of arrivals in any interval of length T is

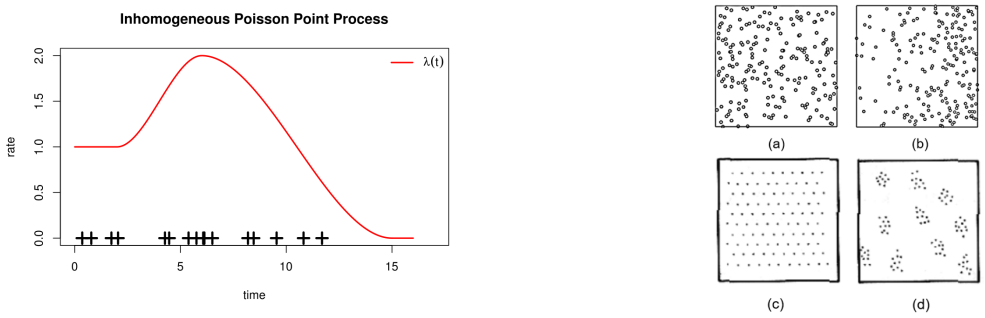


Figure 2.4: (Left): Temporal evolution of a non-homogeneous Poisson process. Events are marked as black crosses and $\lambda(t)$ is the red curve [source: Wikipedia]; (Right): (a) Homogeneous spatial Poisson process with $\lambda = 200$ points$/m^2$ in a $1m^2$ square; (b): Non-homogeneous spatial Poisson process with linearly varying intensity function $\lambda(s) = 350s + 25$ [22]; (c) Binomial spread and (d) Negative binomial clustered processes to model the distribution of e-mail arrivals in a $10 \times 10km^2$ area [23].

obtained by replacing the term λt in eq. (2.3), that reflects the expected amount of arrivals in $[0, t]$, with the integral of λ over T. That is,

$$\mathbb{P}[N([t, t+T)) = n] \quad = \frac{(\int_t^{t+T} \lambda(\tau) d\tau)^n e^{-\int_t^{t+T} \lambda(\tau) d\tau}}{n!}, \quad n \in \mathbb{N}^0. \tag{2.4}$$

Another caveat of temporal Poisson processes is their limitation to a one-dimensional index set (time), which is unsuitable when the events require a model that captures point interactions in space. *Spatial point processes*, introduced in the next section, counter this constraint.

2.1.3 Spatial point processes and Cox processes

The intensity function in a spatial point process (SPP) is defined as $\lambda(s) \geq 0$, where $s \in \mathbb{R}^n$ represents a point in the cartesian (or feature) space.[8,9]

One of the first application of SPPs is the modeling of tree locations in the forestry domain [24], which can be used for tree pattern identification/clustering, and important metrics can be estimated such as tree density and expected total number of trees [24]. They were also used in geographical epidemiology [25], to link suspected point sources of pollution to the frequency of a certain disease within a given area, and in the medical field, to cluster microcalcifications in mammogram images [26].

Spatial Poisson processes are a subclass of SPPs in which the counting process in any bounded region R in space follows a Poisson distribution. They take advantage of the simplicity of the Poisson process in high-dimensional spaces.[10] There are homogeneous (Fig. 2.4(a)) and non-homogeneous (Fig. 2.4(b)) spatial Poisson processes. The recent study in [27] uses non-homogeneous spatial Poisson processes to model the spatial-temporal travel patterns for traffic management. In Fig. 2.4(c)(d), we observe spreading (clustering) phenomena[11], which are not well modeled by spatial Poisson processes, instead, a binomial (negative binomial) PMF is used to shape this behavior.

Cox processes are yet another type of spatial point processes where the unobserved intensity function $\lambda(s)$ is a realization of a non-negative stochastic process Λ, given by a random field. The intensity function measures the expected value of the random field for all points in the bounded region R.[12] Cox processes are successfully used in many fields, for example, in finance, for credit risk modeling [28].

2.2 Time-series modeling

A time-series is a series of data points, indexed in time order, that can be either discrete or continuous. Some examples of time-series are: annual population data, daily closing stock prices, client purchases in a store, etc. Time-series modeling is a dynamic research area that has gained a lot of attention over the last few decades. Its aim is to study time-series past observations in order to develop an adequate model that describes the inherent structure of the series. This model can be later used to generate future values for the series, i.e., to make forecasts. Thus, time-series forecasting can be seen as the act of predicting the future by understanding the past. It has a broad range of applications in numerous fields such as business [29], economics [30], finance [31], science and engineering [32] and health

[8] Spatial point processes can be used as long as the point locations are independent.
[9] The expected number of points in a bounded region R in space is the integral of the intensity function over that region.
[10] They can also be defined over both time and space, resulting in *time-space Poisson processes*.
[11] Spreading (clustering) occurs when points decrease (increase) the probability of other points existing nearby.
[12] The level curves of the random field have higher values in regions where arrivals are more frequent, and vice-versa.

[33]. For instance, in customer churn prediction [34], the time-series is a collection of the time instants of client "arrivals" with respect to the intensity function. Under this light, the underlying intensity function is modeled with the goal of predicting customer churn by extracting information from it.
The underlying intensity function of any time-series data can be modeled either by non-parametric or parametric models. Non-parametric models, contrary to parametric models, do not assume a parameterized family[13] for the probability distribution of the underlying intensity function. For the more curious readers, we present an overview about non-parametric models, in particular, non-parametric Bayesian models and their estimation procedures in Appendix E. As an example of a non-parametric Bayesian model, we briefly explain the GPCox model (see Section E.1.3) proposed by Gunter et al. in 2014 [35], which combines Gaussian processes (in Section E.1.2) with Cox processes (in Section 2.1.3). The proposed model is highly accurate for modeling time-series data with smooth variations in time.
However, in the context of this dissertation, we focus on parametric models, detailed in the next section.

2.2.1 Parametric models

In time-series modeling, the underlying intensity function of the temporal process can be modeled parametrically through the use of parametric models, which assume a parameterized family for the intensity function. We will consider the Poisson family, detailed in Section 2.1.1, for its simplicity and useful properties. The advantage of parametric models is that the estimation process[14] is tractable, in general, and faster compared to non-parametric models. Moreover, in cases where the data verifies the parametric assumption made, these models are more powerful than non-parametric models. On the other hand, they make a lot of assumptions about the data, preventing them to be applied to more contexts. In this section, we will gradually cover *Markov modulated processes* (MMPs), which are parametric models adequate for sequential data. Before introducing MMPs, a brief explanation on *Markov chains* is given.

Markov chains There are many situations in which we would like to represent real data as sequences. For example, in computational biology, such sequences include genes, proteins of a given family and DNA [36]. Models based on *Markov chains* (MC) are adequate for sequential data and are widely used in the computational biology field and on many other fields, such as finance [37], telecommunications [38], robotics [39] and health [40].

> **Example 3** Consider the problem of monitoring if a child is ready for daycare as a stochastic variable with two possible outcomes: **ill**: not ready for daycare; **ok**: ready for daycare. Consecutive recordings of the health state of a child are an example of a sample of a discrete-time stochastic process which can be applied to a Markov chain. The state space of the discrete Markov chain consists of two states: **ill** and **ok** [41].

A collection of n random variables $S_{1:n} = (S_1, ..., S_n)$, where S_t denotes the state at time t (e.g. $S_t =$**ill**/**ok** in Example 3), is a Markov chain of k states if it follows a *Markov process of first order*. That is, a process

[13] Family of distributions with parameters. For example, a Gaussian distribution has parameters: its mean and covariance.
[14] The estimation process is used to obtain an estimate for the model parameters from the available data.

where the effect of variables' past values into the future is summarized only by the current value S_t:

$$\mathbb{P}[S_t|S_{t-1},...,S_1] = \mathbb{P}[S_t|S_{t-1}], \qquad \forall t \in [n]. \tag{2.5}$$

The *Markov property* in eq. (2.5) indicates that to predict S_t we only need the last random variable, S_{t-1}. The joint probability distribution of all n random variables is simplified, using eq. (2.5), yielding

$$\mathbb{P}(S_1,...,S_n) = \mathbb{P}(S_1)\prod_{t=2}^{n}\mathbb{P}[S_t|S_{t-1},...,S_1] = \mathbb{P}(S_1)\prod_{t=2}^{n}\mathbb{P}[S_t|S_{t-1}]. \tag{2.6}$$

A Markov chain (MC) exploits the Markov property from eq. (2.5) to model the probability that a system has of changing from one state to another. The probability of going from state s_i to state s_j at time t, in a MC, is defined as

$$\mathbb{P}[S_t = s_j|S_{t-1} = s_i] \equiv \boldsymbol{A}(S_t = s_j, S_{t-1} = s_i) = \boldsymbol{A}(i,j) = A_{ij} \geq 0, \qquad \sum_{j=1}^{k} A_{ij} = 1, \qquad \forall s_i, s_j \in \mathcal{S}, \tag{2.7}$$

where $\mathcal{S} = \{s_1,...,s_k\}$ is the state space, i.e., the set of possible values for the random variables $\{S_t\}_{t=1}^{n} \equiv S_{1:n}$. For instance, in Example 3, the state space is $\mathcal{S} = \{s_1, s_2\} = \{\textbf{ill}, \textbf{ok}\}$.
Hence, a Markov chain is fully characterized by the set of parameters $\mu = \{\boldsymbol{\pi}, \boldsymbol{A}\}$, where $\boldsymbol{\pi} = \mathbb{P}(S_1) \in \mathbb{R}^k$ is a column vector containing the probabilities of the initial state in the state sequence, and $\boldsymbol{A} \in \mathbb{R}^{k\times k}$ is a squared stochastic matrix that represents the transition probabilities. In Fig. 2.5, a discrete Markov chain with three states ($k = 3$) is illustrated, with the corresponding transitions.
A discrete Markov chain with k states is sampled by following these guidelines:

1. Sample the first state $S_1 \sim \boldsymbol{\pi}$;
2. Sample the next state $S_t \sim A_{(i,\cdot)}, \forall t \in \{2,...,n\}$, given that the previous state was $S_{t-1} = s_i$.

By definition, a Markov chain is a discrete-time Markov chain. However, there are also *continuous-time Markov chains* (CTMC) in which changes to the system can happen at any time along a continuous interval. Note that $\boldsymbol{A}$ is only a matrix for discrete Markov chains. For a CTMC, the transition A_{ij} is

$$A_{ij} = \lim_{\Delta t \to 0} \frac{\mathbb{P}[S_t = s_j|S_{t-\Delta t} = s_i]}{\Delta t}, \qquad A_{ii} = -\sum_{j\neq i} A_{ij}. \tag{2.8}$$

We are now able to define *Markov modulated processes* (MMPs) for time-series modeling.
MMPs use Markov chains to define the intensity function of the time-series as a random sample (trajectory) of the chain: $(S_1,...,S_n)$. In the following section, we address a combination of MMPs with Poisson processes - called a *Markov modulated Poisson process*.

2.2.1.1 Markov modulated Poisson process

A *Markov modulated Poisson process* (MMPP) [42] models the intensity function $\lambda(t)$ as a piece-wise constant function, with a finite set of intensity levels, through a Markov chain with k states. Each state

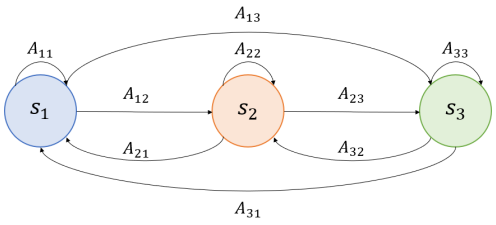


Figure 2.5: Example of a Markov chain with 3 states. The circles are the states and the arrows represent transitions between states, with their associated probability A_{ij}.

$s_i \in \mathcal{S}$ is associated with a homogeneous Poisson process of constant intensity $\lambda_{s_i} \equiv \lambda_i \geq 0$.
The intensity level of the MMPP at time t is given by the parameter λ_i of the Poisson process that is active, according to the state s_i of the Markov chain, at time t, as shown in Fig. 2.6(a).
Essentially, a MMPP is a Cox process with intensity function equal to $\lambda(t) = \lambda_i$ [42]. The Markov chain is completely specified by the set of parameters $\mu = \{\boldsymbol{\pi}, \boldsymbol{A}\}$ whereas the MMPP by $\mu = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$.
MMPPs are good models for stationary data in which the data is well approximated by a piece-wise constant intensity function with highly different intensity levels.[15] For example, if a shop that only sells carnival costumes decides to model their clients' arrivals, the data will exhibit bursty events during February and March and rare events during the rest of the year. These abrupt intensity changes are well captured by discrete-time MMPPs due to the transitions between states in the chain (e.g. states can be Carnival and Non-Carnival). On the other hand, the specification of $\lambda(t)$ as a piece-wise constant function can be a limitation for problems which require smooth transitions [43] [35].
Another interesting and novel approach to model the intensity function of time-series data is the *Markov modulated Gaussian Cox process*, proposed in 2018 by Minyoung Kim [43].

2.2.1.2 Markov modulated Gaussian Cox processes

A *Markox modulated Gaussian Cox process* (MMGCP) is an innovative model that combines the advantages of MMPPs for stationary data, introduced in Section 2.2.1.1, with the benefits of GPCox models for non-stationary data, detailed in Section E.1.3.
In [43], the authors use a CTMC with k states and a set of k latent functions $\{f^i(\cdot)\}_{i=1}^k$.
Each state s_i has a latent function that represents different characteristics of the intensity function at that state. For smoothness, a Gaussian process prior is placed in each latent function. That is, $f^i(\cdot) \sim \mathcal{GP}\big(m^i(\cdot), k^i(\cdot,\cdot)\big)$, where $\mathcal{GP}$ represents a Gaussian process with mean and covariance functions $m^i(\cdot)$ and $k^i(\cdot,\cdot)$, respectively. The intensity function is thereby given by a Cox process, i.e., $\lambda_i(t) = \rho(f^i(\cdot))$, where $\rho(\cdot)$ is a squashing function.[16] The CTMC decides which intensity is active at each time t, i.e., $\lambda_{S_t=s_i}(t) \equiv \lambda_i(t)$.
Contrary to MMPPs, that have a constant intensity level within each state, in MMGCPs the intensity within each state varies in time according to the Gaussian process, as shown in Fig. 2.6(c). The obser-

[15] Note that the total number of states k considered regulates the granularity of the model. A MMPP with a very high number of states can perfectly model the intensity function of the training data and, in the limit (as the number of states tends to infinity), the piece-wise constant model becomes smooth. However, this usually indicates that the model overfitted the training data, which means that it behaves poorly in generalizing for unseen test data.
[16] In [43], the authors adopted the squared link function for the squashing function $\rho(\cdot)$, as in [44].

vations at each time t are Poisson with parameter $\lambda(t)$. The MMGCP model performs outstandingly in

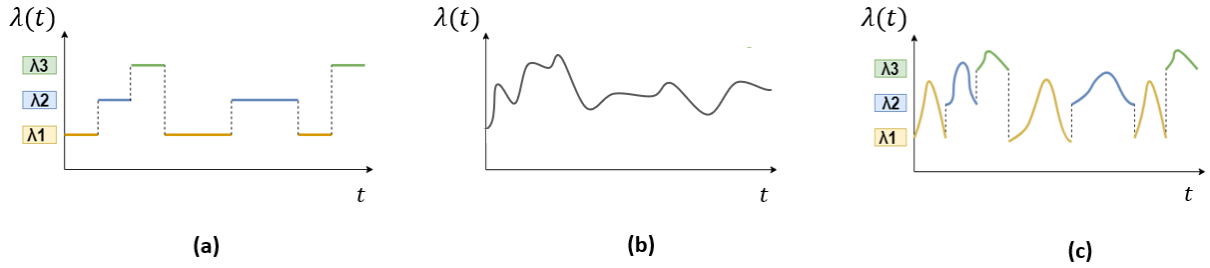


Figure 2.6: Sketch of the temporal evolution of the intensity function for a: (a) MMPP model with $k = 3$ states; (b) GPCox model; (c) MMGCP model with $k = 3$ latent functions.

modeling semi-stationary data [43]. In fact, real data are often not well modeled by an abrupt-changing piece-wise constant intensity function (as in MMPPs), nor by a smooth intensity function (as in GPCox).

Hidden Markov models (HMM) and Hidden semi-Markov models (HSMM) are yet another type of Markov modulated processes, covered in Sections 2.2.1.3 and 2.2.1.4, respectively.

2.2.1.3 Hidden Markov models

A hidden Markov model (HMM) is a statistical model used to represent sequential data with *hidden* states, using the accessible information generated by them, i.e., the observations.[17]
The HMM representation covers the dependency between observations and states in two stochastic processes, namely, a hidden process of states represented by a first-order Markov chain[18] (eq. (2.6)) and a visible process of observations, as shown in Fig. 2.7.
HMMs are widely used in different areas, specially in problems where observations need to be classified into categories. Originally, HMMs were developed for speech recognition [45] [46], where sequence of sounds are recorded (i.e., the observations) from where the sequence of pronounced phonemes (hidden states) needs to be inferred. In medical image processing, HMMs are used, for instance, to classify a brain tumor as benign or malign [47]. HMMs can also be applied for time series forecasting, for example, to predict stock market price trends [48].

Formally, a HMM with n observations and k states is represented by the tuple $(S, O, \boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B})$, where $S = (S_1, ..., S_n) \equiv S_{1:n}$ is the hidden state sequence, $O = (O_1, ..., O_n) \equiv O_{1:n}$ is the observation sequence, $\boldsymbol{A} \in \mathbb{R}^{k \times k}$ and $\boldsymbol{\pi} \in \mathbb{R}^k$ are the parameters of the Markov chain, and $\boldsymbol{B} \in \mathbb{R}^{k \times m}$ is the matrix of observation probabilities. Each observation[19] takes values in a set $\mathcal{V} = \{v_1, ..., v_m\}$ of m possible symbols. Hence, matrix $\boldsymbol{B}$ defines, for each state, the probability of observing each of the m possible observation values. The HMM parameters, which completely specify it, are $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}\}$.

[17] The state sequence is hidden because, unlike observations, states are not observed.
[18] We will consider discrete first-order Markov chains. However, the HMM can also be defined by CTMCs.
[19] Also called emission or symbol.

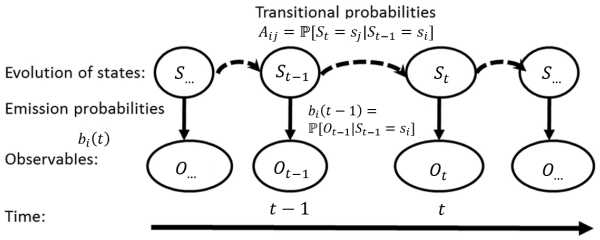


Figure 2.7: Structure and temporal evolution process of a HMM system. Adapted from [49].

Sampling from a HMM Sampling in HMMs is the act of obtaining the hidden state and observation sequences ($S_{1:n}$ and $O_{1:n}$, respectively) from the model parameters (μ) by sampling the corresponding probabilities. That is, at time $t = 1$, the first state of the hidden state sequence (S_1) is sampled from the initial state distribution $\boldsymbol{\pi}$. At each time $t \in [1, n]$, the state $S_t = s_i \in \mathcal{S}$, where the Markov chain is at, dictates the emission probability

$$b_i(t) = b_{S_t = s_i}(O_t = o_t) = \mathbb{P}[O_t = o_t | S_t = s_i] = \begin{cases} B_{io_t}, & \text{if multinomial.} \\ \dfrac{e^{-\lambda_i} {\lambda_i}^{o_t}}{o_t!}, & \text{if Poisson.} \end{cases} \tag{2.9}$$

from where the observation $O_t = o_t \in \mathcal{V}$ is obtained.[20] Then, the Markov chain moves to the next state $S_{t+1} = s_j$ according to the transition probability in eq. (2.7), where a new observation o_{t+1} is obtained from its emission probability, and so on.

Usually, the emission probability in eq. (2.9) is a multinomial distribution and therefore $B_{io_t} = \boldsymbol{B}(i, o_t)$. However, one may consider a parametric emission distribution, such as the Poisson distribution (see Section 2.1.1). In that case, we define a vector of Poisson rates associated to each state, $\boldsymbol{\lambda} \in \mathbb{R}^k$, and the emission probability in eq. (2.9) is the Poisson PMF where λ_i is the parameter of state s_i.[21] In this scenario, the HMM models the intensity function as a inhomogeneous Poisson process (Fig. 2.4).

The joint probability that the HMM will generate the observation sequence $O_{1:n}$ with underlying state sequence $S_{1:n}$ is given by

$$\mathbb{P}(S, O; \mu) \underset{Bayes}{=} \mathbb{P}[O|S, \mu]\mathbb{P}(S; \mu) = \pi_{S_1} b_{S_1}(o_1) \prod_{t=2}^{n} \boldsymbol{A}_{S_{t-1} \to S_t} b_{S_t}(o_t), \tag{2.10}$$

which is also called the *joint likelihood* of the states with the observations.

The joint likelihood is used to learn the model parameters μ[22] that best explain the observations $O_{1:n}$. We explore the HMM supervised learning approach in Section 2.2.2.1 and the unsupervised in Section 2.2.2.4. Even in the simplest (supervised) learning process, the HMM is based on some assumptions. For instance, the number of states (k) of the Markov chain is known and all transitions are possible[23]. However, this is not always true and, in Chapter 3, we discuss how to address these questions.

[20] In some cases, we replace the random variable and its realization ($O_t = o_t$) simply by the instantiation o_t, for ease of notation.

[21] To make the analogy to the multinomial case, an emission probability matrix $\boldsymbol{B}$ can also be constructed, where each element of the matrix is given by the (normalized) Poisson PMF.

[22] If Poisson emissions are considered, the parameter $\boldsymbol{\lambda}$ is learnt along with the model parameters μ.

[23] $A_{ij} > 0, \quad \forall s_i, s_j \in \mathcal{S}$.

One of the main disadvantages of HMMs is that it is not possible to directly change the duration[24] of the system in a particular state. This is implicitly controlled by the transition matrix $\boldsymbol{A}$. In fact, the duration in state s_i of a k-state HMM is modeled by a geometric distribution with fixed parameter $A_{ii}, \forall i \in [k]$, which corresponds to the diagonal entries of the transition probability matrix $\boldsymbol{A} \in \mathbb{R}^{k \times k}$ [50]. Therefore, the probability of duration d in state s_i (known as $p_i(d)$) is given by the geometric probability of $d-1$ times in the same state s_i (failures), before the first transition to any other state (success). That is,

$$p_i(d) = \text{Geom}(d|A_{ii}) = A_{ii}^{d-1}(1 - A_{ii}), \qquad \forall d = 1, ..., d_{max}, \tag{2.11}$$

where A_{ii} represents the success probability of the geometric distribution and d_{max} is the maximum allowed state duration.

The implicit modeling of durations in HMMs can be limiting for real data. Instead, it is advantageous to explicitly specify a state duration distribution as it provides greater expressiveness to the states. *Hidden semi-Markov models* are an extension of HMMs designed to achieve this goal.

2.2.1.4 Hidden semi-Markov models

In *Hidden semi-Markov models* (HSMMs), the state duration is modeled explicitly through non-parametric (e.g. multinomial distribution) or parametric distributions (e.g. Poisson, geometric, etc).

HSMMs have been vastly used for speech synthesis [51] to model the phoneme duration so that the required naturalness of speech is achieved. Another example where the need for explicit durations arise is in time-series segmentation of human motion data [52], where HSMMs, contrary to HMMs, can correctly cluster the different motion activities.

In HSMMs, the state sequence follows a *semi-Markov chain* (SMC), where each state of the SMC has a variable duration taking values from the set $\mathcal{D} = \{1, 2, ..., d_{max}\}$ of possible durations. The duration corresponds to the number of observations produced while in a state. Hence, in HSMMs, each state can emit a sequence of observations [53], as shown in Fig. 2.8.

A SMC with k states is a system defined by a hidden random sequence $S_{1:n}$ and fully characterized by the set of parameters $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{D}\}$. Before defining each SMC parameter in particular, we introduce some notation. As shown in Fig. 2.8, we only know that a state transition occurs after the duration for the current state (d) is over, i.e., after we emit d observations. In other words, without the knowledge about the state duration, being in a state at time t is no longer a signal of a state transition[25] in $t+1$, as it was in HMMs[26]. Therefore, the notation of being in a state at time t is replaced by $S_{\rightarrow t|} = s_j$, which means that state s_j ends at time t, and $S_{|t+1\rightarrow} = s_j$, which means that state s_j starts at time $t+1$.

We are now are able to define each SMC parameter $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{D}\}$:

- The general HSMM transition matrix is not the same as in HMMs, since it introduces the concept of duration by conditioning each state transition in the duration of both states. That is, $\boldsymbol{A} \in \mathbb{R}^{k^2 \times {d_{max}}^2}$ is a matrix whose element $A_{id',jd}$ represents a transition from state s_i with duration d' to state s_j

[24] The duration corresponds to the number of observations made while in a state.

[25] Even if it is a self-transition, i.e., a transition to the same state.

[26] Given that, in HSMMs, self-transitions are not allowed due to the explicit duration modeling, i.e., $A_{ii} = 0$.

with duration d,

$$A_{id',jd} = \mathbb{P}[S_{t:t+d-1} = s_j | S_{t-d'+1:t} = s_i], \qquad s_i, s_j \in \mathcal{S}, \quad d, d' \in \mathcal{D}. \tag{2.12}$$

- An initial state probability distribution $\boldsymbol{\pi} \in \mathbb{R}^{k \times 1}$, with elements $\pi_i = \mathbb{P}[S_{|1\rightarrow} = s_i], \quad \forall i \in [k]$;
- $\boldsymbol{D} \in \mathbb{R}^{k \times d_{max}}$ is the duration probability matrix, where element $D_{jd} = p_j(d)$ represents the probability of occupying state s_j with duration d. The state duration distribution can be non-parametric (e.g. multinomial) or parametric (e.g. geometric, as in HMMs[27]). In parametric state duration distributions, k parameters, one for each state, are used to characterize the durations in the SMC. For instance, in geometric state duration distributions, the PMF of the durations for state s_i is given by eq. (2.11), with an additional parameter $\theta_i \in [0, 1]$ (the parameter of the geometric distribution) instead of A_{ii};
- The hyperparameters are the total number of states k and the maximum allowed state duration d_{max}, which represents the maximum number of observations within a state and it is considered the same for all states. It is used to truncate the forward-backward algorithm (see Section 2.2.2.5) so that the computational burden of the algorithm can be alleviated. These hyperparameters must be defined, with either *a priori* specification or by order estimation strategies (see Chapter 3).

Based on the definition of a semi-Markov chain, a HSMM is a probabilistic model in which a stochastic sequence of observations $O_{1:n}$ is generated by a SMC. The observed symbol at time t (o_t) depends on the state of the SMC at time t (S_t). Therefore, a k-state HSMM is entirely defined by $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}, \boldsymbol{D}\}$, where, aside from the above-mentioned SMC parameters, the HSMM also specifies:

- An emission probability

$$b_{i,d}(O_{t:t+d-1} = o_{t:t+d-1}) = \mathbb{P}[o_{t:t+d-1} | S_{t:t+d-1} = s_i], \tag{2.13}$$

that represents the probability of d observations from time t onwards, while in state s_i.

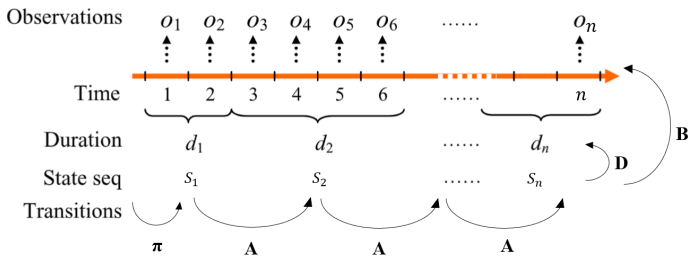


Figure 2.8: General HSMM. Sampling process in a HSMM. Adapted from [53].

Sampling from a HSMM Figure 2.8 illustrates the sampling process in a general HSMM, from where one can easily point out the similarities and disparities between HMMs and HSMMs. In both models,

[27] With the difference that, in HSMMs, we control the parameter of the geometric distribution instead of being fixed by the diagonal of $\boldsymbol{A}$.

the first state is sampled from the initial state distribution $\boldsymbol{\pi}$ and the transitions between states are sampled from the transition matrix $\boldsymbol{A}$. However, in HMMs the transitions between states occur at every timestep and self-transitions are allowed. In contrast, in HSMMs, the transitions between states occur only after the sampled duration for the current state has passed and self-transitions are not allowed.[28] For example, in Fig. 2.8, state $S_1 \in \mathcal{S}$ is sampled from $\boldsymbol{\pi}$. In state S_1, the duration $d_1 = 2$ is sampled from the duration probability matrix $\boldsymbol{D}$. Thus, two observations are generated while in that state. After that, a transition from state S_1 to state S_2 is sampled from $\boldsymbol{A}$, and so on.

Variants of HSMMs There are many variants of HSMMs, such as explicit-duration HMM, variable transition HMM, residential time HMM, among others [53]. The simplest one is the explicit-duration HMM (EDHMM)[29], which assumes that a state transition is not dependent on the duration of the previous state. Thus, eq. (2.12) simplifies to [53]

$$A_{id',jd} = A_{i,jd} = \mathbb{P}[S_{t:t+d-1} = s_j | S_{\rightarrow t|} = s_i] = A_{ij} p_j(d). \tag{2.14}$$

The diagonal entries of the transition matrix are zero ($A_{ii} = 0$) since self-transitions are not allowed. Moreover, in EDHMMs, it is considered that the observations are conditionally independent given the state. That is,

$$\mathbb{P}[o_{t:t+d-1} | S_{t:t+d-1} = s_i] \underset{ind.}{=} \prod_{\tau=t}^{\tau=t+d-1} \mathbb{P}[o_\tau | S_\tau = s_i] = \prod_{\tau=t}^{\tau=t+d-1} b_i(\tau). \tag{2.15}$$

The conditional independence property of the observations in eq. (2.15) allows us to use the simpler version of the emission probability, from eq. (2.9), rather than the one in eq. (2.13). Here it is assumed that the emission distribution is non-parametric. Therefore, matrix $\boldsymbol{B}$ represents a multinomial distribution. However, one could consider parametric state emission distributions, for example, Poisson processes.
Likewise, the transition probability matrix $\boldsymbol{A} \in \mathbb{R}^{k \times k}$, as defined in HMMs (eq. (2.7)) is used rather than the one in eq. (2.12).
Since these models are parametric, the training phase consists in estimating the parameters μ (see Section 2.2.2.4 for HMMs and Section 2.2.2.5 for HSMMs). However, due to the use of explicit durations, the estimation problem for HSMMs is more complex than the one for HMMs. After training, time series forecasting is easily performed using the previously estimated parameters (see Chapter 4). To sum up, the advantage of using HSMMs instead of HMMs is that the durations are modeled explicitly, providing the model a higher flexibility in identifying patterns with different lengths, at the cost of a more complex estimation procedure, as shown in the Section 2.2.2.
Artificial neural networks (ANNs) are yet another type of parametric model that can be used for time-series modeling and forecasting, detailed in the following section.

[28] In HMMs the Markov assumption is followed and, consequently, the state transitions at each time step are independent. On the contrary, in HSMMs the Markov assumption no longer holds because the transition probabilities depend on the duration of the previous state and therefore a state probability history is needed.
[29] This is the HSMM version adopted in the context of this dissertation.

2.2.1.5 Artificial neural networks

ANNs are outstandingly efficient models in learning complicated data patterns. The first proposed ANN model was the Rosenblatt 1958 Perceptron model [54], whose design is inspired in human neurons. Ultimately, the goal of these models is to learn complex non-linear functions by replicating the way information travels in the human brain. From the wide range of ANN configurations currently available, we will only cover two of them: *recurrent neural networks* and *long short-term memory neural networks*.

Recurrent neural networks Recurrent neural networks (RNNs) are efficient models for sequential data as they, unlike HMMs, do not follow the Markovian assumption, i.e., their internal state is able to capture dependencies well behind the previous state. In Fig. 2.9(a), we present the architecture of a RNN, composed by a recurrent cell that receives a sequence $o_{1:n}$ as input, one sample at a time. The input sequence can contain temporal observations and external features. The cell is said to be recurrent because it contains an internal feedback loop - with feedback variable S, the hidden state, which allows past information to persist in the system. Thus, RNNs have selective[30] memory of past events.
The hidden state at time t (S_t) is a function of the hidden state from the previous timestep and the current input. The hidden state is responsible to compute the next state and the output at the current timestep ($\hat{y}_t$), which is a weighted version of the hidden state. The loss is computed between each output $\hat{y}_t$ and the true label y_t.[31,32] After obtaining the loss, in order to update the weights, we need to compute the partial derivative of the loss w.r.t the weights. Then, we backpropagate [55] the derivative to the previous hidden layer and so on, as shown in Fig. 2.9(b). However, the backpropagation induces a *gradient vanishing* problem, similar to numerical underflow in HMMs and HSMMs (see Sections 2.2.2.4 and 2.2.2.5). As gradients travel backwards in the network, they get multiplied by factors smaller than 1, which will likely converge to zero. Therefore, no weight update is performed and the model will not learn long term dependencies in the data (*catastrophic forgetting*) or it will take too long to finish learning.

Long short-term memory In 1997, Hochreiter and Schmidhuber [56] proposed the Long short-term memory (LSTM), which prevents catastrophic forgetting by adding to the hidden state S_t a cell state C_t, a forget gate F_t, an input gate I_t and an output gate G_t, given by

$$\begin{cases} F_t = \sigma\left(W_O^F o_t + W_S^F S_{t-1} + b_F\right) \\ I_t = \sigma\left(W_O^I o_t + W_S^I S_{t-1} + b_I\right) \\ G_t = \sigma\left(W_O^G o_t + W_S^O S_{t-1} + b_G\right) \\ C_t = F_t \cdot C_{t-1} + I_t \cdot \tanh\left(W_O^C o_t + W_S^C S_{t-1} + b_C\right) \\ S_t = G_t \cdot \tanh\left(C_t\right) \end{cases}, \tag{2.16}$$

[30] The hidden state is responsible to select which information persists and which is vanished.
[31] The loss can be, for example, the mean-squared-error between the labels.
[32] In sequential data, the labels can be held out from the input sequence itself by using sliding windows.

where W and b are weights and bias, respectively[33]. The gates with a sigmoid activation function (σ) in eq. (2.16) act as gatekeepers of information, as they keep values when their outputs are close to 1 and remove values when their outputs are close to 0. Each of the gates in eq. (2.16) is responsible for a different task in managing the states of the LSTM. The forget gate determines which elements from the previous cell state C_{t-1} are to be forgotten or kept. The input gate measures how much information, from the current observation o_t and the previous hidden state S_{t-1}, is to be added to the current cell state C_t. After obtaining the current cell state, the output gate selects the values of the cell state that are reflected in the current hidden state. Finally, both the current hidden and cell states (S_t and C_t) are passed to the next timestep. This process is illustrated in the LSTM cell at time t, shown in Fig. 2.9(c). Although the LSTM architecture is far more complex than the one in RNNs, it has the great advantage of being able to learn both long and short-term dependencies by effectively controlling the stream of information that is retained or erased over time. LSTMs have been recently applied to time-series forecasting in many

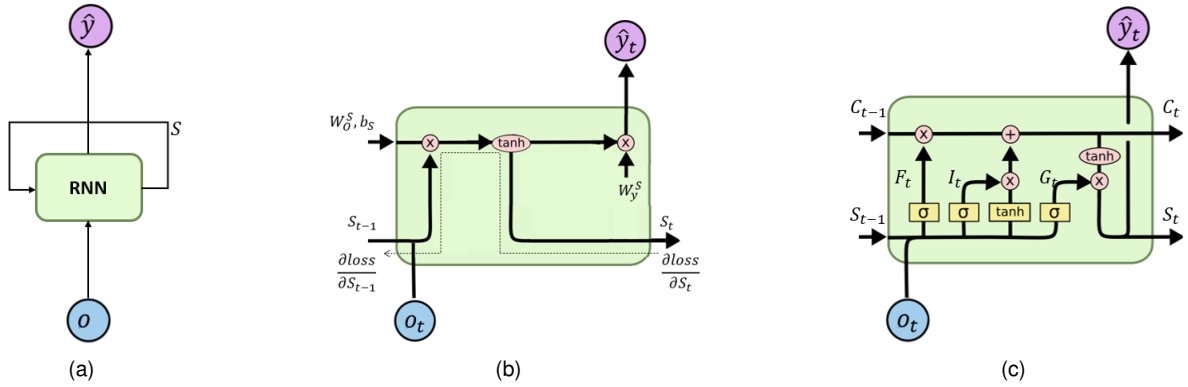


Figure 2.9: (a) Recurrence in a RNN. The input is represented as $o \equiv o_{1:n}$, the predicted output as $\hat{y}$ and the hidden state as S; (b) Standard RNN cell, with input weight w_O^S and bias b_S applied to the hidden state S_{t-1} and the input o_t, and output weight W_y^S; (c) Standard LSTM cell as defined in eq. (2.16). The weights and bias are not shown for a better visualization. Adapted from `https://colah.github.io/`.

domains including traffic [8], engineering [9], weather [6], finance [4] and computer vision [10].

2.2.2 Parametric estimation in Markov models

In the present section, we gradually address the estimation problem in parametric Markov models. We start by reviewing the supervised learning in HMMs, then we jump to the unsupervised learning for a single and a combination of Poisson processes and finally unsupervised learning in HMMs and HSMMs.

2.2.2.1 Supervised learning: HMMs

In the supervised learning setting, not only the observation sequence is available to the learning phase but also the hidden state sequence responsible for those observations. The HMM is trained using n pairs of state/observations $(S_{1:n}, O_{1:n})$. We want to find the best set for the HMM's parameters $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}\}$

[33]The superscript and subscript in the weights W represents the nodes involved in the multiplication. For example, W_O^F denotes the weight of the input o_t (at time t) when applied to the forget gate F_t. The subscript in the bias is the corresponding gate/cell.

according to the observations $(S_{1:n}, O_{1:n})$, using *maximum likelihood* estimators. That is,

$$< \hat{\boldsymbol{A}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{B}} >= \underset{<\boldsymbol{A},\boldsymbol{\pi},\boldsymbol{B}>}{\arg\max} \quad \mathbb{P}[(S,O)|\boldsymbol{A},\boldsymbol{\pi},\boldsymbol{B}]. \tag{2.17}$$

As shown in Appendix A, this problem is solved by simple counting and normalizing operations:

$$\hat{A}_{ij} = \frac{n_{ij}}{\sum_{j=1}^{k} n_{ij}}, \qquad \hat{\pi}_i = \frac{n_{0i}}{\sum_{i=1}^{k} n_{0i}}, \qquad \hat{B}_{jv} = \frac{n_{jv}}{\sum_{v=\min(O)}^{v=\max(O)} n_{jv}}, \qquad \forall i,j \in [k], \qquad v \in \mathcal{V}, \tag{2.18}$$

where n_{ij} is the number of transitions from state s_i to state s_j in the given state sequence, n_{0i} is the number of times state s_i is the first state and n_{jv} is the number of times the observation takes value v in state s_i. In Appendix A, the HMM emission probability is estimated both non-parametrically as a multinomial distribution and parametrically as a Poisson distribution.

The simplicity from this scenario comes from the fact that the state sequence is observed. The following sections address the scenario with a hidden state sequence, which is more realistic.

2.2.2.2 Unsupervised learning: Single homogeneous Poisson process

The learning procedure of a single homogeneous Poisson process requires the estimation of its parameter, i.e., the (scalar) rate λ that represents the expected number of events in a fixed interval[34] defined by a discretization unit ($\triangle t$). [35] After obtaining the vector of observations $o_{1:n}$ discretized by $\triangle t$, the Poisson parameter λ is promptly estimated using maximum likelihood. The log-likelihood of the observation vector $o_{1:n}$ is

$$\log\Big(\mathbb{P}(o_{1:n};\lambda)\Big) \underset{iid}{=} \log\left(\prod_{t=1}^{n} \frac{e^{-\lambda}\lambda^{o_t}}{o_t!}\right) = \sum_{t=1}^{n} -\lambda + o_t\log(\lambda) - \log(o_t!). \tag{2.19}$$

The maximum likelihood estimator of λ is obtained from eq. (2.19) as

$$\hat{\lambda}_{\mathsf{MLE}} = \arg\max_{\lambda}\Big(\log\Big(\mathbb{P}(o_{1:n};\lambda)\Big)\Big), \tag{2.20}$$

i.e., the maximum of the log-likelihood[36] function of the observations w.r.t the parameters. Analytically, this is achieved by taking the derivative of the log-likelihood w.r.t the parameters and equating it to zero to find a stable point. Furthermore, by analyzing the signal (curvature) of the second derivative we can verify if the set of parameters found corresponds indeed to a maximum of the log-likelihood function. That is,

$$\frac{\partial \log\Big(\mathbb{P}(o_{1:n};\lambda)\Big)}{\partial\lambda}\bigg|_{\lambda=\hat{\lambda}} = 0 \quad \wedge \quad \frac{\partial^2 \log\Big(\mathbb{P}(o_{1:n};\lambda)\Big)}{\partial\lambda^2}\bigg|_{\lambda=\hat{\lambda}} \geq 0. \tag{2.21}$$

Solving eq. (2.21) yields

$$\sum_{t=1}^{n}\Big(-1+\frac{o_t}{\hat{\lambda}}\Big) = 0 \Longleftrightarrow \hat{\lambda} = \frac{1}{n}\sum_{t=1}^{n} o_t. \tag{2.22}$$

[34] A Poisson process with $\lambda = 2$ has two expected events within one unit length. In real datasets, the discretization of the unit length can be concretized in, e.g. two expected events in one day, one week, etc, depending on what is reasonable for the data.

[35] Note that in a mixture of Poisson processes the discretization unit must be the same for all Poisson distributions.

[36] The log-likelihood is preferred over the likelihood for mathematical convenience. This can be done because the logarithm is a monotonic increasing function and therefore the maximum is left unchanged.

By analyzing eq. (2.22), we conclude that the maximum likelihood estimator for the rate of a single homogeneous Poisson process is obtained as the mean of the observations.
A more interesting case is discussed for combined Poisson processes.

2.2.2.3 Unsupervised learning: Combined Poisson processes

The combination of two or more homogeneous Poisson processes gives rise to more complex estimation procedures. Consider the merging of two Poisson processes, as illustrated by Property 5 in Section 2.1.1. Both Poisson processes are described by their PMF with parameters λ_1 and λ_2, respectively. A random variable Z that follows a Bernoulli distribution with parameter α is used to switch between the two processes, as shown in Fig. 2.10. The switch activates the observations from the first process with probability α and triggers the observations from the second process with probability $1-\alpha$.

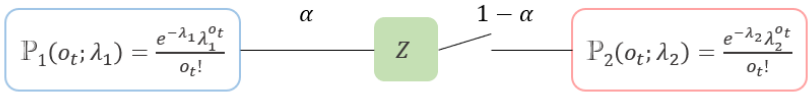


Figure 2.10: Diagram of the combination of two homogeneous Poisson processes with rate λ_1 and λ_2. The combination is done through a switch activated by a Bernoulli random variable Z.

Thus, the likelihood of one observation o_t is given by

$$\mathbb{P}(o_t;\lambda_1,\lambda_2,\alpha) = \alpha\mathbb{P}_1(o_t;\lambda_1) + (1-\alpha)\mathbb{P}_2(o_t;\lambda_2), \tag{2.23}$$

where $\mathbb{P}_1(o_t;\lambda_1) = (e^{-\lambda_1}\lambda_1^{o_t})/o_t!$ and $\mathbb{P}_2(o_t;\lambda_2) = (e^{-\lambda_2}\lambda_2^{o_t})/o_t!$. From eq. (2.23), there are 3 parameters $\mu = \{\lambda_1,\lambda_2,\alpha\}$ that need to be estimated from the data. Again, we can estimate them from the observation log-likelihood, defined, in this case, as

$$\log\Big(\mathbb{P}(o_{1:n};\mu)\Big) \underset{\text{iid}}{=} \log\left(\prod_{t=1}^{n}\mathbb{P}(o_t;\mu)\right) = \sum_{t=1}^{n}\log\left(\alpha\frac{e^{-\lambda_1}\lambda_1^{o_t}}{o_t!} + (1-\alpha)\frac{e^{-\lambda_2}\lambda_2^{o_t}}{o_t!}\right), \tag{2.24}$$

where we used eq. (2.23). Note that the maximum likelihood approach for this model has no closed-form solution since we need to compute derivatives of logarithms with summations inside. Therefore, the set of estimated parameters $\hat{\mu}_{\text{MLE}}$ cannot be found analytically. The method typically used to estimate the parameters when there is no closed-form solution for the derivatives of the log-likelihood function is the *Expectation-Maximization* (EM) algorithm [57], generically described in Appendix B, which comprises two steps: E-step and M-step.

E-step In this step, two expectations (or weights) w_{1_t} and w_{2_t} are computed:

- The probability that sample o_t was generated by the first (second) Poisson process, w_{1_t} (w_{2_t}) is

$$w_{1_t} = \frac{\hat{\alpha}\mathbb{P}_1(o_t;\hat{\lambda}_1)}{\hat{\alpha}\mathbb{P}_1(o_t;\hat{\lambda}_1) + (1-\hat{\alpha})\mathbb{P}_2(o_t;\hat{\lambda}_2)}, \qquad w_{2_t} = \frac{(1-\hat{\alpha})\mathbb{P}_2(o_t;\hat{\lambda}_2)}{\hat{\alpha}\mathbb{P}_1(o_t;\hat{\lambda}_1) + (1-\hat{\alpha})\mathbb{P}_2(o_t;\hat{\lambda}_2)}. \tag{2.25}$$

- The probability of the sample o_t is 1 since it is observed. That is, $\mathbb{P}(o_t; \mu) = 1 = \sum_{j=1}^{p} w_{j_t}$, where p is the number of Poisson processes being combined. Since, in this case, we are considering a combination of two Poisson processes ($p = 2$), then $w_{2_t} = 1 - w_{1_t}$.

M-step In the M-step, the expectations obtained in the E-step are used to re-estimate the set of parameters $\hat{\mu} = \{\hat{\alpha}, \hat{\lambda}_1, \hat{\lambda}_2\}$, by counting and normalizing the expected probabilities. That is,

$$\hat{\alpha} = \frac{1}{n} \sum_{t=1}^{n} w_{1_t}, \quad \hat{\lambda}_1 = \frac{\sum_{t=1}^{n} w_{1_t} o_t}{\sum_{t=1}^{n} w_{1_t}}, \quad \hat{\lambda}_2 = \frac{\sum_{t=1}^{n} w_{2_t} o_t}{\sum_{t=1}^{n} w_{2_t}}. \tag{2.26}$$

After examining a single homogeneous Poisson process and the combination of two of them, remains the question of how to combine multiple Poisson processes. As seen before, this can be done through Markov modulated Poisson processes: with HMMs or HSMMs. The unsupervised learning strategy for HMMs and HSMMs is described in Sections 2.2.2.4 and 2.2.2.5, respectively.

2.2.2.4 Unsupervised Learning: HMMs

The HMM unsupervised learning approach requires the computation of the likelihood of an observation sequence, ($\mathbb{P}(O; \mu)$). Recall that, so far, we have only defined, in eq. (2.10), the joint likelihood of the states with the observations ($\mathbb{P}(S, O; \mu)$). However, by marginalizing out all possible hidden states we can obtain the likelihood, i.e., $\mathbb{P}(O; \mu) = \sum_S \mathbb{P}(S, O; \mu)$. Since the total number of possible state sequences is k^n, for a $k-$state HMM, the complexity of computing the likelihood of an observation sequence $O_{1:n}$ is $\mathcal{O}(k^n)$, which can easily become unfeasible when k and n are reasonably large.

Forward algorithm The *forward* algorithm takes advantage of dynamic programming to reduce the complexity to $\mathcal{O}(k^2 n)$ [58]. Hence, the computational time increases linearly, instead of exponentially, with the number of observations n. The idea is to avoid repeating computations by storing and reusing partial results in a recursive manner. To do so, we define the forward variable $\alpha_t(j)$, which represents the joint probability that the HMM is in state s_j after seeing the first t observations. The definition and recursive computation of the forward variable is presented in eq. (2.27) and the initialization in eq. (2.28).

$$\alpha_t(j) = \mathbb{P}[S_t = s_j, O_{1:t}; \mu] = \sum_{i=1}^{k} \alpha_{t-1}(i) A_{ij} b_j(t), \qquad \forall t = 2, 3..., n \tag{2.27}$$

$$\alpha_1(j) = \pi_j b_j(1), \qquad \forall j \in [k]. \tag{2.28}$$

Note that equation (2.27) sums over all possible previous states s_i, of a transition to state s_j and an emission with value o_t. Both eq. (2.28) and eq. (2.27) represent a recursive version of eq. (2.10). The likelihood of the observations can be promptly obtained as the sum of the forward variables of all states

in the final timestep. That is,

$$\mathbb{P}(O_{1:n} = o_{1:n}; \mu) = \sum_{j=1}^{k} \mathbb{P}[S_n = s_j, o_{1:n}; \mu] = \sum_{j=1}^{k} \alpha_n(j). \tag{2.29}$$

Viterbi algorithm Another HMM algorithm, known for decoding the most likely hidden state sequence from the observations, is the *Viterbi algorithm*. We introduce the Viterbi variable, $v_t(j)$, which represents the probability of being in state s_j after seeing the first t observations and traversing the most probable state sequence $S_{1:t-1}$, i.e.,

$$v_t(j) = \max_{(S_1,\dots,S_{t-1})} \mathbb{P}[S_t = s_j, o_{1:t}, S_{1:t-1}] = \max_{i=1}^{k} \quad v_{t-1}(i) A_{ij} b_j(t). \tag{2.30}$$

The similarity between eqs. (2.27) and (2.30) suggests that the only difference between the Viterbi algorithm and the Forward algorithm is that in the former we compute the $\max$ instead of the summation. In order to determine the state sequence that corresponds to the most probable path, we store backtrace pointers to the maximum Viterbi value between all states in the previous timestep. The optimal path represents the most likely hidden state sequence that can be found by following the backtrace pointers from the last timestep ($t = n$) to the first ($t = 1$).
In conclusion, the Viterbi algorithm finds the optimal path that maximizes the likelihood of the entire observation sequence. On the other hand, the well-known *forward-backward* algorithm [59] finds the posterior distribution over missing data (hidden states) for each observation separately.

Forward-backward algorithm In the forward-backward (FB) algorithm, the optimal underlying state $\hat{S}_t$ is the one with highest probability for the observation at time t. That is,

$$\hat{S}_t = \arg\max_{s_i} \quad \mathbb{P}[S_t = s_i | O, \mu] = \arg\max_{s_i} \quad \gamma_t(i), \tag{2.31}$$

where $\gamma_t(i)$ is the posterior distribution of state s_i, i.e., the probability that at time t the system is at state s_i after observing the whole observation sequence O. It is computed from

$$\gamma_t(i) \underset{Bayes}{=} \frac{\mathbb{P}[S_t = s_i, O; \mu]}{\mathbb{P}(O; \mu)} \underset{ind}{=} \frac{\mathbb{P}[S_t = s_i, O_{1:t}; \mu]\mathbb{P}[O_{t+1:n} | S_t = s_i, \mu]}{\mathbb{P}(O; \mu)} =$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{k} \mathbb{P}[S_t = s_i, O; \mu]} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{k} \alpha_t(i)\beta_t(i)}, \tag{2.32}$$

where $\beta_t(i)$ is the backward variable defined and initialized as

$$\beta_t(i) = \mathbb{P}[O_{t+1:n} | S_t = s_i, \mu]; \qquad \beta_n(i) = 1, \qquad \forall i \in [k], \tag{2.33}$$

which represents the likelihood of the next observations from t up to the final instant n knowing that at time t the Markov chain is in state s_i. It is initialized as ones, for each state, at the last timestep.[37] The

[37] Since it conditions an outside observation on the states and 1 is the neutral element of the multiplication.

consecutive values for the backward variable are computed using the following recursive relation:

$$\beta_{t-1}(i) = \sum_{j=1}^{k} A_{ij} b_j(t) \beta_t(j), \qquad \forall i \in [k], \quad \forall t = n, n-1, ..., 2. \tag{2.34}$$

The likelihood of the observations can be obtained as

$$\mathbb{P}(O; \mu) = \sum_{j=1}^{k} \mathbb{P}[S_1 = s_j, O_{1:n}; \mu] = \sum_{j=1}^{k} \beta_1(j) \pi_j b_j(1). \tag{2.35}$$

Numerical underflow From inspection of eqs. (2.27) and (2.34), it is clear that numerical underflow will occur as $n \to \infty$ since we are working with increasingly large products of probabilities. Examples of commonly used strategies found in the literature to tackle this problem include scaling the forward and backward variables [50] [60] and transferring the probabilities to the log-domain [61].

HMM parameter estimation In the above-mentioned algorithms, the HMM parameters ($\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}\}$) were fixed. It is important to guarantee that these parameters can properly represent our data. The problem of choosing the best HMM parameters, the ones that maximize the likelihood of the observations, is called *parameter estimation*. The *Baum-Welch* algorithm assures that the HMM parameters found *locally* maximize the observation likelihood [59]. It is an expectation-maximization (EM) algorithm that iteratively estimates and updates the HMM parameters based on the forward-backward procedure.

E-step The E-step of the EM algorithm corresponds to the forward-backward algorithm, where the state posterior probability in eq. (2.32) is computed using the parameter estimates ($\hat{\mu}$) from the current iteration. After obtaining the posterior, the expected transition and emission counts are obtained. The expected number of transitions from state s_i (to any other state) is given by

$$\sum_{t=1}^{n} \mathbb{P}[S_t = s_i | O, \mu] = \sum_{t=1}^{n} \gamma_t(i), \quad \forall i \in [k]. \tag{2.36}$$

The expected number of transitions from state s_i to state s_j is given by

$$\sum_{t=2}^{n} \mathbb{P}[S_{t-1} = s_i, S_t = s_j | O, \mu] = \sum_{t=2}^{n} \zeta_t(i,j), \quad \forall i, j \in [k], \tag{2.37}$$

where $\zeta_t(i,j)$ is the probability of a transition from state s_i at time $t-1$ to state s_j at time t, given the whole observation sequence and the model parameters. The expected transition counts are obtained from eq. (2.37) as [50]

$$\zeta_t(i,j) = \frac{\alpha_{t-1}(i) A_{ij} b_j(t) \beta_t(j)}{\sum_{i=1}^{k} \sum_{j=1}^{k} \alpha_{t-1}(i) A_{ij} b_j(t) \beta_t(j)}. \tag{2.38}$$

Equation (2.38) requires running the forward-backward algorithm and computing the forward and backward variables $\alpha_t(i)$ and $\beta_t(i)$ for all timesteps $t = 1, ..., n$ and all hidden states $i \in [k]$. Subsequently, eq. (2.37) is used to get the expected number of transitions for the entire state sequence.[38] Eqs. (2.32)

[38] Note that the denominators in eqs. (2.32) and (2.38) are normalizing terms that represent the observation likelihood, $P(O; \mu)$.

and (2.38) are easily scaled or transferred to the log-domain to avoid numerical underflow [62].

M-step In the M-step of the EM algorithm, the set of parameters (μ) is re-estimated, using the results from the E-step, as ratio of expected counts. That is,

$$\hat{A}_{ij} = \frac{\sum_{t=2}^{n} \zeta_t(i,j)}{\sum_{j=1}^{k}\sum_{t=2}^{n} \zeta_t(i,j)}, \qquad \hat{\pi}_i = \frac{\gamma_1(i)}{\sum_{i=1}^{k} \gamma_1(i)}, \qquad \hat{b}_i(O_t = v) = \frac{\sum_{t=1}^{n} \gamma_t(i)\mathbf{1}_{(O_t=v)}}{\sum_{t=1}^{n} \gamma_t(i)}. \tag{2.39}$$

In eq. (2.39), the emission probabilities are estimated as multinomial distributions. Parametric distributions can be used instead. For example, with Poisson emissions, the observation probability $b_i(t)$ is the Poisson PMF and its parameter λ_i is estimated as

$$\hat{\lambda}_i = \frac{\sum_{t=1}^{n} \gamma_t(i) o_t}{\sum_{t=1}^{n} \gamma_t(i)}, \tag{2.40}$$

i.e., the average of all observations weighted by the corresponding state posterior distribution.
After exploring the unsupervised learning process in HMMs, we analyze what adaptations need to be made, in the forward-backward formulas, for HSMMs. Those changes affect not only the computation of the observation likelihood but also the reestimation of the model parameters.

2.2.2.5 Unsupervised Learning: HSMMs

The HSMM parameter estimates ($\hat{\mu} = \{\hat{\boldsymbol{A}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{B}}, \hat{\boldsymbol{D}}\}$) are also obtained from the EM algorithm.

E-step As before, the E-step corresponds to the forward-backward algorithm. The HSMM forward variable is defined as the joint probability that state s_j ends at t and the observations up to time t, i.e., [39]

$$\alpha_t(j) = \mathbb{P}[O_{1:t}, S_{\rightarrow t|} = s_j; \mu]. \tag{2.41}$$

Equation (2.41) can be re-written (proof can be found in [50]), within the EDHMM formulation, defined in Section 2.2.1.4, as

$$\alpha_t(j) = \sum_{i=1}^{k} \sum_{d=1}^{\min(d_{max},t)} \alpha_{t-d}(i) A_{ij} D_{jd} \prod_{\tau=t-d+1}^{t} b_j(\tau) = \sum_{d=1}^{\min(d_{max},t)} \alpha^*_{t-d}(j) D_{jd} \prod_{\tau=t-d+1}^{t} b_j(\tau), \tag{2.42}$$

where $\alpha^*_t(j) \in \mathbb{R}^k$ is an auxiliary variable that represents the joint probability of the observations up to time t and that state s_j starts at $t+1$. That is,

$$\alpha^*_t(j) = \mathbb{P}[O_{1:t}, S_{|t+1\rightarrow} = s_j] = \sum_{i \neq j} \alpha_t(i) A_{ij}, \tag{2.43}$$

i.e., first we compute the forward variables, for each possible duration $d \in \mathcal{D}$, in each state s_j at timestep t, and then we aggregate the statistics in by weighting them w.r.t. the transition probabilities. In order to

[39] Henceforth, for the sake of notation, we will lose the dependency on the model parameters μ.

compute the forward variable $\alpha_t(j)$, the auxiliary variable $\alpha_t^*(j)$ must be initialized as follows

$$\alpha_t^*(j) = \begin{cases} \pi_j, & t = 0 \\ 0, & t < 0 \end{cases} \qquad \forall j \in [k]. \tag{2.44}$$

As before, the likelihood of the observations $\mathbb{P}(O;\mu)$ is computed using eq. (2.29) with the forward variable from eq. (2.42).

The HSMM backward variable is defined as the conditional probability of future observations from time $t+1$ up to the final instant n given that state s_j ends at t, i.e.,

$$\beta_t(j) = \mathbb{P}[O_{t+1:n}|S_{\rightarrow t|} = s_j]. \tag{2.45}$$

Equation (2.45) can be re-written within the EDHMM formulation as [53]

$$\beta_t(j) = \sum_{i=1}^{k} A_{ji} \sum_{d=1}^{\min(d_{max},t)} \beta_{t+d}(i) D_{id} \prod_{\tau=t+1}^{t+d} b_i(\tau) = \sum_{i=1}^{k} A_{ji} \beta_t^*(i), \tag{2.46}$$

where $\beta_t^*(i)$ is an auxiliary variable that represents the conditional probability of future observations from time $t+1$ up to the final instant n given that state s_i starts at $t+1$. That is,

$$\beta_t^*(i) = \mathbb{P}[O_{t+1:n}|S_{|t+1\rightarrow} = s_i] = \sum_{d=1}^{\min(d_{max},t)} \beta_{t+d}(i) D_{id} \prod_{\tau=t+1}^{t+d} b_i(\tau), \tag{2.47}$$

The backward procedure (from $t = n-1, ..., 0$) relies on the initialization of the backward variable as

$$\beta_t(j) = \begin{cases} 1, & t = n \\ 0, & t > n \end{cases} \qquad \forall j \in [k]. \tag{2.48}$$

M-step In order to reestimate the HSMM parameters, the forward and backward variables from eqs. (2.41) and (2.45) are used. The definition of the initial state distribution estimate ($\hat{\pi}_i$) is

$$\hat{\pi}_i = \mathbb{P}[S_{|1\rightarrow} = s_i|O] \underset{Bayes}{=} \frac{\mathbb{P}[S_{|1\rightarrow} = s_i, O]}{\mathbb{P}(O)} = \frac{\mathbb{P}[O|S_{|1\rightarrow} = s_i]\mathbb{P}[S_{|1\rightarrow} = s_i]}{\mathbb{P}(O)} = \frac{\beta_0^*(i)\pi_i}{\mathbb{P}(O)}, \tag{2.49}$$

where we exploited Bayes' rule and eq. (2.47) evaluated at $t = 0$.

The rest of the parameters' estimates can be similarly derived from the definitions. Here, we will solely present the final formulas and skip the derivations. The unnormalized[40] formulas for the transition,

[40] The normalization assures that the distributions sum to 1.

emission and duration distribution estimates are [50]

$$\begin{aligned}
\hat{A}_{ij} &= \sum_{t=1}^{n} \alpha_t(i) A_{ij} \beta_t^*(j) \\
\hat{B}_{iv_j} = \hat{b}_i(O_t = v_j) &= \sum_{t=1}^{n} \mathbf{1}_{(O_t = v_j)} \times \Big(\sum_{\tau<t} \alpha_\tau^*(i)\beta_\tau^*(i) - \sum_{\tau<t} \alpha_\tau(i)\beta_\tau(i) \Big) \\
\hat{D}_{jd} &= \sum_{t=1}^{n} \alpha_t^*(j) D_{jd} \beta_{t+d}(j) \prod_{\tau=t+1}^{t+d} b_j(\tau).
\end{aligned} \tag{2.50}$$

Although HSMMs allow the creation of more expressive models, its FB algorithm is more complex than the one for HMMs due to the incorporation of variable durations.[41] Furthermore, HMMs and HSMMs suffer from an *identifiability problem* [63], which occurs when the estimated parameters can be permuted and we still obtain the same value of model likelihood. This problem can be avoided by considering parametric distributions for the emissions and durations. Families like Poisson, Gaussian or Gamma distributions are the most frequently used ones.

Numerical underflow HSMMs also suffer, like HMMs, of numerical underflow. The two most common solutions to this problem, in the HMM setting, are scaling and the use of logarithms. Even though scaling is much faster, it is not applicable to HSMMs [62]. Within the scope of this dissertation, we adapted the implementation from [64], which avoids numerical underflow by proposing an efficient and practical implementation of the forward–backward algorithm for an EDHMM. The new FB algorithm redefines the forward and backward variables of eqs. (2.41) and (2.45) in terms of posterior probabilities, conditioned on the available observations. For example, the forward variable becomes

$$\alpha_{t|x}(i,d) = \mathbb{P}\left[S_t = s_i, \tau_t = d \mid o_{1:x}\right], \tag{2.51}$$

where $x = t-1, t$ or n, resulting in the predicted, filtered or smoothed forward probabilities, respectively, and τ_t denotes the remaining sojourn time of the current state S_t [64]. From eq. (2.51), we can observe that the forward variable is conditioned on the observations. The trick behind this re-definition is that by using conditional probabilities normalized at each step, we work with relative changes, which give higher resolution than absolute ones. Thus, numerical underflows are less probable to occur in practice.[42]

After training, these models can be used in many applications. For example, one can compute the likelihood and find the most probable hidden state sequence for a given observation sequence, using the Viterbi algorithm with the learnt parameters.

Yet, the most relevant and promising application is perhaps in *time-series forecasting* - a process in which future observations from a time-series are predicted using information from the past and the parameters (or posterior distribution) of the trained model. The accuracy in time-series forecasting depends fully on the trained model, thus, it is important to learn the best parameters so that the patterns of the time-series data are well captured and less uncertainty is reflected in future predictions.

[41] Rabiner [50] stated that, when considering a maximum duration of $d_{max} \approx 25$, the computation is increased by a factor of 300, when compared to the regular HMM forward-backward procedure.

[42] Source code in Matlab available at `http://sist.sysu.edu.cn/~syu/Publications/hsmm_new.m`

Chapter 3

Hyperparameter selection in parametric Markov models

In the present chapter, we address the problem of selecting the hyperparameters in Markov modulated Poisson processes and we discuss three criteria for model order selection: Akaike-Information Criterion (AIC), Bayesian Information Criterion (BIC) and Mixture Minimum Description Length (MMDL). The innovation for this part is the adaptation of the MMDL criterion to HSMMs. Moreover, we present synthetic experiments and results for the task of comparing the three criteria.

3.1 Selection criteria

The importance of the model order selection problem is reflected in the accuracy of the model and in its vulnerability to overfit/underfit. Markov models can also suffer this problem if the selected hyperparameters are too deviated from the ground truth. For instance, if we select a total number of states much larger than the ground truth, then the model has a high granularity (or high variance), which makes it prone to overfit, i.e., even though the accuracy for the training data is high, the model may not generalize well for unseen data. On the other hand, if we select a total number of states much lower than the ground truth, the model will likely underfit, and the accuracy, even in training, is poor.
The study covered in this section regards Markov modulated Poisson processes, which are addressed separately for Hidden Markov models (HMMs) and Hidden-semi Markov models (HSMMs).

The HSMMs' hyperparameters are the total number of states of the semi-Markov chain (k) and the maximum allowed state duration (d_{max}). It is often the case that these hyperparameters are not known *a priori* and must be derived before training. The order estimation problem in HSMMs consists precisely in estimating the optimal total number of states (k^*) and maximum allowed state duration (${d_{max}}^*$). It is a multivariate optimization problem where both k and d_{max} are optimized w.r.t. the same criterion.
One greedy approach would be to try every possible combination of k and d_{max} and assign the one with the highest model likelihood (eq. (2.29)). However, this approach is extremely inefficient as the number

of combinations grows with $k \times d_{max}$. Ideally, it would be better to guarantee that each state is connected to the set of possible durations ($\mathcal{D} = \{1, 2, ..., d_{max}\}$) in a one-by-one relationship. This would allow the problem to be simplified to a univariate optimization problem, where the focus is only on the selection of k^*.

If we consider a model with high statistical complexity, for example, a model with non-parametric state duration distributions, then this one-by-one relationship is not present, because each state needs d_{max} "parameters" to specify its duration. Likewise, we can consider a model with lower statistical complexity by using parametric state duration distributions, where, in analogy with the multinomial case, each state has a mixture of parametric distributions for its duration probability (e.g. a mixture of geometric distributions). Nonetheless, the problem remains since each state has multiple parameters (as many as the number of components of the mixture of that state) to specify its duration. The next section provides a solution to this problem.

Equivalence between HSMMs Consider a HSMM with three states, such that its state durations follow a parametric mixture of some distribution belonging to the exponential family (e.g. Poisson, Gamma, Geometric or Gaussian). The first state of this HSMM has a three-component mixture duration density, while the second state has a two-component mixture duration density and the last state has a single-component duration density. This HSMM is, in fact, equivalent to another one with six states characterized by single duration densities. A formal proof of this model equivalence can be found in Appendix C, where we considered geometric state duration distributions.

Supported by this equivalence, the multivariate order estimation problem in HSMMs is simplified through the use of the equivalent model with augmented states but only one geometric distribution per state. Consequently, each state has one and only one parameter to specify its duration (the parameter of the geometric distribution), therefore guaranteeing the one-by-one relationship required.

This type of model equivalence can also be found for Gaussian mixtures [65] [1], and for HMMs with Gaussian mixture emission densities [66].

In the equivalent model, d_{max} is the same for all states, even though in reality this may not be true. However, since the forward-backward algorithm for HSMMs (detailed in Section 2.2.2.5) considers a general d_{max}, common to all states, in order to address this issue, the forward-backward algorithm must be completely redefined. Finding a solution to this problem is a possible line of further work.

To sum up, we use parametric state duration distributions to simplify the optimization problem, where each state is associated with one and only one parameter. Then, d_{max} is empirically estimated using the parameters of the state duration distributions. Out of curiosity, if we use geometric distributions[2] for the durations and Poisson distributions for the emissions, we will end up with a setting similar to the one used to model queues [67]. For instance, problems such as internet package flow or particle decay require modeling the arrivals of the packages (or particles). The usual way is to model them as a Poisson

[1] In Gaussian mixtures, a state composed by a mixture of two Gaussians is equivalent (in terms of the observation likelihood) to two states with only one Gaussian distribution per state.

[2] Or any distribution from the exponential family.

process and the waiting time in the queue (or the time between the decay of one particle and the next) by an exponential distribution.

Empirical estimation of d_{max} In parametric state duration distributions, there is a correspondence between the parameters of the distribution (θ_i for state s_i) and the durations. For example, in geometric distributions, a lower parameter is associated with higher durations.
Therefore, d_{max} can be estimated through the following steps: **1)** find the parameter θ_i^*, of state s_i, associated with the highest duration between all states[3]; **2)** find the (first) duration for which the state duration PMF with parameter found in step **1)**, is smaller (or equal) than $\epsilon \approx 0.001$. That is,

$$\mathbb{P}_i(D > d_{max}) = 1 - \mathbb{P}_i(D \leq d_{max}) = \epsilon, \qquad \forall i \in [k], \tag{3.1}$$

where $\mathbb{P}_i(D \leq d_{max})$ is the CDF[4] of the state duration PMF associated with state s_i with parameter θ_i^* found in step **1)** and D is a random variable that takes values in the duration set $\mathcal{D} = \{1, 2, ..., d_{max}\}$. After setting the value of ϵ, eq. (3.1) is analytically solved for d_{max}.

Example 4 If we consider geometric state duration distributions, then the PMF of state s_i is given by

$$\mathbb{P}_i(D = d) = (1 - \theta_i)^{d-1}\theta_i, \qquad \forall d \in \mathcal{D}, \tag{3.2}$$

where the parameter $\theta_i \in]0, 1]$ represents the success probability of the geometric distribution. The corresponding CDF is easily obtained from eq. (3.2) as

$$\mathbb{P}_i(D \leq d) = 1 - (1 - \theta_i)^d. \tag{3.3}$$

According to eqs. (3.1) and (3.3), the maximum state duration is

$$\begin{aligned} &1 - \mathbb{P}_i(D \leq d_{max}) = \epsilon \\ &\qquad \iff d_{max} = \lfloor log_{(1-\theta_i)}(\epsilon) \rfloor \Big|_{\theta_i = \theta_i^*} \end{aligned} \tag{3.4}$$

After determining d_{max}, the HSMM state duration probability matrix $\boldsymbol{D} \in \mathbb{R}^{k \times d_{max}}$ can be filled with values from each state PMF evaluated at the corresponding durations $d \in \mathcal{D}$.
Regarding HMMs, the order estimation problem was deeply studied and several state-of-art solutions are available ranging from cross-validation [68] and heuristic methods to Bayesian approaches [69]. Heuristic methods are less computationally expensive and the accuracy is not substantially compromised. Following that mindset, the key idea is to adapt the heuristic solutions from HMMs to HSMMs, where there is an increased complexity due to the incorporation of variable durations. In the following sections, we discuss the adaptation of three heuristic selection criteria for HSMMs: Akaike-Information Criterion (AIC), Bayesian Inference Criterion (BIC) and Mixture Minimum Description Length (MMDL).

[3] For geometric distributions, it corresponds to the lowest parameter.
[4] Cumulative distribution function.

3.1.1 Akaike-information criterion

The model order selection criterion should express how well the MLE of the model parameters (obtained in the M-step of the EM algorithm), together with the selected total number of states, describe the observations. According to this definition, it would be reasonable to consider the log-likelihood of the observations as one possible criterion. However, this criterion is not sufficient because it does not penalize the complexity of the model. An HMM/HSMM with a larger number of states is, using this criterion, always preferable given its greater expressiveness. Following this reasoning, in 1974, Hirotugu Akaike [70] proposed the Akaike-Information Criterion (AIC), generally defined as

$$\mathsf{AIC}(k) = \log \mathbb{P}[O; \hat{\mu}_k] - \frac{N_k}{2}, \tag{3.5}$$

where the first term is the log-likelihood, hereinafter referred to as LL, of the observations given the MLE parameters ($\hat{\mu}_k$), for a model with k states. In particular, the criteria introduced in this chapter are used for HMMs with parameters $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}\}$ (see Section 2.2.1.3) and for HSMMs with parameters $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}, \boldsymbol{D}\}$ (see Section 2.2.1.4). The second term in eq. (3.5) penalizes the model complexity through the total number of free parameters, N_k. The optimal number of states is the one that maximizes eq. (3.5). That is,

$$k^*_{\mathsf{AIC}} = \arg\max_k \mathsf{AIC}(k). \tag{3.6}$$

For HMMs, the number of free parameters (N_k) can be decomposed in ${N_k} = {N_k}^{\boldsymbol{A}} + {N_k}^{\boldsymbol{\pi}} + {N_k}^{\boldsymbol{B}}$. The number of free parameters for the transition matrix is ${N_k}^{\boldsymbol{A}} = k(k-1)$, since it is a stochastic matrix with k^2 elements. However, one element in every row is dependent on the others because they need to sum to 1, i.e., $\sum_{j=1}^{k} A_{ij} = 1$. For the initial state distribution, ${N_k}^{\boldsymbol{\pi}} = k - 1$, and, finally, for the emission probability, if we consider a parametric Poisson distribution, the total number of free parameters is k, one per state.[5] To sum up, for HMMs, the general AIC criterion defined in eq. (3.5) is extended to[6]

$$\mathsf{AIC}_{\mathsf{HMM}}(k) = \mathsf{LL} - \frac{k^2 + k}{2}. \tag{3.7}$$

For HSMMs, the AIC criterion is equal to the one in eq. (3.7). The total number of free parameters (N_k) is the same with the following exceptions: **1)** a added term ${N_k}^{\boldsymbol{D}}$ which accounts for the state duration density $\boldsymbol{D}$. If we consider it as a parametric distribution, the total number of free parameters is k, one per state.[7]; **2)** The number of free parameters for the transition matrix is ${N_k}^{\boldsymbol{A}} = k(k-2)$, since we are considering that the diagonal entries of the HSMM transition matrix are zero. The differences end up canceling out, giving the same result as for HMMs (eq. (3.7)). The results of the AIC criterion show that the penalization is not substantial for most applications. The next criterion addresses this issue.

[5] For non-parametric state emission distributions, the number of free parameters would be $k(m-1)$, for m observable symbols.
[6] We dropped all terms that do not depend on the number of states k.
[7] For non-parametric state duration distributions, the number of free parameters would be $k(d_{max} - 1)$.

3.1.2 Bayesian information criterion

In 1978, Schwarz [71] proposed the Bayesian Information Criterion (BIC), given by

$$\text{BIC}(k) = \log \mathbb{P}[O; \hat{\mu}_k] - \frac{N_k}{2} \log(n), \tag{3.8}$$

where the second term penalizes the model complexity through the total number of free parameters (N_k) weighted by the total number of observations (n).

From inspection of eqs. (3.5) and (3.8), we can conclude that the BIC and AIC criteria are very similar. However, as long as there are more than e observations[8], the penalization in the BIC criterion is higher than the penalization in the AIC criterion. By penalizing more, the BIC criterion is more realistic, avoiding model overfitting issues. However, contrary to what is implied in eq. (3.8), each model parameter is affected differently by the observation sequence: some by the whole observations (n), others by just some parts. The next criterion addresses this issue.

3.1.3 Mixture minimum description length

In 1978, Jorma Rissanen [72] proposed the Minimum Description Length (MDL) model order selection criterion. Then, in 1999, Figueiredo et al. [65] adapted the MDL criterion for mixture models, resulting in the Mixture Minimum Description Length criterion (MMDL).

In mixture models, the model complexity penalty term does not need to consider the whole data ($log(n)$), given that each component of the mixture is not estimated from all observations but only from those that were, in fact, generated by that component. Hence, the term $log(n)$ is replaced by a quantity that measures how much data was generated by a given component. This minimizes the length of description needed for each parameter. For mixture models, this quantity is easily obtained as $log(nc_j)$, where c_j is the probability of the j^{th} mixture component. In 2003, Bicego et al. [66] adapted the MMDL criterion from mixture models to HMMs. We follow the same reasoning to adapt it to HSMMs.

MMDL for HMMs In HMMs, both the transition matrix $\boldsymbol{A}$ and the initial state distribution $\boldsymbol{\pi}$ are estimated from the entire observed data samples n. Therefore, $\boldsymbol{A}$ and $\boldsymbol{\pi}$ are still weighted by the standard $log(n)$ term from eq. (3.8). On the contrary, the emission probability $\boldsymbol{B}$ is estimated using only the data samples generated by the corresponding state.[9] The quantity that specifies the state probabilities[10] is given by the stationary probability distribution $\boldsymbol{p}_\infty = [p_\infty(1), ..., p_\infty(k)]$, which represents the "average" occupation of each (steady) state after the Markov chain has reached its equilibrium. In practice, one could consider that, by the time the observations start, the Markov chain has already reached its stationary state. In this scenario, $\boldsymbol{p}_\infty$ is simply the initial state distribution $\boldsymbol{\pi}$ [73]. However, without making this assumption, the HMM stationary probability distribution ($\boldsymbol{p}_\infty$) can be obtained as the left eigenvector of the transition matrix $\boldsymbol{A}$ associated with its unit eigenvalue [74]. In Bicego et al. [66], the MMDL criterion for a HMM

[8] Since $log(e) = 1$.

[9] The states are analogous to the components of the mixture in mixture models.

[10] The state probabilities correspond to the component probabilities (c_j) in mixture models.

with k states is

$$\text{MMDL}_{\text{HMM}}(k) = \text{LL} - \frac{N_k^{\boldsymbol{A}} + N_k^{\boldsymbol{\pi}}}{2}\log(n) - \frac{N_1^{\boldsymbol{B}}}{2}\sum_{i=1}^{k}\log(np_\infty(i)), \tag{3.9}$$

where $N_1^{\boldsymbol{B}}$ is the number of free parameters of the emission density of a HMM with just one state ($k = 1$). The second quantity is weighted by $\sum_{i=1}^{k}\log(np_\infty(i))/2$, where $np_\infty(i)$ is designated the "effective sampling size" of state s_i, which measures the percentage of the total data n presumably generated by steady state s_i.

MMDL for HSMMs Few changes are required to adapt the MMDL criterion in eq. (3.9) from HMMs to HSMMs. If we consider the HSMM variant of an explicit-duration HMM, the required adaptations include: **1)** introduction of the state duration distribution $\boldsymbol{D}$ in eq. (3.9); **2)** definition of the number of free parameters (N_k) and **3)** definition of the HSMM stationary probability distribution.
Regarding the first adaptation, the state duration distribution $\boldsymbol{D}$ is introduced in one of the penalty terms of eq. (3.9). Since the explicit-duration HMM considers that the duration depends only on the corresponding state, $\boldsymbol{D}$ is actually estimated from the observations that were presumably generated by that state, and not from all the observed data. Therefore, the number of free parameters of $\boldsymbol{D}$ will appear in the second quantity of eq. (3.9), together with the number of free parameters of $\boldsymbol{B}$. Hence, for a HSMM with k states, the MMDL criterion is given by

$$\text{MMDL}_{\text{HSMM}}(k) = \text{LL} - \frac{{N_k}^{\boldsymbol{A}} + {N_k}^{\boldsymbol{\pi}}}{2}\log(n) - \frac{{N_1}^{\boldsymbol{B}} + {N_1}^{\boldsymbol{D}}}{2}\sum_{i=1}^{k}\log(np_\infty(i)). \tag{3.10}$$

For the second requirement, the number of free parameters for $\boldsymbol{A}$ and $\boldsymbol{\pi}$ are $N_k^{\boldsymbol{A}} = k(k-2)$ and $N_k^{\boldsymbol{\pi}} = k-1$. In what the emissions and durations are concerned, if we use parametric distributions, then there is only one free parameter in a HSMM with just one state, i.e., $N_1^{\boldsymbol{B}} = N_1^{\boldsymbol{D}} = 1$.[11] Replacing the free parameters in eq. (3.10) and dropping the terms that do not depend on k, yields

$$\text{MMDL}_{\text{HSMM}}(k) = \text{LL} - \frac{k^2-k}{2}\log(n) - \sum_{i=1}^{k} log(np_\infty(i)). \tag{3.11}$$

In eq. (3.11), the stationary distribution $\boldsymbol{p}_\infty$ for an explicit-duration HMM still needs to be specified.

3.1.3.1 HSMM stationary probability distribution

The stationary probability distribution for semi-Markov chains is not the same as for regular Markov chains. However, it can be found empirically. In Barbu et al. [73], the authors proposed the following estimator

$$p_\infty(i) = \frac{\hat{v}_i(n)\hat{m}_i(n)}{\sum_{i=1}^{k}\hat{v}_i(n)\hat{m}_i(n)}, \quad \hat{v}_i(n) = \frac{N_i(n)}{N(n)}, \quad \forall i \in [k], \tag{3.12}$$

[11] For non-parametric distributions, the number of free parameters for the state emission density would be $(m-1)$ and for the state duration density would be $(d_{max}-1)$.

where $\hat{v}_i(n)$ is an estimator for the stationary probability distribution of the embedded[12] Markov chain. It is obtained, as shown in eq. (3.12), as the ratio of the number of visits of the embedded Markov chain to state s_i up to time $t = n$ ($N_i(n)$) and number of jumps of the embedded Markov chain in the time interval $(0, n]$ ($N(n)$), both available from the estimated state sequence after running the *Viterbi* algorithm.
In eq. (3.12), $\hat{m}_i(n)$ is an estimator of the mean sojourn time in state s_i, considering all observations up to time n. In Barbu et al. [73], they define it as

$$\hat{m}_i(n) = \frac{\sum_{l=1}^{N_i(n)} d_{il}}{N_i(n)}, \qquad \forall i \in [k], \tag{3.13}$$

where d_{il} is the duration of the l^{th} visit of the embedded Markov chain to state s_i.
In practice, $\hat{m}_i(n)$ is obtained by averaging the EM estimate for the state duration probability matrix ($\hat{\boldsymbol{D}}$), weighted by the corresponding durations. For the sake of interpretation, we replace eq. (3.13) in eq. (3.12), which yields the final estimator for the stationary probability distribution of a semi-Markov chain

$$p_\infty(i) = \frac{\sum_{l=1}^{N_i(n)} d_{il}}{\sum_{i=1}^{k} \sum_{l=1}^{N_i(n)} d_{il}} \approx \frac{\sum_{l=1}^{N_i(n)} d_{il}}{n}. \tag{3.14}$$

This represents the ratio between the total time spent in state s_i and the length of the observation sequence. Thus, as desired, the stationary probability distribution for semi-Markov chains ($\boldsymbol{p}_\infty$), given by eq. (3.12), is a measure of the "average" state occupation. Moreover, the approximation in eq. (3.14) is valid for a small maximum state duration d_{max}, where it is reasonable to assume that the truncation of the semi-Markov chain at time $t = n$ corresponds to the exact time of a state transition.

After computing $\boldsymbol{p}_\infty$ for all states, the MMDL criterion in eq. (3.11) is fully defined. However, for any of the model criteria in eqs. (3.7) to (3.11), in order to obtain the optimal number of states k^*, a greedy approach of testing multiple values for k and selecting the one with the highest criterion value is not desirable. On the one hand, this approach is highly dependent on the initialization, which requires training the model, with the same k, multiple times to obtain a reliable result for the log-likelihood, the parameter estimates and, consequently, the chosen criterion. On the other hand, it is a slow and inefficient approach. In the next section, we describe a strategy that counters these constraints.

3.1.3.2 Sequential pruning strategy

In 2003, Bicego et al. [66] proposed a sequential pruning strategy for the selection of the number of states in HMMs, which can also be used for HSMMs. The proposed method consists of a "decreasing" learning strategy wrapped around the HMM training phase. The idea is to start with a large number of states (k_{max})[13] and run the EM algorithm. After convergence, the optimal HMM estimated parameters ($\hat{\mu}_k$) are found. These are used to compute the value of the chosen model order selection criterion (denoted as C_k), e.g., the MMDL criterion for HSMMs in eq. (3.11). The least probable state (denoted

[12] The embedded Markov chain of a semi-Markov chain with state sequence $\{4, 4, 1, 1, 1, 2, 3, 3, 3\}$ is $\{4, 1, 2, 3\}$ with the corresponding jump times $\{2, 5, 6\}$.
[13] The minimum and maximum allowed number of states (k_{min} and k_{max}) are user-defined arguments.

as LPS) is found using the stationary probability distribution (p_∞). Then, the LPS is pruned by removing (and normalizing[14]) the corresponding rows/columns from the estimated parameters. Finally, we run EM again with the estimated parameters from the last iteration without the state that was removed. This process, shown in Algorithm 2 found in Appendix D, repeats itself until the minimum number of states (k_{min}) is reached.

The reason why this strategy is called "decreasing" learning is because we start with a large number of states and, in each iteration, we prune the least probable state of the model, resulting in a reduced model which is used as the initial configuration for the next EM training, with one less state. This strategy guarantees a lower sensitivity to the initialization of the EM algorithm given that in each iteration we initialize in a better situation after removing the LPS.

In Sections 3.2 and 4.4.3, we show the results of the experiments with synthetic and real data, respectively, for the task of comparing the criteria addressed in this section, along with their pruning version. After selecting the optimal hyperparameters, using one of the supramentioned strategies, the chosen model is trained from the observations.

3.2 Synthetic experiments

In this work, we adapted the code from [64] to efficiently implement EDHMMs - the simplest version of HSMMs, in which a state transition does not depend on the duration of the previous state - and we considered Poisson emissions and geometric duration distributions. Besides that, the order estimation strategy was implemented and wrapped around the estimation algorithm.[15]

In the present experiment, we compare the AIC, BIC and MMDL order selection criteria, described in Sections 3.1.1, 3.1.2, and 3.1.3, respectively. Furthermore, we compare them in their standard and pruning versions. The standard version is similar to the pruning version, but instead of initializing the parameters of the EM with the result of the last iteration with the less probable state pruned, we simply initialize them at random with one less state, according to Algorithm 3 found in Appendix D.

The synthetic data consists of one sequence ($N = 1$) of 1000 observations ($n = 1000$), sampled from a HSMM with $k = 5$ states. To implement the model equivalence described in Section 3.1, we modeled the duration distribution of each state parametrically through geometric distributions with parameters $\boldsymbol{\theta} = [\theta_1, ..., \theta_k]^\intercal$, where $\theta_i \in]0, 1]$ is the parameter of the geometric duration distribution of state s_i. The

[14] If normalization is not possible (this problem can arise when considering a large amount of states), we set distributions to uniform i

[15] The source code is available at `https://github.com/filiparente/hsmm_mmdl`.

rest of the parameters $\mu = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{\lambda}\}$ are obtained at random (except $\boldsymbol{\lambda}$), according to eq. (3.15).

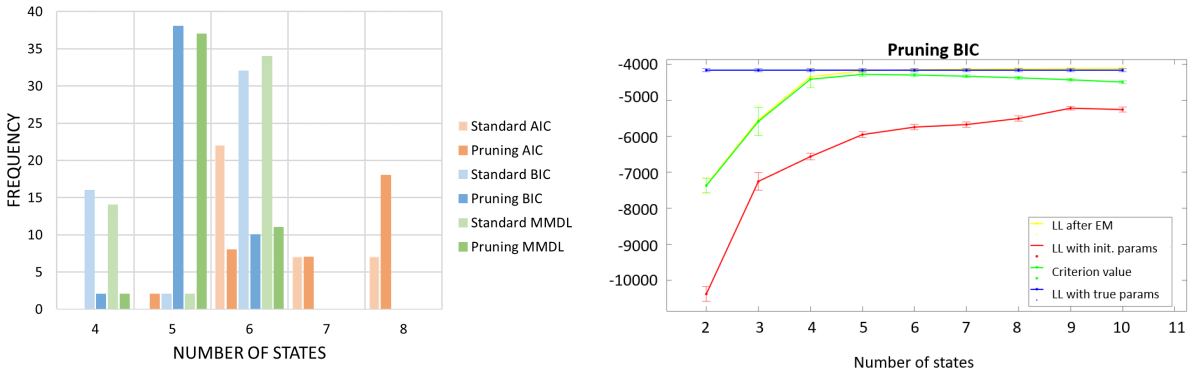
$$\boldsymbol{\pi} = \begin{bmatrix} 0.0062 \\ 0.0926 \\ 0.1914 \\ 0.2593 \\ 0.4506 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 15 \\ 36 \\ 49 \\ 105 \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} 0 & 0.5714 & 0.1364 & 0.2727 & 0.0195 \\ 0.2718 & 0 & 0.2953 & 0.3255 & 0.1074 \\ 0.3043 & 0.0435 & 0 & 0.4174 & 0.2348 \\ 0.3511 & 0.2801 & 0.2660 & 0 & 0.1028 \\ 0.0198 & 0.2673 & 0.2178 & 0.4950 & 0 \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} 0.7572 \\ 0.8123 \\ 0.7970 \\ 0.8077 \\ 0.8293 \end{bmatrix}. \tag{3.15}$$

The maximum allowed state duration (d_{max}) can be computed from eq. (3.4), using the minimum value of $\boldsymbol{\theta}$ in eq. (3.15) as $\theta_i^* = 0.7572$, and $\epsilon = 0.001$, yielding a value of $d_{max} = 4$.

We performed 50 iterations, where we sampled independent observation and state sequences from the setup of eq. (3.15). Subsequently, we initialized, at random, the parameters μ used as starting point of the EM algorithm. The initialization is the same for all three criteria. Then, we ran the standard and pruning versions of BIC, AIC and MMDL and we obtained the optimal number of states (k^*) and optimal model parameters ($\mu_{k^*}^*$) chosen by each criterion, according to lines 9 and 10 of Algorithm 3, in Appendix D. Since the true number of states is $k = 5$, we considered $k_{max} = 10$ and $k_{min} = 2$ for pruning. The maximum number of iterations for the EM algorithm was set to 100.

In Fig. 3.1(a), we show a histogram of the number of times each state was chosen as optimal state for each criterion. We observe that the pruning version of the BIC and MMDL criteria are the most accurate ones, since they hit most often the right total number of states ($k = 5$). We also note that the only criterion to estimate 7 and 8 states is the AIC (in the pruning and standard versions), which confirms that the penalization on the model complexity is lower in this criterion when compared to the others.

In Fig. 3.1(b), we present a comparison of the evolution of different log-likelihoods with the number of



(a) (b)

Figure 3.1: (a) Histogram of the chosen optimal number of states per criterion, with both the standard and pruned versions. The BIC is shown in blue, the AIC in orange and the MMDL in green; (b) Comparison of the evolution of the log-likelihood with the initialized parameters (in red), the log-likelihood with the parameter estimates after running EM (in yellow), the criterion value (in green) and the log-likelihood with the original parameters from eq. (3.15) (in blue). The dot and error bars represent the mean and standard deviation of the 50 runs. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

states, for the pruning version of BIC. We observe that the parameter estimates, when compared to the

initial parameters, provide a much better training log-likelihood (yellow line compared to red line)[16], as expected. We also observe the point where the model complexity starts to deviate the criterion value (in green) from the log-likelihood after the EM (in yellow), at around $k = 5$. We note that the criterion value is always below the real log-likelihood, whereas the log-likelihood after EM is slightly above the real one from $k = 6$ onwards, which indicates that the parameter estimates are better than the true ones in explaining the observation sequences.[17]

In Table 3.1, we present the mean and standard deviation (for the 50 runs) of the chosen optimal number of states, *hit rate* and number of EM iterations at convergence, for each criterion. The hit rate measures the percentage of correct state assignments between the estimated and true state sequences. That is,

$$\text{Hit rate} = 1 - \frac{||S - \hat{S}||_0}{n \times N}, \tag{3.16}$$

where $||\cdot||_0$ is the ℓ_0-norm[18], S is the true state sequence and $\hat{S}$ is the predicted state sequence, output of the Viterbi algorithm (see Section 2.2.2.4).

Table 3.1: Estimation results for all criteria: chosen optimal number of states, hit rate and number of EM iterations at convergence, round to 4 decimal places. Mean and standard deviation from the 50 runs.

Criteria	k^*	Hit rate	Nº EM iterations
Standard BIC	5.3200 ± 0.9355	0.8087 ± 0.1221	85.3200 ± 22.1548
Standard AIC	7.3800 ± 1.4831	0.8294 ± 0.0801	99.4800 ± 3.6770
Standard MMDL	5.4000 ± 0.9035	0.8187 ± 0.1195	87.5000 ± 20.7318
Pruning BIC	$\mathbf{5.1600 \pm 0.4677}$	$\mathbf{0.8992 \pm 0.0553}$	$\mathbf{65.5000 \pm 37.1666}$
Pruning AIC	7.7800 ± 1.2664	0.7698 ± 0.0647	88.1400 ± 28.4174
Pruning MMDL	$\mathbf{5.1800 \pm 0.4819}$	$\mathbf{0.8987 \pm 0.0556}$	$\mathbf{67.3600 \pm 36.5004}$

From inspection of the table, we conclude that the pruning version of the BIC and MMDL are the heuristics that perform best, with a average number of states closer to the real one, a high hit rate and with the lowest number of iterations due to the pruning strategy. We also note that the number of EM iterations for the AIC criterion, from the standard to its pruning version, do not differ as much as for the BIC and MMDL criteria, because the complexity is increased by having more states. However, for the other criteria, the pruning saves more than 20 iterations while achieving a better hit rate.

Finally, regarding the parameter estimates, in Table 3.2 we present the mean and standard deviation of the parameter estimates of the Poisson rates, computed only at the runs where the chosen optimal number of states corresponds to the mode of k. For example, the mode of k for the standard AIC is 6, since it is the most frequent value in Fig. 3.1.[19]

[16] Note that the best log-likelihood is the one closest to zero, since the maximum of the logarithmic function for input probabilities (between 0 and 1) is 0.

[17] Note that these results are the mean and standard deviation for the 50 runs, with different state and observation sequences, but sampled by the same setup in eq. (3.15).

[18] The ℓ_0-norm corresponds to the number of non-zero entries.

[19] This way, we can compare Poisson rates with the same (and most often) chosen k.

Table 3.2: EM estimates of the Poisson rates for each criteria, averaged at the runs where the chosen number of states equals the mode.

Criteria	k_{mode}	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$	$\hat{\lambda}_4$
Std. BIC	6	2.3257 ± 0.0759	15.0777 ± 0.2266	35.8849 ± 0.9667	49.1458 ± 0.8269
Std. AIC	6	2.3243 ± 0.0807	15.1670 ± 0.3141	36.3674 ± 1.0199	49.1873 ± 0.9935
Std. MMDL	6	2.3213 ± 0.0759	15.0772 ± 0.2425	35.9201 ± 0.9483	49.1319 ± 0.8057
Pr. BIC	5	2.3310 ± 0.0961	15.1169 ± 0.2822	36.4999 ± 1.2348	49.0948 ± 1.0829
Pr. AIC	8	2.3400 ± 0.1259	15.1011 ± 0.3089	33.8049 ± 1.8010	38.7381 ± 2.2263
Pr. MMDL	5	2.3339 ± 0.0957	15.1223 ± 0.2842	36.4917 ± 1.0271	49.1569 ± 1.0271

Criteria	k_{mode}	$\hat{\lambda}_5$	$\hat{\lambda}_6$	$\hat{\lambda}_7$	$\hat{\lambda}_8$
Std. BIC	6	102.9748 ± 1.7507	108.6210 ± 3.8192	—	—
Std. AIC	6	102.4908 ± 2.1127	109.6341 ± 4.4743	—	—
Std. MMDL	6	103.0019 ± 1.7010	108.8965 ± 3.9179	—	—
Pr. BIC	5	105.1517 ± 1.1652	—	—	—
Pr. AIC	8	47.8050 ± 1.4308	50.6237 ± 1.4288	100.4058 ± 12.3623	107.5472 ± 3.1176
Pr. MMDL	5	105.1510 ± 1.1813	—	—	—

We observe that the parameter estimates for $k = 5$ are pretty accurate when compared to the real ones from eq. (3.15), with no relevant distinction between the BIC and MMDL criteria. We also note that when a number of states higher than the real one ($k = 5$) is chosen, the additional parameters cover the high levels (49 or 105), which makes sense given that the variance of the Poisson distribution is equal to its parameter. Therefore, high parameters have higher variance and, in theory, need more states to represent them. This phenomena can also be explained by the transition matrix in eq. (3.15), which contains high transition probabilities, for example, to move to state s_4. As an example of the parameter estimates of the geometric state duration distributions, we show the results for the Pruning MMDL criterion with $k_{\text{mode}} = 5$:

$$\hat{\boldsymbol{\theta}} = \begin{bmatrix} 0.7549 \\ 0.8140 \\ 0.7978 \\ 0.8331 \\ 0.8311 \end{bmatrix} \pm \begin{bmatrix} 0.0280 \\ 0.0238 \\ 0.0607 \\ 0.0617 \\ 0.0326 \end{bmatrix}, \tag{3.17}$$

$$\hat{\boldsymbol{\lambda}} = \begin{bmatrix} 2.3339 \\ 15.1223 \\ 36.4917 \\ 49.1569 \\ 105.1510 \end{bmatrix} \pm \begin{bmatrix} 0.0957 \\ 0.2842 \\ 1.0271 \\ 1.0271 \\ 1.1813 \end{bmatrix}, \tag{3.18}$$

which is an accurate estimate of the real parameter from eq. (3.15).

From the present experiment, we conclude that the pruning strategy is preferred over the standard version of the same criteria, and that the best criteria are the BIC and MMDL, with similar accuracy in the parameter and state sequence estimation and in the choice of the total number of states.

Chapter 4

Time-series forecasting

Time-series forecasting has gained a lot of attention over the past decades in a wide range of domains that include medicine [2], finance [3] and weather [5]. For instance, in finance, predicting stock prices is crucial for the economic sustainability of many companies [3]. Another business that is greatly affected by time-series forecasting is retail [7], in which reliable demand forecasts allow retailers to make precise adjustments to product inventories. These adjustments are crucial for retailers' success, as they avoid placing too much or too little of a product in a store.

We begin by presenting one example of time-series forecasting applied to a publicly available Kaggle dataset that consists of 5 years of daily sales data.[1] Figure 4.1[2] illustrates the evolution of the real and predicted observations for training (historical) data, as well as 3 months of forecasted data in the future. We verify that the weekly and seasonal demand patterns captured during training are reflected

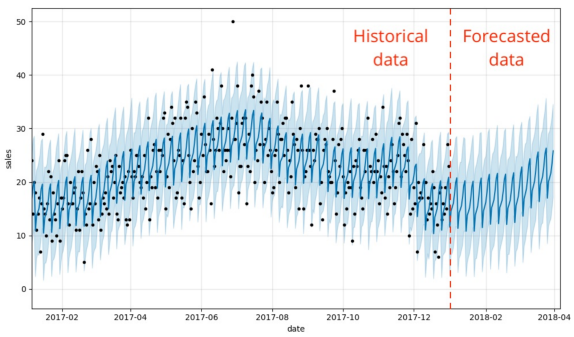


Figure 4.1: Time-series forecasting of daily sales data. The black dots represent true observations, while the darker blue line represents predictions and the lighter blue band represents a 95% uncertainty interval. A forecast for 3 months in the future is presented.[3]

in the future. In this chapter, we address time-series forecasting using Markov models and LSTMs. We begin by exploring one-step-ahead prediction in HMMs and HSMM. Then, we propose a novel context aware HSMM (the *HSMM-LR*) that conditions the state transitions on external features through a logistic regression approach. Finally, we present several synthetic experiments to assess the estimation

[1] For 50 individual items across 10 different stores. Here we present a single item of a store.
[2] The (almost) 5 years of training data are not fully depicted in Fig. 4.1, so that the weekly and seasonal patterns are visible.
[3] Source: `https://databricks.com/blog/2020/01/27/time-series-forecasting-prophet-spark.html`

accuracy in HSMMs and we apply all the Markov models considered (HMM, HSMM and HSMM-LR) and the LSTM to a real time-series forecasting problem using a social media dataset.

4.1 One-step-ahead prediction

One-step-ahead prediction corresponds to predicting one observation at a time, using all past observations up to that instant. In the next two subsections we detail the equations for one-step-ahead prediction separately for HMMs and HSMMs, where we considered Poisson emissions in both cases.

4.1.1 HMMs

Recall the state posterior $\gamma_t(i)$ defined in eq. (2.36). Considering that the last observation available is at time t, the first prediction is for the observation at $t+1$, which is

$$\hat{o}_{t+1} = \sum_{i=1}^{k} \mathbb{P}[S_{t+1} = s_i | o_{1:t}] \hat{c}_i = \sum_{i=1}^{k} \gamma_{t+1|t}(i) \hat{c}_i, \tag{4.1}$$

where $\hat{c}_i$ is a random sample drawn from the Poisson distribution associated with state s_i with parameter $\hat{\lambda}_i$, estimated during training. It is easy to obtain $\gamma_{t+1|t}(i)$ from $\gamma_t(i)$ (the state posteriors computed by the forward-backward algorithm in Section 2.2.2.4) using the definition of state transitions in eq. (2.7)

$$\gamma_{t+1|t}(j) = \mathbb{P}[S_{t+1} = s_j | o_{1:t}] = \frac{\sum_{i=1}^{k} \hat{A}_{ij} \gamma_t(i)}{\sum_{j=1}^{k} \sum_{i=1}^{k} \hat{A}_{ij} \gamma_t(i)}, \forall j \in [k], \tag{4.2}$$

where the denominator is simply a normalization term.
To sum up, at the end of training (time $t = T$), we have access to the state posteriors $\gamma_T(i), \forall i \in [k]$. Thus, we can predict the next count $\hat{o}_{T+1}$ using eq. (4.1) and (4.2). However, the state posteriors need to be adjusted for further predictions, so that the prediction errors do not accumulate with time [50]. This can be achieved by conditioning on the previous (real) observation, as follows

$$\gamma_{t+1|t+1}(i) = \mathbb{P}[S_{t+1} = s_i | o_{1:t+1}] \underset{Bayes}{=} \frac{\mathbb{P}[S_{t+1} = s_i, o_{1:t+1}]}{\mathbb{P}(o_{1:t+1})} = \frac{\mathbb{P}[S_{t+1} = s_i, o_{1:t+1}]}{\sum_{i=1}^{k} \mathbb{P}[S_{t+1} = s_i, o_{1:t+1}]}. \tag{4.3}$$

Given that we are following the assumption that the observations are conditionally independent given the state, we can further simplify the numerator of eq. (4.3) as

$$\begin{aligned} \mathbb{P}[S_{t+1} = s_i, o_{1:t+1}] &= \mathbb{P}[S_{t+1} = s_i | o_{1:t}] \mathbb{P}(o_{1:t}) \mathbb{P}[o_{t+1} | S_{t+1} = s_i] \\ &= \gamma_{t+1|t}(i) \mathbb{P}(o_{1:t}) \mathcal{P}(\hat{\lambda}_i, o_{t+1}), \end{aligned} \tag{4.4}$$

where $\mathcal{P}$ represents a Poisson distribution. Replacing eq. (4.4) in (4.3), yields

$$\gamma_{t+1|t+1}(i) = \frac{\gamma_{t+1|t}(i) \mathcal{P}(\hat{\lambda}_i, o_{t+1})}{\sum_{i=1}^{k} \gamma_{t+1|t}(i) \mathcal{P}(\hat{\lambda}_i, o_{t+1})}. \tag{4.5}$$

Thus, the one-step-ahead prediction algorithm alternates between filtering (eq. (4.2)) and conditioning (eq. (4.5)) steps. One can think of this state posterior as an "adjustment" of $\gamma_{t+1|t}(\cdot)$, by introducing the real observation o_{t+1}. It will be used as $\gamma_t(\cdot)$ in eq. (4.2) for the next prediction at time $T+2$. This process continues until all required predictions are made (e.g. until the end of the test set).

4.1.2 HSMMs

The one-step-ahead prediction in HSMMs is different than the one from HMMs due to the incorporation of variable durations. In this section, we consider the efficient forward-backward implementation from [64], where the probability for the one-step-ahead prediction of the observation o_t is given by [64]

$$\mathbb{P}(o_t|o_{1:t-1}) = \sum_{i,d} \alpha_{t|t-1}(i,d)\hat{b}_i(t). \tag{4.6}$$

In eq. (4.6), $\alpha_{t|t-1}(i,d)$ is the forward variable from eq. (2.51)[4], and $\hat{b}_i(t)$ is the Poisson PMF of eq. (2.9), with parameter $\hat{\lambda}_i$, estimated from the EM algorithm. By marginalizing out the durations d in eq. (4.6), we obtain [64]

$$\mathbb{P}(o_t|o_{1:t-1}) = \sum_{i=1}^{k} \gamma_{t|t-1}(i)\hat{b}_i(t), \tag{4.7}$$

where $\gamma_{t|t-1}(i)$ is the posterior of state s_i at time t, given all the observations up to time $t-1$. That is,

$$\gamma_{t|t-1}(i) = \mathbb{P}[S_t = s_i|o_{1:t-1}] = \sum_{d} \alpha_{t|t-1}(i,d). \tag{4.8}$$

Although, in HSMMs, the predicted observation at time $t+1$ is also given by eq. (4.1), it still remains the question of how to obtain $\gamma_{t+1|t}(\cdot)$ from $\gamma_{t|t-1}(\cdot)$. The forward variable (defined in eq. (2.51)) is computed using the following recursion [64]

$$\alpha_{t|t-1}(i,d) = \mathcal{F}_{t-1}(i)\hat{D}_{id} + b_i^*(t-1)\,\alpha_{t-1|t-2}(i,d+1). \tag{4.9}$$

Before explaining each element of eq. (4.9), we give a brief interpretation of the equation. The intuition behind eq. (4.9) is that the probability of being in state s_i with d observations yet to make while in that state is the summation of two probabilities. The first one materializes the hypothesis that the system, at time $t-1$, transits from any state to state s_i, with duration d. The second hypothesis considers that the system was already in state s_i, of course with one more observation to make compared to the previous timestep $t-1$ (which explains the $d+1$ term). Furthermore, both elements are weighted, as expected, by the "filtered" emission probability of the previous timestep, $b_i^*(t-1)$ (as seen in the unrolled version of eq. (4.9), presented in eq. (4.13)). The elements in eq. (4.9) are:

[4] It corresponds to the joint probability that, at time t, the system is at state s_i and still has d observations to make in that state, given all the observations up to time $t-1$.

- $\mathcal{F}_t(i)$ represents the probability of entering state s_i at time $t+1$ [64]. That is,

$$\mathcal{F}_t(i) = \mathbb{P}\left[\tau_t = 1, S_{t+1} = s_i | o_{1:t}\right] = \sum_j \mathcal{E}_t(j)\hat{A}_{ji}, \tag{4.10}$$

  where $\mathcal{E}_t(j)$ is the probability of ending state s_j at time t, given by [64]

$$\mathcal{E}_t(j) = \mathbb{P}\left[S_t = s_j, \tau_t = 1 | o_{1:t}\right] = \alpha_{t|t-1}(j,1) b_j^*(t). \tag{4.11}$$

- $b_i^*(t)$ is the "filtered" emission probability[5], given by [64]

$$b_i^*(t) = \frac{\hat{b}_i(t)}{\mathbb{P}\left(o_t | o_{1:t-1}\right)} = \frac{\hat{b}_i(t)}{\sum_{j,d} \alpha_{t|t-1}(j,d)\hat{b}_j(t)}. \tag{4.12}$$

We can unroll eq. (4.9) by replacing all terms defined above, yielding

$$\alpha_{t|t-1}(i,d) = \left(\sum_j \alpha_{t-1|t-2}(j,1) b_j^*(t-1)\ \hat{A}_{ji}\right)\hat{D}_{id} + b_i^*(t-1)\,\alpha_{t-1|t-2}(i,d+1). \tag{4.13}$$

Note that all parameter estimates in eq. (4.13) ($\hat{A}_{ji}, \hat{D}_{id}, \hat{b}_i(t)$) are obtained at the M-step of the EM algorithm after convergence. The model is trained using the observations from $t \in [1,T]$, where T corresponds to the timestep of the last training sample. Then, we make predictions about the observations using the test set from $t \in [T+1, n]$, according to Algorithm 1.

Algorithm 1 One-step-ahead prediction algorithm for HSMMs

1: Initialize $t \leftarrow T+1$
2: Collect the forward variable $\alpha_{t-1|t-2}(i,d)$ from the end of training ($\alpha_{T|T-1}(i,d)$)
3: **while** $t \leq n$ **do**
4: **Conditioning**
5: Conditioned on the previous (true) observation o_{t-1}, compute: $b_i^*(t-1) \leftarrow$ eq. (4.12)
6: **Filtering**
7: Compute the next forward variable using the estimated parameters from the EM algorithm ($\hat{\mu} = \{\hat{\boldsymbol{A}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{D}}\}$): $\alpha_{t|t-1}(i,d) \leftarrow$ eq. (4.13).
8: Compute the state marginal probability using the forward variable from the previous step: $\gamma_{t|t-1}(i) \leftarrow$ eq. (4.8)
9: **Prediction**
10: Predict the current observation, $\hat{o}_t$. If we consider a MAP prediction, then

$$\hat{o}_t \sim \mathcal{P}\left(\hat{\lambda}_{\arg\max_i(\gamma_{t|t-1}(i))}\right). \tag{4.14}$$

Another way to predict is to consider not only the most probable state but all states, weighted by the corresponding posteriors found in step (8). That is,

$$\hat{o}_t = \sum_i \gamma_{t|t-1}(i) c\big(\hat{\lambda}_i\big), \tag{4.15}$$

where $c(\hat{\lambda}_i) \sim \mathcal{P}(\hat{\lambda}_i)$ is a sample from the Poisson distribution of state s_i with parameter $\hat{\lambda}_i$.
11: $t \leftarrow t+1$
12: **end while**

---

[5] Which measures how well the model fits the observations.

Estimation of parametric state duration distributions In eq. (2.50), we considered multinomial distributions for the emissions and durations. However, in practice, in the experiments of Sections 4.3 and 4.4, we have used Poisson emissions and geometric state duration distributions. The parameters of the Poisson distributions are obtained as

$$\hat{\lambda}_i = \sum_{j=1}^{m} v_j \hat{B}_{iv_j}, \qquad \forall i \in [k], \quad v_j \in \mathcal{V} = \{v_1, ..., v_m\}, \tag{4.16}$$

where $\hat{B}_{iv_j} = \mathbb{P}[O_t = v_j | S_t = s_i]$ corresponds to element (i, j) of the emission probability matrix $\hat{\boldsymbol{B}}$, computed at the M-step of the EM algorithm (eq. (2.50)).

If the state durations are modeled by geometric distributions, then the PMF of the durations in state s_i ($p_i(d)$) is given by eq. (2.11), with parameter $\theta_i \in]0, 1]$ (instead of A_{ii}). When considering the efficient FB implementation from [64], the geometric parameter θ_i is updated, in the M-step, according to [75]

$$\hat{\theta}_i = \frac{\sum_{t=2}^{n} \gamma_{t|n}(i)}{\sum_{t=2}^{n} \sum_{d=1}^{d_{\max}} d\mathcal{G}_{t|n}(i, d)}. \tag{4.17}$$

In equation (4.17), $\gamma_{t|n}(i)$ is given by eq. (4.8) and $\mathcal{G}_{t|n}(i, d)$ corresponds to the probability of entering state s_i, at time t, with duration d, both conditioned on all training observations [64]. That is,

$$\mathcal{G}_{t|n}(i, d) = \mathbb{P}[S_t = s_i, \tau_{t-1} = 1, \tau_t = d | o_{1:n}] = \mathcal{F}_{t-1}(i) p_i(d) \beta_t(i, d), \tag{4.18}$$

where τ_t is the residual time in state S_t and $\beta_t(i, d)$ is the redefined backward variable from [64].

In this section, we have defined how to do one-step-ahead prediction in HMMs and HSMMs for sequential data. It is often the case, in real sequential datasets, that the observations depend not only on previous observations but also on external factors. For example, for stock price forecasting, history of past stock prices is not sufficient to make a reliable forecast. Instead, external information, such as news articles and analyst opinions, should be somehow included in the model [76]. In the next section, we discuss how to add external information into Markov models.

4.2 Context aware HSMM

For hidden (semi-)Markov models, we can think about introducing external information as a state conditionality. For instance, in Example 3, where we wanted to model if a child is ready or not for daycare with a Markov model with two states $(k = 2)$: **ill** and **ok** -, fever recordings are helpful external features that enrich the model. In this scenario, we could collect n fever measurements, one at each day, and store them on a feature vector $\boldsymbol{C} \equiv \{\boldsymbol{c}_t\}_{t=1}^{n} \in \mathbb{R}^{n \times p}$ with p features.[6] Then, we could add the fever information into the model by conditioning the state transitions. For instance, a state transition **ill** $\rightarrow$ **ok** from time $t - 1$ to time t is supported by a good fever measurement at time $t - 1$, e.g., $\boldsymbol{c}_{t-1} \approx 36.6°$C.

In order to condition the state transitions on features $(\boldsymbol{c}_t)$ that vary in time, the transition matrix in eq. (2.7)

[6] In this case, $p = 1$, since we are only considering a single feature (fever measurement).

has to be redefined as a non-stationary transition matrix $\boldsymbol{A} \equiv \{\boldsymbol{A}^{(t)}\}_{t=1}^{n} \in \mathbb{R}^{n \times k \times k}$, with elements

$$A_{ij}^{(t)} = \mathbb{P}[S_t = s_j | S_{t-1} = s_i, \boldsymbol{c}_{t-1}]. \tag{4.19}$$

We first derive the expressions for a HMM, with no explicit durations. Fig. 4.2 illustrates how the external features condition the state transitions in a HMM. In the next section, we cover a simple method to

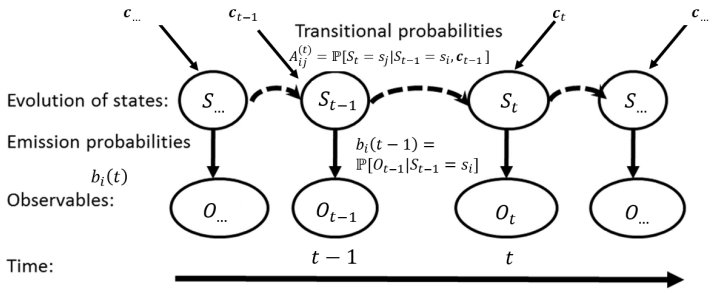


Figure 4.2: Structure and temporal evolution process of a HMM with state transitions conditioned on external features $\{\boldsymbol{c}_t\}_{t=1}^{n}$. Adapted from [49].

condition state transitions on external features using logistic regression.

4.2.1 Logistic regression for parametric Markov models

While classification models predict a binary variable (Yes/No), regression models predict a continuous output. Examples of classification problems include classifying if a tumor is benign or malign depending on extracted features such as dimension and shape [77]. Examples of regression problems include predicting wine yield from climatic and vegetational features [78]. Besides continuous outputs, regression models can also predict probabilities, acting as stochastic models. That is the case of the logistic regression model. The conditional probability of eq. (4.19) can be fit, through a logistic regression model, as a function of the features, given by

$$A_{ij}^{(t)} = \mathbb{P}[S_t = s_j | S_{t-1} = s_i, \boldsymbol{c}'_{t-1}] = \frac{e^{\boldsymbol{\beta}_{ij}^{\intercal} \cdot \boldsymbol{c}'_{t-1}}}{\sum_{g=1}^{k} e^{\boldsymbol{\beta}_{ig}^{\intercal} \cdot \boldsymbol{c}'_{t-1}}} \in [0,1], \qquad \forall i,j \in [k], \quad \forall t \in [1,n], \tag{4.20}$$

where $\boldsymbol{\beta}_{ij} = [\beta_{ij_0}, \quad \beta_{ij_1}, \quad \ldots \quad, \beta_{ij_p}]^{\intercal} \in \mathbb{R}^{p+1}$ are the bias and weights of the logistic regression model with p features, and $\boldsymbol{c}'_{t-1} = [1, \quad \boldsymbol{c}_{t-1}]^{\intercal} \in \mathbb{R}^{p+1}$.

The logistic regression, contrary to linear regression, uses a logistic (sigmoid) function (RHS of eq. (4.20))[7] to squeeze the output to the interval $[0,1]$, which, in this case, is required, since $A_{ij}^{(t)}$ represents a state transition probability. Instead of fitting an hyperplane to the data, as in linear regression models, logistic regression models fit a sigmoid function, and the bias and weights are responsible for defining its slope. The sigmoid function is the decision boundary, since, for classification, the datapoints above (or below) the decision boundary are classified into one class or another.

Here, the (multiple) classes are associated with state transitions, and the same goes for the logistic re-

[7] Usually, one would find the logistic regression term generally defined as $1/(1+e^{-x})$. However, the difference is that, in the RHS of eq. (4.20), we are considering multiple classes instead of two binary classes.

gression weights $\boldsymbol{\beta}_{ij}$.[8] Each class is linked to an indicator variable $y_{t_{ij}} \in \{0, 1\}$, which indicates whether or not there was a state transition $s_i \to s_j$ at time t. The class probability is given by $A_{ij}^{(t)}$ (eq. (4.20)), and it represents the probability of a state transition $s_i \to s_j$, supported by the feature vector $\boldsymbol{c}'_{t-1}$.

Because logistic regression predicts probabilities, rather than just classes, we can fit it using a maximum likelihood approach, where the parameters of the logistic regression are estimated as the MLE of the log-likelihood function. This requires defining the log-likelihood of the full HMM model in Fig. 4.2. The likelihood of this model is given by eq. (2.10), where the stationary transition matrix is replaced by the dynamic version in eq. (4.20). That is, [9,10]

$$\mathbb{P}\big[S, O|\mu, \boldsymbol{\beta}, \boldsymbol{C}\big] = \prod_{t=2}^{n}\prod_{i=1}^{k}\prod_{j=1}^{k}\mathbb{P}(S_t = s_i|S_{t-1} = s_j, \boldsymbol{c}'_{t-1})^{y_{t_{ij}}} = \prod_{t=2}^{n}\prod_{i=1}^{k}\prod_{j=1}^{k}\left(A_{ij}^{(t)}\right)^{y_{t_{ij}}}. \tag{4.21}$$

The log-likelihood (LL) is obtained, after some manipulation of eq. (4.21), as [9]

$$\mathsf{LL} = \sum_{t=2}^{n}\sum_{i=1}^{k}\sum_{j=1}^{k} y_{t_{ij}}\left(\boldsymbol{\beta}_{ij}^{\intercal}\cdot\boldsymbol{c}'_{t-1} - \log\left(\sum_{g=1}^{k} e^{\boldsymbol{\beta}_{ig}^{\intercal}\cdot\boldsymbol{c}'_{t-1}}\right)\right). \tag{4.22}$$

After defining the log-likelihood, the maximum likelihood estimation of the logistic regression weights $\boldsymbol{\beta}_{ij}$ requires computing the partial derivative of the log-likelihood w.r.t the parameters and equating it to zero. That is, fixing one feature $z \in \{0, ..., p\}$, we obtain

$$\frac{\partial \mathsf{LL}}{\partial \beta_{ij_z}} = \sum_{t=2}^{n} y_{t_{ij}}\left(1 - A_{ij}^{(t)}\right)c'_{t-1,z} = 0, \tag{4.23}$$

where $c'_{t-1,z}$ is the z^{th} feature of the feature vector $\boldsymbol{c}'_{t-1}$.

Here, we considered that the state sequence is observed. However, in real scenarios, the state sequence is hidden and we only observe the feature vector $\boldsymbol{C}$ and the observation sequence $o_{1:n}$. Therefore, the hard label $y_{t_{ij}}$ needs to replaced by an expected/soft label, which corresponds to the transition posterior ($\zeta_t(i, j)$) learnt in the E-step of the EM algorithm (see Chapter 2, Section 2.2.2.4).
Note that the soft labels only affect, in a simple manner, the likelihood function in eq. (4.21), but not the way it depends on the parameters. Therefore, only the MLE of the transition matrix changes, the remaining parameters are left unchanged.
It is well known that the logistic regression problem has no closed form solution given that the sigmoid function is non linear. Thus, in eq. (4.23), it is not possible to set the derivative equal to zero analytically. Instead, an iterative numerical method must be implemented to obtain an estimate for the weights of the logistic regression. The most widely used method, due to its simplicity and the ease to include a regularizer, is the *Newton-Raphson* method, addressed in the next section.

[8] The constraint that $\sum_{j=1}^{k} A_{ij}^{(t)} = 1, \quad \forall t \in [1, n]$ is implicitly satisfied by the logistic regression term in eq. (4.20).
[9] The terms that do not depend on the logistic regression weights are removed.
[10] Adapted from `https://www.stat.cmu.edu/~cshalizi/350/lectures/26/lecture-26.pdf`.

4.2.1.1 Newton-Raphson for logistic regression

The update function of the parameters/weights $\boldsymbol{\beta}_{ij} \in \mathbb{R}^{p+1}$ in the Newton-Raphson method is as follows

$$\boldsymbol{\beta}_{ij}^{(\text{new})} = \boldsymbol{\beta}_{ij}^{(\text{old})} - \boldsymbol{H}_{ij}^{-1} \nabla E(\boldsymbol{\beta}_{ij}), \tag{4.24}$$

where $\boldsymbol{H}_{ij} \in \mathbb{R}^{(p+1)\times(p+1)}$ is the Hessian matrix, given by the second derivative of the error function of the weights, $E(\boldsymbol{\beta}_{ij})$. In our problem, the error function is the cross entropy of the model, computed as the negative log-likelihood,[11] that is: $E(\boldsymbol{\beta}_{ij}) = -\text{LL}$, where the log-likelihood term is given by eq. (4.22). The gradient of the error function w.r.t β_{ij_z} is given by eq. (4.23).

The Hessian matrix $\boldsymbol{H}_{ij} \in \mathbb{R}^{(p+1)\times(p+1)}$ is the second derivative of the error function with respect to β_{ij_z}. That is, with elements

$$H_{ij}(z, z') = \frac{\partial}{\partial \beta_{ij_z}} \frac{\partial}{\partial \beta_{ij'_z}} E(\boldsymbol{\beta}_{ij}), \qquad \forall z, z' \in \{0, ..., p\} \tag{4.25}$$

Even though the Newton-Raphson method is the most commonly used method to solve logistic regression, it is neither efficient, in terms of computational time, nor scalable. When there are a lot of classes and features, the algorithm takes too much time to converge, since it requires, at each iteration, the inversion of a $(p+1) \times (p+1)$ matrix - the Hessian matrix.

Researchers have been trying to develop new algorithms to solve multinomial logistic regression efficiently, namely, quasi-Newton methods, which use an approximation for the Hessian matrix rather than compute it directly. The intuition about quasi-Newton methods is to use the secant of the gradient instead of the actual second-order gradient, as shown in Fig. 4.3.

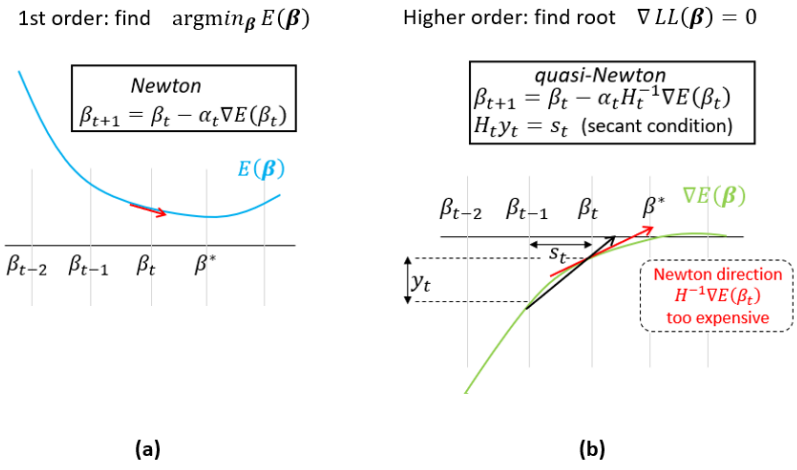


Figure 4.3: Comparison between (a) Newton and (b) quasi-Newton methods.[12]

[11] Since, in general, we want to minimize a function, but here, instead, we want to maximize the log-likelihood function.

[12] Adapted from `https://www.intel.com/content/www/us/en/artificial-intelligence/posts/toward-higher-order-training-a-progressive-batching-l-bfgs-method.html`.

4.2.1.2 Other methods for logistic regression

In 1992, Bohning [79] proposed an approximate algorithm which avoids the inversion of the Hessian matrix at each iteration and replaces it by the inverse of a smaller squared matrix - called the lower bound -, computed only once, at the beginning of the algorithm.
Recently, more complex methods like Coordinate Descent with Newton steps (CDN), Orthant-Wise Limited memory Quasi-Newton method (OWL-QN) [80] and Broyden–Fletcher–Goldfarb–Shanno method (BFGS) [81], have achieved a higher speed solving logistic regression problems.[13]
For the experiments in Sections 4.3 and 4.4, we use the Limited memory BFGS (L-BFGS) method, an extension of the BFGS, which never explicitly forms or stores the Hessian matrix.
BFGS is a quasi-Newton optimization method for multidimensional nonlinear unconstrained functions. It iteratively builds up an approximation of the inverse Hessian matrix through finite difference of successive gradient vectors, as shown in Fig. 4.3(b). It starts at an initial starting point and computes the approximation of the Hessian matrix at that point. Then, it repeats the process of computing the search direction using the approximated Hessian matrix, computes an optimum step size using a line search algorithm and uses that step size to go to the next point. Then, the Hessian matrix is computed again at the new position. This process repeats until convergence or, if convergence is not possible[14], some stop criteria is met.

After having detailed how the weights of the logistic regression are estimated, we introduce, in the following section, some implementation issues and common solutions.

4.2.1.3 Implementation issues

It is well known that the logistic regression can suffer from complete separation [83]. If the data is perfectly separated a priori, the L-BFGS algorithm does not converge, because the weights would become infinite. This problem can be solved by introducing a penalization on the weights in the log-likelihood of eq. (4.22). The resulting log-likelihood is given by

$$\mathsf{LL} = \sum_{t=2}^{n}\sum_{i=1}^{k}\sum_{j=1}^{k} y_{t_{ij}}\left(\boldsymbol{\beta}_{ij}^{\intercal}\cdot\boldsymbol{c}'_{t-1} - \log\left(\sum_{g=1}^{k} e^{\boldsymbol{\beta}_{ig}^{\intercal}\cdot\boldsymbol{c}'_{t-1}}\right)\right) - \frac{1}{2}\varphi||\boldsymbol{\beta}_{ij}||^2, \tag{4.26}$$

where $\varphi \in [0,1]$ is the regularization parameter and the added term $-\frac{1}{2}\varphi||\boldsymbol{\beta}_{ij}||^2$ corresponds to the regularization, which penalizes large weights $\boldsymbol{\beta}_{ij}$.
The gradient in eq. (4.23) would also contain a regularization term as $\frac{\partial}{\partial\boldsymbol{\beta}_{ij}}\Big(-\frac{1}{2}\varphi||\boldsymbol{\beta}_{ij}||^2\Big) = -\varphi\boldsymbol{\beta}_{ij}$.
The regularization parameter φ is a hyperparameter that must be tuned.
Another implementation detail concerns how the weights are initialized. Usually they are either set to zero or initialized randomly. However, a more clever approach is to initialize them with the resulting weights from the previous iteration - called a *warm start*. In the experiments with both synthetic and

[13] This article [82] presents a theoretical and practical comparison of some of these methods.
[14] BFGS is only guaranteed to converge if the function to optimize has a quadratic Taylor expansion near the maximum [81].

real datasets, in Sections 4.3 and 4.4, respectively, we used warm start to initialize the weights of the L-BFGS method and binary-search to tune the regularization parameter φ.

We are now able to fully specify a new model: HSMM-LR, which is simply a hidden semi-Markov model with state transitions conditioned on external features through a logistic regression (LR) model.

4.2.2 HSMM-LR

Since we are considering the simplest version of HSMMs - the explicit duration hidden Markov model (EDHMM), introduced in Section 2.2.1.4 -, in which a state transition does not depend on the duration of the previous state, then the transition matrix is, in fact, the same as in HMMs. The only difference is the additional state duration matrix which is estimated separately. For this reason, we can use the same expressions as in HMMs (eqs. (4.21) to (4.23)) to condition the EDHMM state transitions in external features through logistic regression. Moreover, self-transitions are not allowed in this model, therefore, $A_{ii}^{(t)} = 0$, which means that the logistic regression weights $\boldsymbol{\beta}_{ii}$ need not to be estimated.
The rest of the weights are estimated as the output of k L-BFGS blocks, one for each state. That is,

$$\boldsymbol{\beta}_i = \text{L-BFGS}\Big(\boldsymbol{C}'_{t_i-1}, \zeta_{t_i}(i,\cdot)\Big) \in \mathbb{R}^{(p+1)\times k}, \qquad \forall i \in [k]. \tag{4.27}$$

The inputs of the i^{th} L-BFGS block in eq. (4.27) are:

- $\boldsymbol{C}'_{t_i-1} \in \mathbb{R}^{r\times(p+1)}$: a subset of the feature vector $\boldsymbol{C}' \in \mathbb{R}^{n\times(p+1)}$, which selects only the features at r timesteps t_i. Those timesteps correspond, according to the predicted model, to the instants where a transition from state s_i (to any other state) occurred. These instants are obtained from the predicted state sequence ($\hat{S}$), output of the *Viterbi* algorithm, which we run simultaneously with the E-step of the EM algorithm.[15]
  That is, after the E-step, we compute the predicted state sequence ($\hat{S}$) and we store the indexes where there occurs a transition from state s_i in $\hat{S}$. Those indexes correspond to the timesteps t_i.

- $\zeta_{t_i}(i,\cdot) \in \mathbb{R}^{r\times k}$: the transition posteriors from state s_i to all states, also selected only at timesteps t_i. In fact, instead of the state transition posteriors ζ, we use the normalized transition probabilities from the M-step $\left(\hat{A}_{i(\cdot)}^{(t_i)}\right)$, without the diagonals due to the explicit duration formulation. This is an approximation for numerical reasons, since the transition posteriors are not normalized.

The output of the i^{th} L-BFGS block in eq. (4.27) is the weight vector $\boldsymbol{\beta}_i \in \mathbb{R}^{(p+1)\times k} = [\boldsymbol{\beta}_{i1}, ..., \boldsymbol{\beta}_{ik}]$, where $\boldsymbol{\beta}_{ij} \in \mathbb{R}^{p+1}$, and k is the number of states. Therefore, the complete set of weights $\boldsymbol{\beta}$ is given by

$$\boldsymbol{\beta} = [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_i, ..., \boldsymbol{\beta}_k] = \begin{bmatrix} \boldsymbol{\beta}_{11} & \boldsymbol{\beta}_{21} & ... & \boldsymbol{\beta}_{k1} \\ \vdots & ... & \boldsymbol{\beta}_{ij} & \vdots \\ \boldsymbol{\beta}_{1k} & \boldsymbol{\beta}_{2k} & ... & \boldsymbol{\beta}_{kk} \end{bmatrix} \in \mathbb{R}^{k\times(p+1)\times k}, \tag{4.28}$$

[15] Recall that the state sequence, predicted by the model in the *Viterbi* algorithm, is obtained by computing the $\arg\max$ of the state posteriors at each timestep.

The HSMM-LR parameter estimation problem is solved through an iterative method based on the EM algorithm.

In the E-step, we estimate the state and transition posteriors using the forward-backward algorithm with a dynamic state transition matrix $\hat{\boldsymbol{A}}^{(\boldsymbol{t})}$.

In the M-step, the HSMM parameters are (re-)estimated ($\hat{\mu} = \{\hat{\boldsymbol{A}}^{(\boldsymbol{t})}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{D}}\}$) based on those posteriors. The non-stationary transition matrix $\hat{\boldsymbol{A}}^{(\boldsymbol{t})}$ is estimated for each timestep t separately, instead of averaging the results for all timesteps (see eq. (2.50)). The L-BFGS is included in the M-step, where the logistic regression weights are (re-)estimated using the transition matrix $\hat{\boldsymbol{A}}^{(\boldsymbol{t})}$ and the feature vector, according to eq. (4.27). After updating the logistic regression weights, the conditioned transition matrix is re-computed according to eq. (4.20). The resulting matrix is used as the input of the EM algorithm in the next iteration. This process repeats until the EM converges or some stop criteria is met.

To obtain the timesteps t_i, we need the predicted state sequence, computed from the $\arg\max$ of the state posteriors at each timestep. The $\arg\max$ computation brings uncertainty to the estimation of the state sequence, therefore, its use should be avoided. Besides, by using the $\arg\max$, the algorithm becomes even more initialization-dependent. Following that mindset, another option is to solve k L-BFGS blocks, where, instead of selecting features and labels at timesteps t_i, we input all features $\boldsymbol{C}' \in \mathbb{R}^{n\times(p+1)}$ and all transition probabilities $\hat{A}^{(\cdot)}_{i(\cdot)} \in \mathbb{R}^{n\times k}$ as labels. That is,

$$\boldsymbol{\beta}_i = \text{L-BFGS}(\boldsymbol{C}', \hat{A}^{(\cdot)}_{i(\cdot)}) \in \mathbb{R}^{(p+1)\times k}, \qquad \forall i \in [k]. \tag{4.29}$$

In Section 4.4.4.3, we present the results of the experiments with real data, for the task of comparing the logistic regression method herein proposed with and without all features.

To conclude, in the present section we introduce a new model, HSMM-LR, that is able to both learn temporal patterns in data and how they relate with external information. It remains the question of how to make forecasts with this model.

One-step-ahead prediction in HSMM-LR The one-step-ahead prediction problem in HSMM-LR is addressed in the same way as for HSMMs, in Section 4.1.2. The only adaptation needed is the replacement of the stationary transition matrix (in eq. (4.13)) by the time-dependent version from eq. (4.20). That is,

$$\alpha_{t|t-1}(i,d) = \left(\sum_j \alpha_{t-1|t-2}(j,1) b^*_j(t-1)\ \hat{A}^{(t)}_{ji}\right)\hat{D}_{id} + b^*_i(t-1)\,\alpha_{t-1|t-2}(i,d+1). \tag{4.30}$$

The rest of the elements remain the same and can be revisited in eqs. (4.8) and (4.12). Note that the parameter estimates are the output of the M-step of the EM algorithm after convergence. After computing the forward variable for all timesteps in eq. (4.30), the one-step-ahead prediction is obtained from eqs. (4.14) or (4.15).

4.3 HSMM estimation accuracy: synthetic experiments

Before presenting the forecasting results, we show some synthetic experiments we have performed to assess the estimation accuracy of HSMMs, using the HMM as a baseline.

In the experiments from Sections 4.3.1 to 4.3.6, the order estimation strategy is off and the number of states (k) and maximum state duration (d_{max}) are known *a priori*.

4.3.1 Analysis of the effect of the distance between Poisson distributions

The model herein considered is a HSMM with $k = 3$ and $d_{max} = 2$. The synthetic data consists of $N = 10$ sequences with $n = 1000$ observations each, sampled from this model with random multinomial distributions for parameters $\boldsymbol{\pi}$, $\boldsymbol{A}$ and $\boldsymbol{D}$. In the conducted experiment, the parameters of the three Poisson distributions become increasingly closer in 39 iterations. In Table 4.1, we show the true and estimated parameters for the first and last iterations. In each iteration, both λ_2 and λ_3 are reduced by a factor of 1.1 (starting, at iteration 0, from $\lambda_1 = 1, \lambda_2 = 100, \lambda_3 = 200$), while $\lambda_1 = 1$ remains the same. Fig. 4.4[16] illustrates the results for all iterations. We can verify that, due to the proximity between the means of

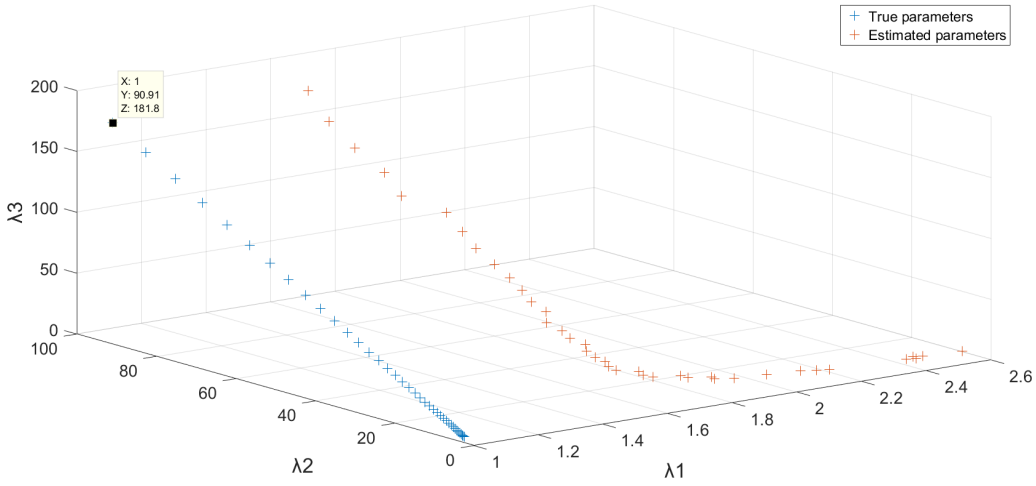


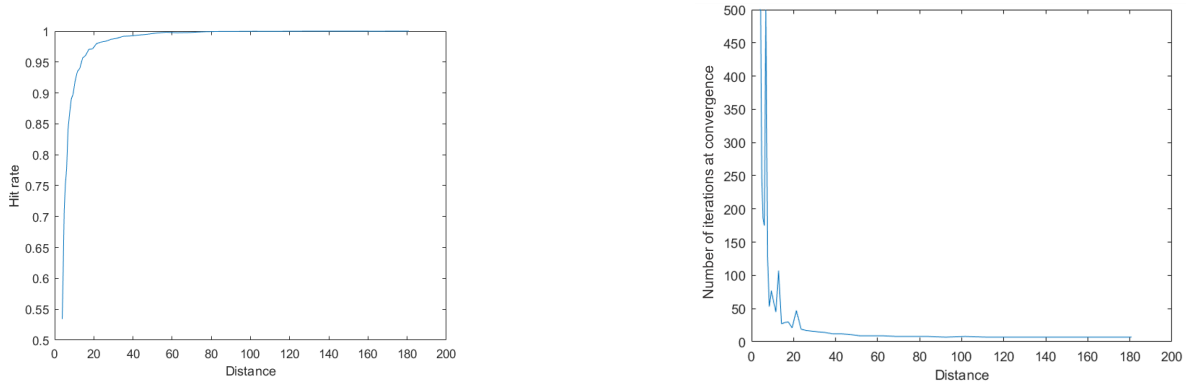
Figure 4.4: Evolution of the parameter estimates for λ_1, λ_2 and λ_3. The original parameters are depicted in blue while the estimated ones are shown in red.

Table 4.1: Results of the estimated rates of the three Poisson distributions.

Iteration	Original parameters			Estimated parameters			Overlap [%] [17]
	λ_1	λ_2	λ_3	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$	
1	1	90.91	181.82	1.60	90.79	181.89	7.64×10^{-3}
11	1	35.05	70.10	1.96	35.52	70.19	1.4
39	1	2.43	4.86	2.46	3.55	6.06	86.17

[16] Due to the scale, the first state in Fig. 4.4 is the only one in which the parameter estimation error is evident.
[17] The overlap percentage is an approximation of the integral of the overlap area between multiple Poisson distributions.

the Poisson processes, the estimation accuracy is compromised. This is confirmed by analyzing the hit rate, defined in eq. (3.16). Fig. 4.5(a) shows the evolution of the hit rate with the distance between the Poisson distributions. The distance is measured as the sum of the absolute difference of consecutive means for the three Poisson distributions. The distance between the Poisson distributions in the last

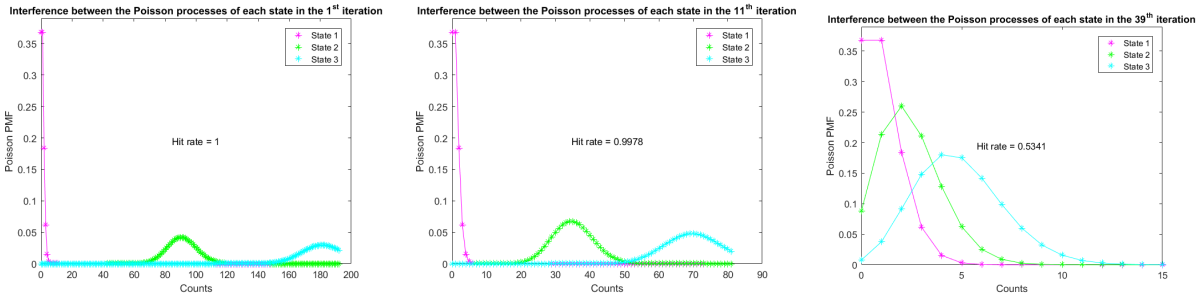


(a) (b)

Figure 4.5: Evolution of the: (a) hit rate, and (b) number of iterations of the EM algorithm at convergence, with the distance between the Poisson distributions.

iteration (39^{th}) is small ($d_{39} = 3.86$), when compared to the distance in the first iteration ($d_1 = 180.8$). The hit rate is approximately 0.55 at the last iteration, and, as the distance increases, it converges to 1.

The hit rate decay phenomenon starts to occur as soon as the Poisson distributions overlap. The data samples (observations) generated by the Poisson distributions are similar to each other, which leads to clustering issues in the estimation algorithm. The observations are therefore classified as belonging to the wrong cluster/state, which not only affects the hit rate but also the correct estimation of the Poisson means. In Fig. 4.6, the Poisson PMF of the three states is depicted for (a) the 1^{st}, (b) the 11^{th}, and



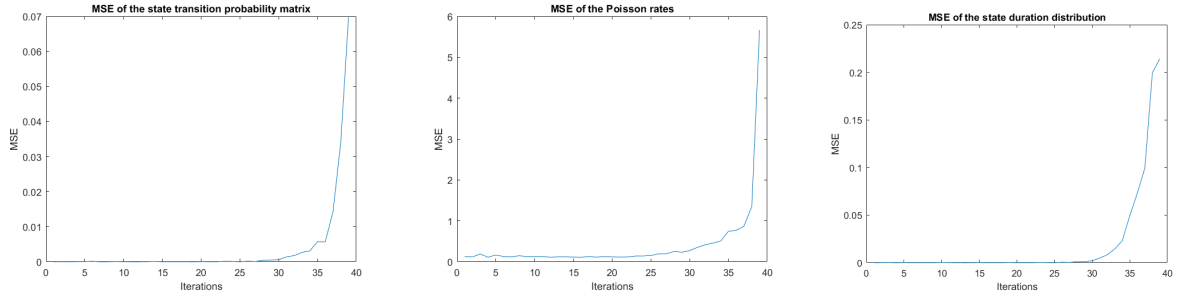
(a) (b) (c)

Figure 4.6: Interference between the Poisson processes of each state in the: (a) 1^{st} iteration; (b) 11^{th} iteration; (c) 39^{th} iteration.

(c) the 39^{th} iterations. The 11^{th} iteration is the first one where the hit rate is below 0.998 and, in fact, it corresponds to the moment when the distributions associated with the second and third states start overlapping, according to Fig. 4.6(b). In the last iteration, all distributions overlap, with a total overlap percentage of 86.17%, according to Table 4.1, which justifies the poor hit rate and estimation accuracy. It also supports the results illustrated by Fig. 4.5(b), concerning the convergence of the EM algorithm.

The maximum number of iterations[18] is reached when the distance is small. On the contrary, when the distance is enough to ensure that the Poisson distributions do not considerably overlap, the algorithm stop criteria is met before the maximum number of iterations is reached.

Finally, in Fig. 4.7, we present the evolution of the mean-squared-error (MSE)[19] of the estimated parameters along iterations. By inspection of the figure, we conclude that, as we approach the high overlap region towards iteration 39, the parameter estimates diverge, and the MSE increases, as expected.[20]



(a) (b) (c)

Figure 4.7: Mean-squared-error of the estimated: (a) state transition probability matrix, (b) means/rates of the Poisson processes, and (c) state duration distribution.

We also analyze the effect of increasing the number of sequences (N) and sequence length (n) in Appendix F. In the following experiments, we consider a HSMM with $k = 3$ and $d_{max} = 4$, Poisson emissions and non-parametric state duration distributions. In detail, the setup was the following[21]

$$\boldsymbol{\pi} = \begin{bmatrix} 0.6293 \\ 0.3621 \\ 0.0086 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 15 \\ 30 \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} 0 & 0.9767 & 0.0233 \\ 0.4667 & 0 & 0.5333 \\ 0.1927 & 0.8073 & 0 \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} 0.2630 & 0.4539 & 0.0089 & 0.2742 \\ 0.3244 & 0.0594 & 0.3747 & 0.2415 \\ 0.0086 & 0.4716 & 0.3704 & 0.1495 \end{bmatrix}. \tag{4.31}$$

The multinomial distributions for $\boldsymbol{\pi}$, $\boldsymbol{A}$ and $\boldsymbol{D}$ were obtained randomly. To increase statistical significance, the errors are compared based on 100 independent Monte Carlo runs. In each run, the original matrices are the same, but by sampling from them we obtain different state and observation sequences.

4.3.2 Analysis of the effect of the duration

The HSMM herein considered have the setup from eq. (4.31), except the state duration probability matrix $\boldsymbol{D}$. In this experiment, we fix the maximum allowed state duration at $d_{max} = 100$, and we analyze the effect of varying the expected average duration ($\mathbb{E}(d|S_t)$) of the first state. The expected average duration of the second and third states remains always equal to 10 units of duration, while the expected average duration of the first state increases from 10 to 100 in steps of 10. The synthetic data consists of one sequence ($N = 1$) of 100 observations ($n = 100$).

[18] Set to 500 in this experiment.

[19] $\text{MSE}(\hat{\boldsymbol{A}}) = \sum_{i=1}^{k}\sum_{j=1}^{k}|\hat{A}_{ij} - A_{ij}|^2$, $\text{MSE}(\hat{\boldsymbol{D}}) = \sum_{i=1}^{k}\sum_{d=1}^{d_{max}}|\hat{D}_{id} - D_{id}|^2$, $\text{MSE}(\hat{\boldsymbol{\pi}}) = \sum_{i=1}^{k}|\hat{\pi}_i - \pi_i|^2$.

[20] We do not show the MSE of the initial state distribution ($\hat{\boldsymbol{\pi}}$), because the overlap does not directly affect its estimation.

[21] Probabilities were rounded to 4 decimal places.

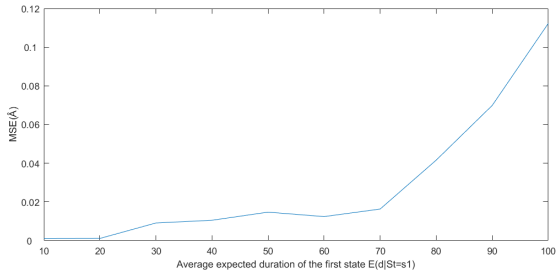


Figure 4.8: Evolution of the mean-squared-error of the transition matrix, with the average expected duration of the first state, $\mathbb{E}(d|S_t = s_1)$.

Parameter estimation of the Poisson rates			
d^*	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$
10	$\mathbf{2.87 \pm 2.51}$	$\mathbf{15.13 \pm 0.67}$	$\mathbf{30.00 \pm 3.14}$
30	3.06 ± 4.23	14.92 ± 1.22	26.17 ± 6.90
50	3.84 ± 5.11	15.62 ± 1.77	23.22 ± 9.63
70	3.25 ± 3.95	15.00 ± 1.64	21.88 ± 8.89
90	6.27 ± 6.30	13.70 ± 3.44	15.00 ± 8.11

Table 4.2: Poisson parameter estimates for $\lambda_1 = 2$, $\lambda_2 = 15$ and $\lambda_3 = 30$, with increasing average expected duration of the first state, $d^* \equiv \mathbb{E}(d|S_t = s_1)$.[23]

The effect of increasing the expected average duration of the first state is analyzed in terms of the accuracy in the estimation of the Poisson rates (Table 4.2[22]), and the mean-squared-error of the state transition matrix (Fig. 4.8).

As expected, the mean-squared-error increases with the duration given that there are less transitions in the state sequence, and, therefore, the estimation of the transition matrix is compromised. The results of the estimation of the Poisson rates (Table 4.2) prove that, for high durations, the parameters are not correctly estimated. This result does not depend only on the duration but also on the original transition matrix, obtained randomly as

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0.9767 & 0.0233 \\ 0.4667 & 0 & 0.5333 \\ 0.1927 & 0.8073 & 0 \end{bmatrix}.$$

This transition matrix almost forces a transition to state 2 whenever we are in state 1. Thus, adding the fact that state 1 has high durations, this causes state 3 to almost disappear from the state sequence, which, in turn, causes $\hat{\lambda}_3$ to be the new $\hat{\lambda}_2$ and $\hat{\lambda}_1$ tries to merge with $\hat{\lambda}_2$, as shown in Table 4.2.

4.3.3 Model mismatch

The synthetic data used in this experiment consists of $N = 1$ sequence of $n = 100$ observations, sampled from the setup of eq. (4.31). In the model mismatch test, we train the HSMM using values for d_{max} different than the true one ($d_{max} = 4$). In other words, the HSMM is trained using an observation sequence with $d_{max} = 4$, but the duration matrix $\boldsymbol{D} \in \mathbb{R}^{k \times d'_{max}}$ to be estimated considers a different maximum duration: $d'_{max} = \{2, 6, 10\}$. After running the parameter estimation algorithm, the results are compared to the case where the true model hyperparameter $d_{max} = 4$ is known. We also assess the results of the data, generated by a HSMM, but trained by a HMM.

To compare the results, we present the estimation error of the state and observation sequences separately, given by

$$\text{err}(S) = \sqrt{\frac{1}{N \times n} \sum_{i=1}^{N} \sum_{t=1}^{n} \left| \hat{S}_t^{(i)} - {S_t}^{(i)} \right|^2}, \qquad \text{err}(O) = \sqrt{\frac{1}{N \times n} \sum_{i=1}^{N} \sum_{t=1}^{n} \left| \hat{\lambda}_{\hat{S}_t^{(i)}} - {O_t}^{(i)} \right|^2}, \tag{4.32}$$

[22] For ease of reading, we present only the results for half the durations.

[23] Results are presented in the format mean $\pm$ std for the 100 Monte Carlo runs. The best values are identified in bold.

Table 4.3: Results of the sequence estimation for the model mismatch experiment, presented in the format mean $\pm$ std for the 100 Monte Carlo runs. The best values are identified in bold and the true $d_{max} = 4$ is shown in bold.

		Sequence estimation		
Model	d_{max}	err(S)	err(O)	Hit rate
HMM	-	0.2002 ± 0.0598	$\mathbf{3.6390 \pm 0.3529}$	0.9552 ± 0.0221
HSMM	2	0.4697 ± 0.0300	5.8896 ± 0.4293	0.7784 ± 0.0285
HSMM	**4**	$\mathbf{0.1372 \pm 0.0631}$	3.7392 ± 0.3874	$\mathbf{0.9771 \pm 0.0162}$
HSMM	6	0.1469 ± 0.0660	3.7380 ± 0.3837	0.9739 ± 0.0182
HSMM	10	0.1525 ± 0.0701	3.7371 ± 0.3843	0.9718 ± 0.0198

Table 4.4: Results of the parameter estimation for the model mismatch test. The values closer to the true values $\lambda_1 = 2$, $\lambda_2 = 15$ and $\lambda_3 = 30$ are identified in bold.

		Parameter estimation			
Model	d_{max}	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$	MSE$(\hat{\boldsymbol{A}})_{[\times 10^{-3}]}$
HMM	-	$\mathbf{2.02 \pm 0.34}$	14.74 ± 0.60	29.47 ± 1.38	158.00
HSMM	2	3.16 ± 1.52	15.34 ± 1.18	26.89 ± 2.25	2.60
HSMM	**4**	2.32 ± 0.31	$\mathbf{15.04 \pm 0.59}$	$\mathbf{30.24 \pm 1.19}$	$\mathbf{0.05}$
HSMM	6	2.32 ± 0.32	15.05 ± 0.60	30.26 ± 1.19	0.06
HSMM	10	2.32 ± 0.32	15.05 ± 0.60	30.28 ± 1.23	0.07

and the hit rate from eq. (3.16). These results are presented in Table 4.3. From inspection of the table, we observe that the HSMM algorithm yields the best estimation results, with the lowest state sequence error and highest hit rate occurring for $d_{max} = 4$. Whenever there is a model mismatch, there is a penalization in the estimation accuracy. The scenario with worst results occurs for $d_{max} = 2$ because, in this case, the hyperparameter is below the true value. We also notice that the lowest estimation error for the observation sequence was not achieved for HSMM with $d_{max} = 4$, as expected, but with the HMM. This implies that the Poisson rates of each state are better estimated when we train with a HMM. In fact, from Table 4.4, we find that the Poisson rate of the first state is closer to the true value when trained by a HMM, whereas the Poisson rates for both the second and third states are better estimated for the HSMM with $d_{max} = 4$. However, the difference in the first state is enough to guarantee a lowest estimation error for the observation sequence.

The lowest value for the mean-squared-error of the transition matrix, in Table 4.4, is achieved for the HSMM model when the true hyperparameter $d_{max} = 4$ is known. This should come as no surprise, given that in HSMMs the durations are taken into account in the definition of the transitions between states. Thus, the best result should be the one without model mismatch.[24]

We conclude that a model mismatch where $d_{max} < 4$ gets the worse results in terms of estimation

[24]Note that the high value in the MSE of the HMM state transition matrix results from the way the transitions are accounted for in HMMs, which is not the same as in HSMMs. Furthermore, contrary to HMMs, in HSMMs the diagonal entries of the transition matrix are zero.

accuracy. Besides, a model mismatch with $d_{max} > 4$ is worse than when the true model hyperparameter is known. However, in that case, the results are not significantly worsened because, even though d_{max} is larger than the true value, the higher durations can be assigned with low or even zero probability in the state sequence.

4.3.4 Semi-Markov assumption

It is important to test the robustness of HSMMs to the semi-Markov assumption, for HMM-driven data. To do that, we generate data from a HMM with a similar setup as before (eq. (4.31)), except for the transition matrix, which is obtained randomly but with no zero-diagonal entries. That is,

$$\boldsymbol{A} = \begin{bmatrix} 0.3225 & 0.3644 & 0.3131 \\ 0.1883 & 0.2916 & 0.5201 \\ 0.2636 & 0.1671 & 0.5693 \end{bmatrix}. \tag{4.33}$$

We run the EM algorithm for a HMM and HSMMs with different values of d_{max}, to compare the obtained accuracy. The results are summarized in Table 4.5.

Table 4.5: Results of the sequence estimation with different models used in the training phase (EM algorithm) for HMM-driven data. Results are presented in the format mean $\pm$ std for the 100 Monte Carlo runs. The best values are identified in bold.

		Sequence estimation		
Model	d_{max}	err(S)	err(O)	Hit rate
HMM	-	0.1470 ± 0.0467	$\mathbf{3.6714 \pm 0.3184}$	0.9740 ± 0.0140
HSMM	2	$\mathbf{0.0238 \pm 0.0541}$	3.8695 ± 0.3682	$\mathbf{0.9965 \pm 0.0092}$
HSMM	4	0.0397 ± 0.0613	3.8240 ± 0.3604	0.9947 ± 0.0098
HSMM	6	0.0428 ± 0.0643	3.8223 ± 0.3576	0.9937 ± 0.0117
HSMM	10	0.0460 ± 0.0732	3.8324 ± 0.3811	0.9916 ± 0.0265

From inspection of the table, we conclude that the HSMM can properly capture the dynamics of HMM-driven data, in particular, it performs better when the maximum allowed duration is $d_{max} = 2$. It is also important to notice that the HSMM estimate of the state duration distribution ($\hat{\boldsymbol{D}}$) should reflect the fact that the durations in HMMs are implicitly modeled as a geometric distribution with parameter A_{ii}. We can build the original HMM duration matrix by filling in the values from eq. (2.11), using the parameters from the diagonal entries of eq. (4.33). The result, after normalization, is presented in eq. (4.34), as $\boldsymbol{D}_{\text{HMM}}$, fixing $d_{max} = 2$, and the HSMM estimate (for $d_{max} = 2$) is shown as $\hat{\boldsymbol{D}}$.

$$\boldsymbol{D}_{\text{HMM}} = \begin{bmatrix} 0.7561 & 0.2439 \\ 0.7742 & 0.2258 \\ 0.6372 & 0.3628 \end{bmatrix}, \qquad \hat{\boldsymbol{D}} = \begin{bmatrix} 0.7017 & 0.2983 \\ 0.5667 & 0.4333 \\ 0.7518 & 0.2482 \end{bmatrix}. \tag{4.34}$$

We can state that even though the estimate in eq. (4.34) is not perfect, the $\arg\max$ of the duration for

each state is matching. As future work, we suggest investigating if, with more observations ($n > 100$), the state duration's estimate would be closer to $\boldsymbol{D}_{\text{HMM}}$.

4.3.5 Robustness to Nonstationarity

Another interesting experiment is to add nonstationary properties to the data and assess how well the EM algorithm predicts the parameters. That is, we change the setup in the last half of the data sequence. The data is HSMM-driven and we consider two scenarios. In the first scenario, the changes are made in the state transition matrix $\boldsymbol{A}$ and in the state duration distribution $\boldsymbol{D}$, whereas, in the second scenario, we analyze the nonstationarity with a slight drift only in the Poisson parameters $\boldsymbol{\lambda}$.

In the first scenario, half of the sequence (i.e, 50 observations) is governed by the original matrices $\boldsymbol{A}$ and $\boldsymbol{D}$ from the setup of eq. (4.31), while the other half is governed by different matrices, $\boldsymbol{A'}$ and $\boldsymbol{D'}$, given by

$$\boldsymbol{A'} = \begin{bmatrix} 0 & 0.7 & 0.3 \\ 0.7 & 0 & 0.3 \\ 0.4 & 0.6 & 0 \end{bmatrix}, \qquad \boldsymbol{D'} = \begin{bmatrix} 0.3 & 0.5 & 0.05 & 0.15 \\ 0.4 & 0.1 & 0.3 & 0.2 \\ 0 & 0.6 & 0.3 & 0.1 \end{bmatrix}. \tag{4.35}$$

The sequence estimation results are summarized in Table 4.6. In eq. (4.36), we show the resulting estimates of all parameters, except $\boldsymbol{\pi}$.

Table 4.6: Results of the sequence estimation for HSMM-driven data with nonstationary properties.

	Sequence estimation		
Setup	err(S)	err(O)	Hit rate
$\boldsymbol{A}, \boldsymbol{D} \rightarrow \boldsymbol{A'}, \boldsymbol{D'}$	0.1304 ± 0.0731	3.7046 ± 0.3403	0.9777 ± 0.0169
$\boldsymbol{A'}, \boldsymbol{D'}$	$\mathbf{0.1214 \pm 0.0732}$	$\mathbf{3.6324 \pm 0.3254}$	$\mathbf{0.9799 \pm 0.0172}$
$\boldsymbol{A}, \boldsymbol{D} \rightarrow \boldsymbol{A''}, \boldsymbol{D''}$	0.1573 ± 0.0644	3.6030 ± 0.3693	0.9711 ± 0.0191

Comparing the results of sequence estimation between the nonstationary (first row of Table 4.6) and stationary cases (Table 4.3, HSMM with $d_{max} = 4$), we find that the nonstationary properties added to the HSMM-driven data did not change much the results. In fact, we do observe a slight error reduction in the sequence estimation results.

$$\hat{\boldsymbol{A}} = \begin{bmatrix} 0 & 0.8161 & 0.1839 \\ 0.5741 & 0 & 0.4259 \\ 0.3106 & 0.6894 & 0 \end{bmatrix}, \hat{\boldsymbol{D}} = \begin{bmatrix} 0.2594 & 0.4982 & 0.0399 & 0.2025 \\ 0.3467 & 0.0935 & 0.3316 & 0.2282 \\ 0.0439 & 0.5127 & 0.2937 & 0.1497 \end{bmatrix}, \hat{\boldsymbol{\lambda}} = \begin{bmatrix} 2.3237 \\ 14.9875 \\ 30.0222 \end{bmatrix} \pm \begin{bmatrix} 0.2897 \\ 0.6442 \\ 1.1428 \end{bmatrix}. \tag{4.36}$$

From eq. (4.36), we also observe that the estimates of $\boldsymbol{A}$ and $\boldsymbol{D}$ are halfway between the original ones (from the setup of eq. (4.31)) and the slightly changed ones (from eq. (4.35)), as expected.

One would anticipate that the nonstationary case would give worse results, given that a change in the dynamics of the data brings uncertainty to the estimation algorithm. Instead, we observed a slight improvement in the results, which may have happened because the changes were not significant. To better understand this theory, we fed the algorithm with an entire sequence generated by matrices $\boldsymbol{A'}$ and $\boldsymbol{D'}$. The results are summarized in the second row of Table 4.6. Indeed, our assumption holds, given that the results are better than the ones corresponding to the nonstationary case (first row of Table 4.6) and the ones from the stationary case (Table 4.3, HSMM with $d_{max} = 4$) using only the original matrices $\boldsymbol{A}$ and $\boldsymbol{D}$.

To discover the nonstationarity tipping point for HSMMs, we tested with a setup notably different from the original one. That is, the second half of the state and observation sequences is sampled from

$$\boldsymbol{A''} = \begin{bmatrix} 0 & 0.3 & 0.7 \\ 0.8 & 0 & 0.2 \\ 0.9 & 0.1 & 0 \end{bmatrix}, \qquad \boldsymbol{D''} = \begin{bmatrix} 0.19 & 0.2 & 0.01 & 0.6 \\ 0.05 & 0.7 & 0.15 & 0.1 \\ 0.3 & 0.2 & 0.4 & 0.1 \end{bmatrix}. \tag{4.37}$$

The results are summarized in the third row of Table 4.6 and, in eq. (4.38), we show the resulting estimates of all parameters, except $\boldsymbol{\pi}$.

$$\hat{\boldsymbol{A}} = \begin{bmatrix} 0 & 0.5580 & 0.4420 \\ 0.5678 & 0 & 0.4322 \\ 0.6195 & 0.3805 & 0 \end{bmatrix}, \hat{\boldsymbol{D}} = \begin{bmatrix} 0.2261 & 0.2876 & 0.0226 & 0.4637 \\ 0.2371 & 0.2168 & 0.3492 & 0.1968 \\ 0.2135 & 0.2689 & 0.3844 & 0.1332 \end{bmatrix}, \hat{\boldsymbol{\lambda}} = \begin{bmatrix} 2.3185 \\ 14.8431 \\ 29.7839 \end{bmatrix} \pm \begin{bmatrix} 0.2163 \\ 0.8363 \\ 1.3936 \end{bmatrix}. \tag{4.38}$$

We see that the estimation accuracy of the state sequence is worse in this case, as anticipated.

Given that, in order to significantly affect the estimation results, it is required a drastic shift in the data's setup, we can conclude that the HSMM is rather robust to data nonstationarity in $\boldsymbol{A}$ and $\boldsymbol{D}$. An interesting study to develop as future work is to learn how to detect if a (drastic or slight) change occurred in the data's dynamics, since it requires the parameter estimation problem to be performed separately for each detected region where a different dynamic is present.

In the second scenario, we analyze the effect of nonstationarity in the observations by forcing a slight drift in the Poisson parameters. That is, the first half of the sequence is governed by the rates $\boldsymbol{\lambda}$ from the setup of eq. (4.31), while the second half is governed by slightly different rates $\boldsymbol{\lambda}' = \boldsymbol{\lambda} + 0.5$. By increasing the means by 0.5, we are also increasing the variance and, therefore, adding uncertainty to the model. The rest of the parameters ($\boldsymbol{\pi}$, $\boldsymbol{D}$ and $\boldsymbol{A}$) are the ones from eq. (4.31). The sequence estimation results are summarized in the first row of Table 4.7[25], where we also show the predicted Poisson rates.

We conclude that the results are very similar to the stationary case. The difference in the parameter estimation of the Poisson rates is supported by the slight drift added in the last half of the data, which

[25]The estimation error of the observation sequence (err(O)) is not shown because in this experiment the Poisson rates are not constant along the (sampled) state sequence.

Table 4.7: Results of the sequence estimation for HSMM-driven data with nonstationary properties. Original Poisson parameters: $\lambda_1 = 2$, $\lambda_2 = 15$, $\lambda_3 = 30$.

	Sequence estimation				
Setup	err(S)	Hit rate	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$
$\boldsymbol{\lambda} \to \boldsymbol{\lambda}'$	0.1334 ± 0.0750	0.9766 ± 0.0183	2.5563 ± 0.3726	15.3123 ± 0.6221	30.4230 ± 1.1766
$\boldsymbol{\lambda} \to \boldsymbol{\lambda}''$	0.2012 ± 0.0711	0.9543 ± 0.0284	3.6312 ± 0.5020	14.8939 ± 0.7117	28.3598 ± 1.1997

was well captured by the model. Again, we also tested with a larger parameter drift: $\lambda_1'' = 5$, $\lambda_2'' = 15$ and $\lambda_3'' = 27$. The parameter associated with the second state stayed the same, while the others changed by an additive factor of 3 in opposite directions. The results are summarized in the second row of Table 4.7. The estimation accuracy is significantly reduced in this case because the observations are more informative in determining the hidden states. Also, the Poisson estimates of the first and third states are halfway between the original ones and the drifted ones, which is expected, from the merging property of Poisson processes (see Property 5 in Section 2.1.1).

Overall, the results indicate that the HSMM model is rather robust to nonstationarity both in the transitions and durations of the state sequence and in the observations.

4.3.6 Robustness to supervised initialization

The data used in the present experiment consists of $N = 2$ sequences of $n = 1000$ observations, sampled from a HSMM with the setup of eq. (4.31), three states ($k = 3$) and $d_{max} = 4$.

In all previous experiments, the EM algorithm, for each Monte Carlo sequence, starts with ten different random initial estimates (of the parameters), and the results are averaged. It is known that the EM algorithm may converge to a local maximum likelihood estimate of the parameters, depending on the starting values. The high dependence of the EM algorithm on the initialization point is the reason why a variety of heuristic approaches are used to avoid the algorithm stopping at the nearest local maximum and search for the global maximum instead.

In this experiment, the initialization process is no longer random but fixed, using the maximum likelihood estimate of a percentage of the labeled data.[26] Thus, the initial parameters are obtained from a supervised approach (see Appendix A), since, from the labeled data, we have access to the state sequence. Then, we train the model always on the last 20% of data (i.e., using the last 2×200 observations).

Different percentages of labeled data were tested, in order to determine how the amount of supervised information used in the initialization affects the estimation accuracy.[27] The errors in the sequence estimation and observations are shown in Table 4.8. For instance, in the first line of the table, we use 10% of the observation and state sequences to initialize the parameters ($\boldsymbol{A}$, $\boldsymbol{\pi}$, $\boldsymbol{\lambda}$ for HMMs, and additionally $\boldsymbol{D}$ for HSMMs). Then, we train each model (HMM and HSMM) in the last 20% of the data, thus discarding 70% of data.

[26] The labeled data consists of both the observation and state sequences.

[27] The remaining data - i.e., the data not used to train or initialize the model - is discarded.

Table 4.8: Results of the sequence estimation for HSMM-driven data. The results are averaged over 100 Monte Carlo runs and averaged for the two sequences.

Initialization [%]	err(S)		err(O)	
	HMM	HSMM	HMM	HSMM
10	0.1287 ± 0.0309	0.0926 ± 0.0334	2.7004 ± 0.2037	2.8091 ± 0.2216
20	0.1290 ± 0.0298	0.0852 ± 0.0335	2.7043 ± 0.1864	2.7690 ± 0.2049
30	0.1292 ± 0.0300	0.0840 ± 0.0342	2.6988 ± 0.1962	2.7557 ± 0.1990
40	$\mathbf{0.1277 \pm 0.0294}$	0.0830 ± 0.0347	$\mathbf{2.6892 \pm 0.1799}$	2.7557 ± 0.2040
50	0.1285 ± 0.0302	0.0830 ± 0.0348	2.7013 ± 0.1899	2.7528 ± 0.2002
60	0.1293 ± 0.0292	0.0832 ± 0.0350	2.6998 ± 0.1920	2.7444 ± 0.1984
70	0.1296 ± 0.0292	0.0827 ± 0.0348	2.7107 ± 0.1966	2.7431 ± 0.1970
80	0.1306 ± 0.0295	$\mathbf{0.0824 \pm 0.0347}$	2.7090 ± 0.2000	$\mathbf{2.7427 \pm 0.1968}$

In what the state sequence error is concerned, we observe that the HMM is pretty much independent of the initialization. Since the data is generated by a HSMM, initial HMM estimates will not improve the model accuracy. On the other hand, as the percentage of labeled data increases, the state sequence error for HSMMs decreases. Regarding the observation sequence error, the differences are a result of the Poisson estimates, which we do not show here because there are no large variations. In fact, even with the weakest initial estimate (the one obtained from 10% labeled data), the EM algorithm, after some iterations, converges to a value close to the global maximum ($\boldsymbol{\lambda} = [2, 15, 30]$)

In eq. (4.39), we show the HSMM estimate of all parameters for the first Monte Carlo sequence, with 30% of labeled data. We can see that the estimate for $\boldsymbol{\pi}$ is influenced by the fact that there are only two sequences to train the model. The rest of the parameters ($\boldsymbol{\lambda}, \boldsymbol{A}, \boldsymbol{D}$) are, in fact, a pretty good estimate of the original parameters from eq. (4.31).

$$\hat{\boldsymbol{\pi}} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix}, \quad \hat{\boldsymbol{\lambda}} = \begin{bmatrix} 2.3482 \\ 14.8431 \\ 29.8074 \end{bmatrix}, \quad \hat{\boldsymbol{A}} = \begin{bmatrix} 0 & 0.9696 & 0.0304 \\ 0.4891 & 0 & 0.5109 \\ 0.2489 & 0.7511 & 0 \end{bmatrix}, \quad \hat{\boldsymbol{D}} = \begin{bmatrix} 0.2760 & 0.5101 & 0 & 0.2139 \\ 0.2998 & 0.0491 & 0.3693 & 0.2818 \\ 0 & 0.3750 & 0.5256 & 0.0994 \end{bmatrix}. \tag{4.39}$$

4.4 time-series prediction in social media

In this section, we show the time-series forecasting results on a real social media dataset, described in Section 4.4.1. Then, we present a set of different experiments to assess the prediction accuracy.

4.4.1 Dataset

The models used for time-series modeling and forecasting were traditional Markov models with and without external features (HSMM-LR and HMM/HSMM, respectively), and LSTMs.

The dataset consists of an open-source Kaggle dataset[28], which contains 16 million tweets about bitcoin, collected from the Twitter application. The bitcoin dataset contains tweets from 01/01/2016 to 23/11/2019 and is organized into 9 columns: *tweet-id*, *user*, *fullname*, *timestamp*, *url*, *likes*, *replies*, *retweets* and *text*.

In Table 4.9, we present two samples from the bitcoin dataset as an illustration of the field contents. In our experiments, we only use the *timestamp* and *text* fields.

Table 4.9: Example of two samples from the bitcoin dataset. [29]

User	Fullname	Timestamp	Likes	Replies	Retweets	Text
elonmusk	Elon Musk	2018-10-22 23:51:15	1081	24907	8051	"@vicentes @Grimezsz Wanna buy some Bitcoin? ;) ;) https://t.co/9ZbBJ5fuVq"
DetroitCrypto	J. Scardina	2019-05-27 11:49:22	0	0	0	"Current Crypto Prices! BTC: $8721.99 USD ETH: $266.62 USD ..."

4.4.1.1 Data statistics and preprocessing

The dataset, before the preprocessing steps, contains 74.38% of english sentences in the *text* field.[30] The preprocessing steps were the following:

- Select only the *timestamp* and *text* columns and sort the timestamps by chronological order.
- Filter only the entries from 05/2019 to 11/2019.
  Before 05/2019, the amount of tweets was significantly lower, since bitcoin was not a very popular topic yet. These oscillations may be due to the popularity of the topic but also to the data collection process.
  Thus, the first timestamp is 2019-05-01 00:00:02 and the last timestamp is 2019-11-23 15:45:57. The total number of tweets is reduced from 16 millions to 14194416 tweets, of which 10421938 are english (73.42%). We used only english tweets.
- Clean text.
  For example, the first sample from Table 4.9 would become "@user @user Wanna buy some Bitcoin? WINKSMILE WINKSMILE URL".
- Discretize dataset.

After preprocessing, we have a $120\,\mathrm{GB}$ dataset.

4.4.1.2 Discretization

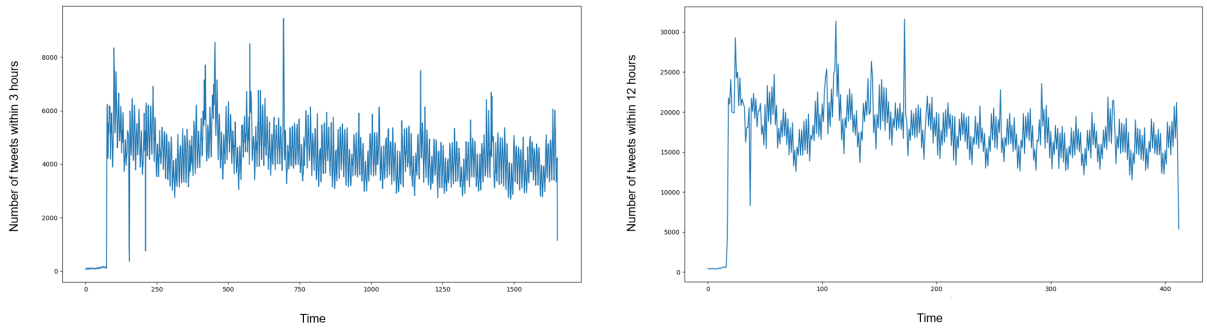
The last preprocessing step is the discretization of the dataset so that discrete hidden (semi-)Markov models can be used. Since discretization by itself is not desirable, because it is problem-dependent, we have considered a wide range of different discretization values: $\triangle t = \{1, 3, 4, 6, 12, 24, 48\}$ hours, which means that the observations ($O \equiv \{O_t\}_{t=1}^{n}$) are the number of tweets in each interval of 1 hour, 3 hours,

[28] Which can be found in `https://www.kaggle.com/alaix14/bitcoin-tweets-20160101-to-20190329`.

[29] The *URL* field is not shown because it is null for the two samples. The *tweet-id* is also omitted.

[30] This value was obtained after running a language model, available at `https://github.com/Mimino666/langdetect`, to recognize the language for each sentence.

and so on. In Fig. 4.9, we show the collected observations discretized by (a) 3 hours and (b) 12 hours. We verify that Fig. 4.9(b) is almost a version of Fig. 4.9(a) with bigger discretization units, thus higher counts. From Fig. 4.9, we observe that the data is not uniform.[31] The fluctuations in the data allow us to



(a) (b)

Figure 4.9: Tweets distribution for (a) 3 hours of discretization and (b) 12 hours of discretization. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

extract, using the models previously described (HMMs, HSMMs and LSTMs), useful patterns that carry information and can be used to make forecasts, i.e., to predict the evolution of the number of tweets in the future. As bitcoin is a relatively recent topic, the focus on this area is still on exponential growth. It is essential to add external information so that the predictions are more realistic. For example, in a given week, the number of tweets about bitcoin may drastically increase because there has been a drop in the price of the virtual currency, or vice versa. Since this information may be present in the text of the tweets, we focused on the *text* field to obtain external features for time-series prediction of bitcoin data.

4.4.1.3 Feature extraction

The features consist of a representation of each tweet sentence. This representation is a collection of important properties of the sentence that reflects its linguist meaning[32] and is compressed in a numerical format to be interpreted by machines. There are many techniques to obtain word representations. The simplest way is perhaps by using *term frequency-inverse document frequency* (TF-IDF) [84], an information retrieval technique that reflects the importance of a word in a document of a corpus. Usually, the corpus is simply a collection of sentences, where each sentence represents a document. The TF-IDF of a word is a numerical value obtained as the product of two information retrieval statistics: the *term frequency* (TF) and the *inverse document frequency* (IDF).

The term frequency accounts for the number of times a term (word) w occurs in a document d [85]. That is,

$$\text{TF}(w) = f_{w,d}, \tag{4.40}$$

where f represents numerical frequency. However, the TF by itself will emphasize highly frequent words, such as ”the” and ”of”, which do not add any meaning to the sentence. To counter this, the inverse document frequency measures the number of documents in which that same word w appears in the

[31] Observation statistics for a 3 hour discretization unit: $\min(O) = 65$, $\max(O) = 9442$, $\bar{O} \pm \sigma(O) = 4263.3345 \pm 1259.7274$.
[32] Including relations between words, syntax and semantics.

whole corpus [86]. That is[33],

$$\text{IDF}(w) = \log\left(\frac{N+1}{|d \in D : w \in d| + 1}\right) + 1, \tag{4.41}$$

for any document d in a collection of N documents called the *corpus* (D). Thus, IDF reduces the importance given to highly frequent words and increases the weight of words that rarely occur. In Table 4.10, we show the most frequent words selected from the bitcoin dataset according to their IDF value. Finally, the TF-IDF is computed as

$$\text{TF-IDF}(w) = \frac{\text{TF}(w)\text{IDF}(w)}{||\text{TF}(w)\text{IDF}(w)||_2},$$

where $||\text{TF}(w)\text{IDF}(w)||_2$ is the ℓ_2-norm of the product of the TF and IDF terms.

To sum up, TF-IDF is a simple statistic that can be used for word representation.

More complex representations can be obtained using *natural language processing* (NLP) algorithms. For example, *Word2Vec* [87] or *GloVe* [88] provide a vector representation for each distinct word in the vocabulary considered. These vectors are obtained from an artificial neural network (see Section 2.2.1.5) that learns word associations from a large corpus of text. The word associations include synonymous/antonymous relations that can be later used to report the level of semantic similarity between the words represented by the corresponding vectors.

However, since these methods (TF-IDF, Word2Vec and GloVe) are specialized in obtaining individual word embeddings for each word in a sentence, the global context of the sentence is not properly modeled. A more interesting and quite recent NLP technique developed by Google is called *BERT* [89], which uses transformers [90] to learn a bidirectional encoding representation of words. The pretrained[34] BERT model can be used to obtain word representations which have a richer meaning since they include the context of the sentence. For instance, the same word in TF-IDF, Word2Vec or GloVe has always the same vector representation no matter the context, whereas BERT provides contextualized embeddings (e.g. the word "running" in the sentence "He is running for president." has a different meaning than in "He is running 5km today.").

The BERT core-architecture consists of a bidirectional transformer. The transformer replaces the traditional LSTM in the sequential representation of a sentence by using solely attention - as in "Attention is all you need" [90].

Attention is the mechanism of connecting words through their meanings in a sentence. For example, in the sentence "We heard the dog bark", the words "dog" and "bark" have a very high cross-attention value.[35] BERT achieved state-of-the-art performance on a number of natural language processing and understanding (NLU) tasks, including General Language Understanding Evaluation (GLUE).

In this dissertation, we used sentence embedding features extracted from TF-IDF and BERT only. Then, we trained the HSMM-LR on both features, separately. The BERT sentence embedding[36] is a vector of

[33] Eqs. (4.40) and (4.41) are the TF and (smoothed) IDF versions, respectively, used in the context of this dissertation. There are other versions which are not covered here.

[34] BERT is available as a pretrained model, trained on a large plain text Wikipedia corpus.

[35] Aside from intra-sentence attention, there is also inter-sentence attention and self-attention mechanisms, all included in BERT.

[36] Obtained from `https://github.com/UKPLab/sentence-transformers`.

dimension 768 (per sentence), as illustrated in Fig. 4.10.

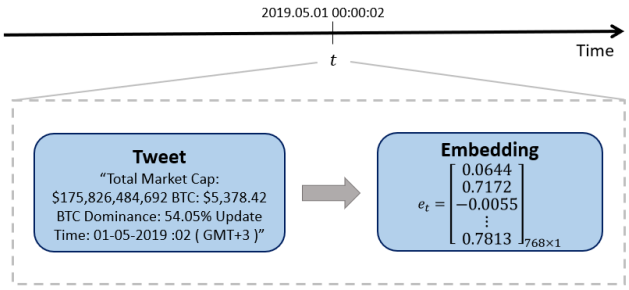


Figure 4.10: Example of a BERT sentence embedding.

IDF	Words
idf_1	*time*, *usd*, *2019*, *crypto*, *cryptocurrency*, *trx*, *bch*, *btc*, *btcusd*, *eth*, *market*, *bitcoin*, *xrp*
idf_2	*ltc*, *trading*, *price*, *update*
idf_3	*blockchain*

Table 4.10: TF-IDF most frequent words selected from the bitcoin dataset, where $idf_1 \approx 8.98 < idf_2 < idf_3$.

In what the TF-IDF features are concerned, it is reasonable to consider documents at the tweet level or the window ($\triangle t$) level. That is, either each tweet sentence is a document or all sentences belonging to each discretized interval are a document, respectively. We considered documents at the $\triangle t$ level. Thus, the IDF in Table 4.10 is proportional to the number of times those words appear in each window of 1h (1h is the discretization unit). The dimension of the TF-IDF features corresponds to the vocabulary extent, i.e., the total number of words in the corpus. Since this number was very high, we needed to cut off the number of features by term frequency, i.e., remove the less frequent words according to their IDF.

After obtaining the embedding of each tweet sentence, we still need to combine them somehow to obtain the embedding of each discretized window. Since concatenation is not an option, because we would end up with vectors with different dimensions per window, we have decided to average all embeddings inside a discretized window. Furthermore, to account for the effect of previous windows, we introduce another variable of discretization, $\triangle w = \{0, 1, 3, 5, 7\}$. A value of $\triangle w = 0$ means that the feature vector for time t ($\boldsymbol{c}_t$), contains only the average embedding from the current time window, whereas a value of $\triangle w = 7$ averages the embeddings of the previous 7 windows. To average windows, we use a weighted summation which gradually gives less weight to the average embedding of the farther back in time windows, according to an exponential factor. This process is illustrated in Fig. 4.11.

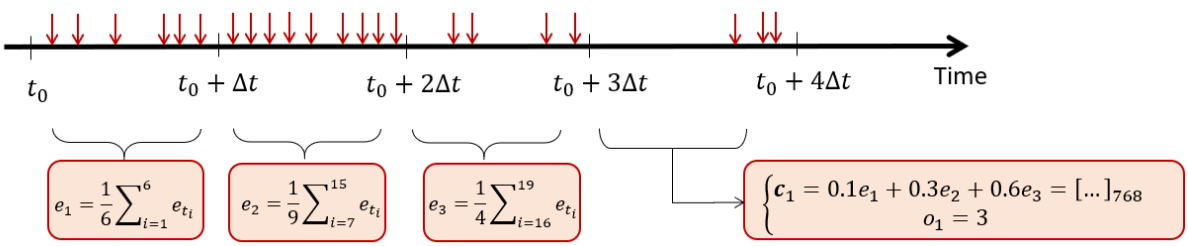


Figure 4.11: Process of obtaining the embedding of a window by weighted summation of the average embeddings of the previous windows. The red arrows represent tweet arrivals in time. $\triangle w = 3$.

In Fig. 4.11, t_0 is the start date, e_1 is the average embedding of the first window (with discretization unit $\triangle t$), which contains 6 tweets, e_2 is the average embedding of the second, window which contains 9

tweets, and e_3 is the average embedding of the third window which contains 4 tweets. Since we considered $\triangle w = 3$, the first sample of the feature vector ($\boldsymbol{c}_1$) is only obtained in the fourth window. The correspoding observation is $o_1 = 3$ (because the fourth window contains 3 tweets) and the feature vector is a weighted summation of the 3 previous average embeddings (e_1 with weight 0.1, e_2 with weight 0.3 and e_3 with weight 0.6). By continuing this sliding window process we obtain the whole feature vector $\{\boldsymbol{c}_t\}_{t=1}^n$ and observation sequence $\{o_t\}_{t=1}^n$.

Note that the higher the discretization and number of windows considered ($\triangle t$ and $\triangle w$, respectively), the more information we are averaging at the same time, which can lead to a signal close to noise.
Both TF-IDF and BERT features are normalized according to the z-score normalization

$$\frac{\boldsymbol{c}_z - \bar{\boldsymbol{c}}_z}{\sigma(\boldsymbol{c}_z)}, \qquad \boldsymbol{c}_z \equiv \{\boldsymbol{c}_{t,z}\}_{t=1}^n, \quad z \in [p], \tag{4.42}$$

where p is the total number of features. For example, for BERT[37], we obtain the mean and standard deviation for each of the $p = 768$ features individually, using only the feature vectors from the train set. Then, we normalize the feature vectors of all sets[38] by subtracting the training mean and dividing by the training standard deviation, according to eq. (4.42).

4.4.2 Experimental setup

Before explaining our experimental setup for time-series forecasting, we present some implementation details that need to be addressed.

Splitting the dataset into train, validation and test sets One implementation detail consists of how to split the dataset into a train set, a validation set, and a test set. The train set is used to train the model. The validation set is used to evaluate the model, during training, for hyperparameter tuning. Finally, the test set is used just for testing purposes. Since the dataset is sequential, we split the first 80% for training, then 10% for validation, and 10% for testing. For example, for a time discretization of $\triangle t = 1$ hour, the number of points in the train, validation (val) and test sets are: 3786, 473 and 474, respectively.
In each transition between sets (train/val and val/test), we discarded a number of samples equal to $\triangle w$, so that we avoid introducing a look-ahead-bias.[39]
For example, the TF-IDF features were extracted through a *TfIdfVectorizer* class from Scikit-learn open-source library [92]. The train set is used to learn the vocabulary and the IDF value of each word in the vocabulary. Then, the documents in the validation and test sets are transformed (from sentences to their TF-IDF value) according to the vocabulary and the IDFs found in the train set. This way we avoid look-ahead-bias.

[37] The source code for BERT can be found in [91]. Our adaptation can also be consulted at the following github URL: `https://github.com/filiparente/Predtweet`.
[38] Train, validation and test sets.
[39] A look-ahead-bias can occur, when splitting the dataset, if information about the future (e.g. some samples from the validation and test sets) is present in the train set. The trained model is biased by this information.

Removing initial low counts Another implementation detail concerns the initial low counts in Fig. 4.9. We removed them since we assume that the data collection process is responsible for these outliers.

Experimental description We are now able to detail the implementation of time-series forecasting for the Twitter dataset described in Section 4.4.1. We trained the following models: HMM, HSMM, HSMM-LR and LSTM (found in Sections 2.2.2.4, 2.2.2.5, 4.2.2 and 2.2.1.5, respectively). The Markov models were trained simultaneously, according to Algorithms 4 and 5, and evaluated according to Algorithm 6 (the algorithms can be found in Appendix D).

Algorithm 4 is the main of the forecasting program, and it starts by defining the setup, i.e., arguments such as the maximum and minimum number of states for pruning (k_{max} and k_{min}), the percentages of train, validation and test sets, the features (BERT/TFIDF or None), etc.

Then, we test each discretization (see Section 4.4.1) and we perform the sequential pruning strategy (explained in Algorithm 2, from Chapter 3, Section 3.1.3.2) separately for HMMs and HSMMs using the MMDL criterion (see Chapter 3, Section 3.1.3).[40] Since the features do not affect the observation sequence, we only prune when $\triangle w = 0$, and we use the same number of states for the subsequent values of $\triangle w$. The next time we prune again is when the discretization $\triangle t$ changes, and, instead of starting from the initial k_{max}, we start by the number of states chosen in the previous discretization plus 5 more states. The reason for this is to make the hyperparameter grid-search tuning of the discretization values a bit more efficient. Also, note that a lower discretization needs, in theory, less states to model the observations when compared to a higher discretization, since the signal is basically smoothed (see Fig. 4.9(b) vs. Fig. 4.9(a)).

After choosing the total number of states, we run Algorithm 5, where, first, we need to pre-process the data, which include: **1)** removing the initial low counts; **2)** splitting the dataset into sequential train, validation and test sets and **3)** normalizing the train set and using that normalization in the validation and test sets.

Then, we loop 10 times each Markov model to account for different initializations, since the EM algorithm is very initialization-dependent. Note that the initialization is the same for HSMM and HSMM-LR models, since the parameters are initialized randomly and only depend on the chosen number of states k^*_{model}, which is the same for both HSMM and HSMM-LR models. This loop is also used to obtain different values for the MSE of the predictions in the validation set, so that later we use the trained parameters from the iteration with lowest MSE as the best parameters, to evaluate and report errors in the test set, according to Algorithm 6 (see Appendix D).

It is important to highlight that two consequent predictions, with the same trained model, hold different MSE because of the Poisson variance. Therefore, in this case, we compute the MSE of the predictions in the validation set without Poisson fluctuations, i.e., we solely use the Poisson parameter $\boldsymbol{\lambda}$ and we do not sample from the Poisson distribution. This way, the difference between the MSE of the predictions is only due to the parameter initialization.

Another detail concerns the L-BFGS regularization parameter (φ) tuning. We chose to do a binary-

[40] The number of states chosen for the HSMM-LR model is the same as the HSMM.

search hyperparameter tuning between $[0, 1]$, and the criterion to evaluate the candidate parameter is the MSE of the predictions in the validation set, without Poisson fluctuations.

Finally, Algorithm 6 concerns the evaluation of the forecasting results, in order to obtain a report that contains the estimated parameters and estimation errors (e.g. mean-squared-error of the predictions). We compute 50 predicted sequences for the test set, here with Poisson fluctuations. Note that, in order to make forecasts, in the transitions between train/validation/test sets, we have access to the state posteriors from the end of train (validation), to make the first forecast (see Section 4.1) for the validation (test) set, and the last true observation from train (validation) is also available to adjust the first prediction. Now that we have detailed our experimental setup, we present the results, which are split into model order selection results, in Section 4.4.3, and forecasting results, in Section 4.4.4.

4.4.3 Model order selection

This section concerns the order selection of the HMM and HSMM, which are used to model and forecast future data from the Twitter dataset, described in Section 4.4.1. The HMM and HSMM are pruned separately, according to Algorithm 4 (see Appendix D). To do so, we use the following setup:

- $k_{max} = 50$ and $k_{min} = 5$ for pruning;
- BERT features;
- "Smart" initialization of the Poisson rates $\boldsymbol{\lambda}$, i.e., the observation set $\mathcal{V}$ is sorted and split into k columns, and we assign each λ_i to the mean of the i^{th} column;
- Geometric state duration distributions;
- All discretization units ($\triangle t = \{1, 3, 4, 6, 12, 24, 48\}$) and all window values ($\triangle w = \{0, 1, 3, 5, 7\}$);
- The logistic regression weights ($\boldsymbol{\beta}$) are learnt, in the HSMM-LR model, considering the setting where we feed each L-BFGS block a partial set of the feature vector (see Section 4.2.2).

Table 4.11: Pruning results for a discretization unit of 1, 3, 4 and 6 hours for all windows $\triangle w = \{0, 1, 3, 5, 7\}$. The pruning results include: the chosen optimal number of states for the HMM (k^*_{HMM}) and HSMM (k^*_{HSMM}) and the maximum state duration for the HSMM (d^*_{max}); $\hat{\varphi}$ is the chosen value for the regularization parameter of the L-BFGS at each combination $\triangle t, \triangle w$.

$\triangle t$ [h]	$\triangle w$	k^*_{HMM}	k^*_{HSMM}	d^*_{max}	$\hat{\varphi}$
1	0	36	14	20	0.6172
	1				0.9375
	3				0.5469
	5				1
	7				0.9961
3	0	33	12	19	0.9844
	1				0.9922
	3				0.9375
	5				0.5157
	7				0.75

$\triangle t$ [h]	$\triangle w$	k^*_{HMM}	k^*_{HSMM}	d^*_{max}	$\hat{\varphi}$
4	0	22	12	6	1
	1				1
	3				0.9375
	5				0.0079
	7				0.0079
6	0	23	10	5	0.5
	1				0.9922
	3				0.9844
	5				1
	7				1

The chosen number of states and maximum allowed duration are shown in Table 4.11, for the first four discretization units only. We also present the values chosen for the regularization parameter of the L-BFGS algorithm ($\hat{\varphi}$), found as a result of a binary-search tuning. The reason why we chose a

binary-search algorithm instead of the usual grid-search and random-search algorithms [93] is because hyperparameter tuning is a time-consuming task. Grid-search will find the optimal value for the hyperparameters (assuming they are contained in the grid) eventually, while random-search will usually find a good approximation in far fewer iterations [93]. Even though random-search is currently preferred over grid-search, a classic binary-search algorithm is usually faster than both, while achieving a performance close to random-search.[41]

From inspection of Table 4.11, we confirm that, in general, as the discretization unit increases, the optimal number of states (for both HMMs and HSMMs) decreases. Furthermore, one would expect that the maximum duration would increase, since, as the signal is smoother, there is a higher probability of staying longer in each state. However, the opposite happens, which may be due to the empirical estimation procedure of d_{max}, detailed in Chapter 3, Section 3.1, where we define d_{max} as the maximum duration for which no state duration probability is below $\epsilon = 0.001$. However, as the chosen total number of states decreases, all states are used and, possibly, the choice of ϵ is what is forcing a low d_{max}. One solution, as a suggestion of future work, is to change the value of ϵ according to the number of states. A more clever solution would be to have a different d_{max} per state. However, as mentioned in Chapter 3, Section 3.1, that would require a large adaptation to the forward-backward procedure.

It is also important to highlight, from Table 4.11, that the total number of states chosen by the HSMM is always smaller than the one chosen by the HMM. This is justified by the fact that, in HSMMs, the order selection criterion is more restrictive since it depends on both the number of states and the state durations.

In Figs. 4.12(a)(b), we present the evolution, with the number of states, of the MMDL criterion value (in blue) and the training log-likelihood (in red) for (a) the HMM and (b) the HSMM. In Figs. 4.12(c)(d), we show the prediction MSE of the validation set in blue, and train set in red, with the same X-axis for (c) the HMM and (d) the HSMM. From Fig. 4.12(a), we confirm that the optimal number of states chosen for the HMM ($k^*_{\text{HMM}} = 36$), corresponds in fact to the maximum value of the MMDL criterion (in blue), even though the maximum of the training log-likelihood (in red) is only achieved at $k = 50$. Likewise, in Fig. 4.12(b), the optimal number of states chosen for the HSMM is $k^*_{\text{HSMM}} = 14$. These plots correspond only to windows of 1h ($\triangle t = 1\text{h}$), and are in line with the first rows of Table 4.11.

We can state that, from Fig. 4.12(a), the pruning has a higher effect on the HMM, since, for a low number of states, the HMM training log-likelihood reaches values much lower than the HSMM. For example, in the last iteration (where $k = k_{min} = 5$), the HMM has a training log-likelihood of approximately -11×10^4, while the HSMM stops at approximately -3.3×10^4.

It is also important to note that the HSMM criterion value at the chosen k ($C_{k^*_{\text{HSMM}}} = -2.525 \times 10^4$) is better than the HMM ($C_{k^*_{\text{HMM}}} = -3.303 \times 10^4$), even though the criterion chooses 22 less states. However, this may be due to the initialization. Despite the fact that the sequential pruning strategy reduces the dependency of the EM estimates on the initialized parameters, the initialization may still affect the results, especially because we start the pruning strategy of the HMM and HSMM with different random initial parameters. One last comment on Fig. 4.12(b) regards the alternation of the points between consecutive

[41] Also, binary-search can be used initially to get a rough sketch of where the parameters lay inside the search space, and then grid or random-search methods can be used to further refine the hyperparameter prediction in a reduced search space.

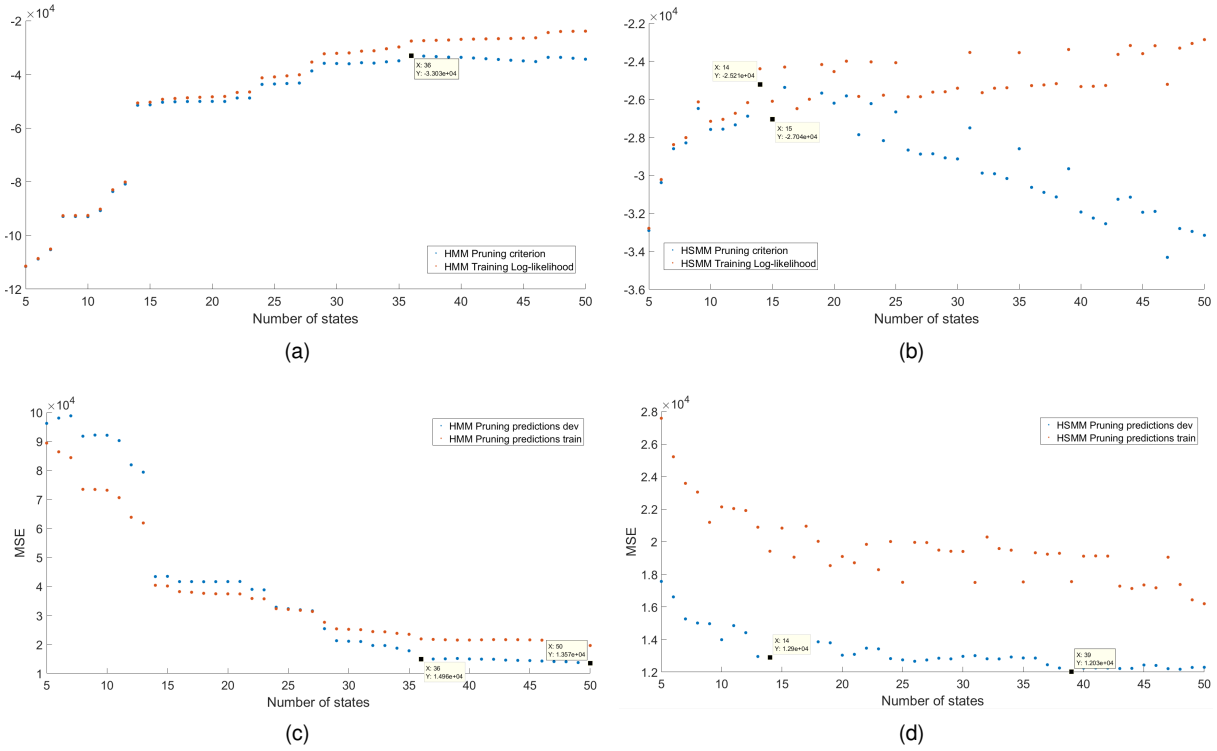


Figure 4.12: $\triangle t = 1$h, (a) Evolution, with the number of states, of the MMDL criterion value (in blue) and the training log-likelihood (in red) for the HMM; (b) Same as (a) but for the HSMM; (c) Evolution, with the number of states, of the prediction MSE of the validation set in blue, and train set in red, for the HMM; (d) Same as (c) but for the HSMM. The X-axis of all four figures represent each iteration of the pruning strategy, where the number of states starts at $k_{max} = 50$ and ends at $k_{min} = 5$. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

number of states, which also occurs for the HMM in Fig. 4.12(a), but it is not visible due to the scale. We believe that this behavior is related to the initialization of the EM parameters, which corresponds to the parameter estimates from the previous iteration with the least probable state pruned. One reason can be that we prune a state that does not correspond to the least probable state, i.e., the stationary probability distribution may not be well approximated. Another reason can be that, when the number of states is large, we prune a state from the parameters and, after normalization, the probability row of one parameter (e.g. the transition matrix) sums to zero. In that case, we set that row as uniform. Therefore, in the next iteration the EM estimates may be worse.

Regarding Figs. 4.12(c)(d), the prediction MSE is computed for the train and validation set without Poisson fluctuations, using the trained model, for each k, before pruning the least probable state. We can observe, from Fig. 4.12(c), that the HMM prediction MSE for the validation set with the chosen $k^*_{\text{HMM}} = 36$ is pretty close to the best prediction MSE at $k = 50$. There is no guarantee that the number of states chosen by the criterion leads to the lowest prediction errors, as it would be ideal. Recall that the criterion avoids overfitting by penalizing models with a large number of states. Likewise, for the HSMM in Fig. 4.12(c), the prediction error for the chosen $k^*_{\text{HSMM}} = 14$ is worse than for $k = 44$, which corresponds to the minimum. Note that even though the prediction errors for the HSMM are lower than for the HMM (with the chosen k), this may be again due to the initialization.

In Table 4.12, we present the results for multiple seeds instead of just one, for the 1h discretization unit. The multiple seeds allow us to assess how different random initialization of the parameters[42] affect the pruning results. We also tested for $k_{max} = 100$ and for all three selection criteria from Chapter 3.

Table 4.12: Pruning results for all three criteria (AIC, BIC and MMDL) using two sequential pruning strategies, one starting at $k_{max} = 50$ and the other at $k_{max} = 100$. The optimal number of states is k^*_{HMM} for the HMM and k^*_{HSMM} for the HSMM. The optimal maximum state duration of the HSMM is d^*_{max}. Results presented as mean and standard deviation from 10 different runs/seeds.

Criterion	k_{max}	k^*_{HMM}	k^*_{HSMM}	d^*_{max}
AIC	50	49.60 ± 0.70	29.00 ± 6.86	19.60 ± 1.58
	100	96.50 ± 1.35	26.30 ± 5.44	20.00 ± 2.11
BIC	50	45.30 ± 3.95	14.50 ± 2.01	19.60 ± 0.52
	100	69.80 ± 4.66	15.00 ± 2.36	19.90 ± 1.59
MMDL	50	45.40 ± 4.01	14.50 ± 2.01	19.60 ± 0.52
	100	69.80 ± 4.66	15.00 ± 2.36	19.90 ± 1.59

The first thing to notice in Table 4.12 is that the results for BIC and MMDL criteria are almost the same, which we also note in the synthetic results.

We can also observe, by comparing with the first rows of Table 4.11, that the results for MMDL criterion with $k_{max} = 50$ are similar with the exception of the HMM, that adds almost 10 more states. Another thing to notice is that the AIC criterion chooses always a lot more states than the BIC/MMDL criteria, since it penalizes less the model complexity. In fact, the chosen optimal number of states for the HMM, using the AIC criterion, is always close to the maximum number of states k_{max}.

Comparing the results from the two values of k_{max}, we conclude that the HMM is the only model for which the results are very different from $k_{max} = 50$ to $k_{max} = 100$, with the AIC criterion adding almost 50 states and the BIC/MMDL criteria adding almost 25 states.

The optimal maximum duration is surprisingly stable at around 20 duration units per state.

In Fig. 4.13, we show, in (a), (b) and (c), the criterion and the log-likelihood per iteration, i.e., from $k_{max} = 100$ to $k_{max} = 5$, of one sample from the 10 independent runs. In (a), we depict those results for the HMM, with the AIC criterion, while in (b) and (c) the results are for the HSMM with the MMDL criterion. Note that Fig. 4.13(b) is simply a projection of Fig. 4.13(c), without the axis of the chosen maximum allowed duration d_{max}. Finally, Fig. 4.13(d) presents the MSE predictions with errors bars [43] for the HSMM with the MMDL criterion.

From Fig. 4.13(a), we verify an abrupt jump at around $k = 35$, where both the training log-likelihood and the AIC criterion are reduced by approximately 65%. One possible reason for this phenomena is the removal of a least probable state that was capturing large outlier levels (see the high and low peaks in Fig. 4.9). By removing that state, the Poisson PMF of those peaks hinders the log-likelihood, because no level from $\boldsymbol{\lambda}$ can truly explain those observations.

Another thing to notice is that, with a larger k_{max}, the criteria penalization on the model complexity is more visible, since in both Figs. 4.12(a)(b)(c), there is a notorious drop on the criterion value after the

[42] Except the Poisson rates, which have a "smart" initialization that only depends on the train observation sequence.
[43] Computed as the standard deviation of the predictions for all 10 runs.

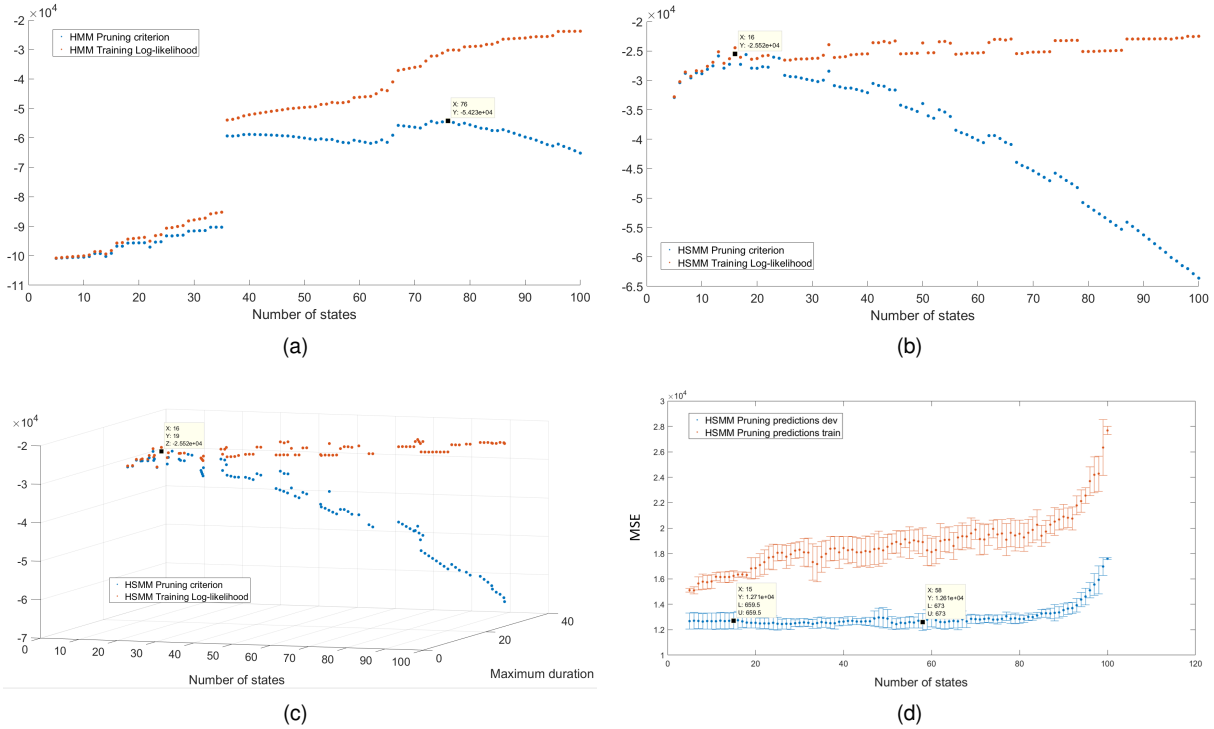


Figure 4.13: (a)(b)(c) One sample of the evolution, with the number of states, of the criterion value (in blue) and the training log-likelihood (in red) for (a) the HMM with the AIC criterion and (b)(c) the HSMM with MMDL criterion. Note that (c) is simply a 2D version of (b), without the durations, for a better visualization; (d) MSE of the prediction (with error bars) of the validation set (in blue), and train set (in red), for the HSMM with MMDL criteria. The pruning strategy starts at $k_{max} = 100$ and ends at $k_{min} = 5$. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

chosen k, especially for the HSMM.

Furthermore, the alternation between points previously identified can also be associated, for the HSMM, with different choice of durations (see Fig. 4.13(c)).

Finally, Fig. 4.13(d) has a different scope when compared to Fig. 4.12(d), since in the latter we present the results from only one run while in the former we average the prediction results of the 10 runs, for each k. However, in both, the training has worse predictions than the validation set. We observe that the average prediction MSE for the $k_{avg} = 15$, from Table 4.12, is close to the average minimum at $k = 58$.

After analyzing the pruning results, we are now able to explore in more detail the time-series prediction results in the next section. We note that the hyperparameters[44] used for the experiments in the next section are the ones from Table 4.11 with $\triangle t = 1$h, i.e., $k^*_{\text{HMM}} = 36$, $k^*_{\text{HSMM}} = 14$ and $d^*_{max} = 20$.

4.4.4 time-series prediction results

We present a set of experiments performed to assess the quality of the predictions. In the first experiment (Section 4.4.4.1), we compare different discretizations. In the experiments from Section 4.4.4.2 to Section 4.4.4.4, we compare different setups, while in Section 4.4.4.5, we compare BERT and TFIDF

[44]The hyperparameters include the number of states, k, the maximum allowed state duration, d_{max}, and regularization parameter of the L-BFGS, φ.

features. Finally, the aim of the last experiment, in Section 4.4.4.6, is to compare all Markov models and an LSTM in what regards the prediction accuracy, with the best setup, features and discretization found in the previous experiments.

4.4.4.1 Analysis of the effect of the discretization

In this first experiment, we test the Markov models (HMM, HSMM and HSMM-LR) in the forecasting task, using the same setup from Section 4.4.3. We want to assess how different combinations of $\triangle t$ and $\triangle w$ affect the forecasting results. Following that mindset, we compare the results using the mean normalized root mean-squared-error (MNRMSE) metric between the predictions and the true observations, computed as

$$\text{MNRMSE}(\hat{o}, o) = \frac{\text{RMSE}(\hat{o}, o)}{\bar{o}} = \frac{\sqrt{\frac{\sum_{t=1}^{n}(o_t - \hat{o}_t)^2}{n}}}{\bar{o}}. \tag{4.43}$$

The use of this particular metric is sustained by the fact that, when comparing results from distinct discretization units, we must account for the different observation range. With higher discretization units, the observations are also higher. In Tables 4.13 and 4.14, we present the results for this metric in the validation and test sets, for all Markov models considered and for a discretization unit of 1 and 3 hours, respectively. The results are presented as the mean and standard deviation of the MNRMSE metric of 50 predicted observation sequences for the respective trained model. Since the HMM and HSMM models do not use external features, the only model that depends on the number of windows ($\triangle w$) is the HSMM-LR.

From inspection of the tables, we observe that the prediction results worsen as the hourly discretization unit increases. For this reason, we do not show the results for the rest of the discretizations ($\triangle t = \{4, 6, 12, 24, 48\}$), and, in the next experiments, we will focus solely on the 1 hour discretization, which is the one that achieves the best results (lowest MNRMSE) among all discretizations.

Table 4.13: Results of the discretization experiment for the time-series prediction task in social media, with a 1 hour discretization unit. The metric shown is the MNRMSE (see eq. (4.43)) and the best values for the validation and test sets are identified in bold.

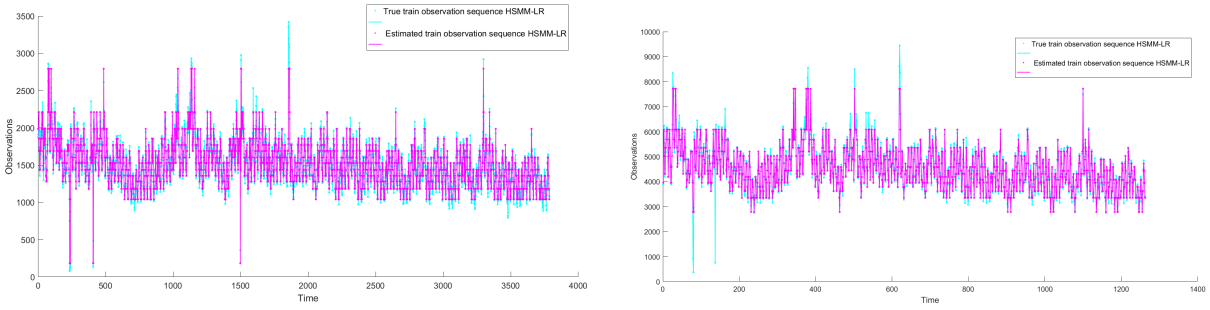
$\triangle t$ [h]	$\triangle w$	Set	Models HMM	HSMM	HSMM-LR
1	0	Val	0.0878 ± 0.0084	0.0873 ± 0.0111	$\mathbf{0.0872 \pm 0.0115}$
		Test	0.0864 ± 0.0086	0.0882 ± 0.0109	0.0861 ± 0.0110
	1	Val			0.0874 ± 0.0121
		Test			0.0873 ± 0.0115
	3	Val			0.0918 ± 0.0122
		Test			0.0860 ± 0.0123
	5	Val			0.0900 ± 0.0127
		Test			0.0856 ± 0.0118
	7	Val			0.0959 ± 0.0133
		Test			$\mathbf{0.0840 \pm 0.0123}$

The best results (lowest MNRMSE), identified in bold in Tables 4.13 and 4.14, are achieved by the

Table 4.14: Results of the discretization experiment for the time-series prediction task in social media, with a 3 hour discretization unit. The metric shown is the mean normalized root mean squared error and the best values for the validation and test sets are identified in bold.

$\triangle t$ [h]	$\triangle w$	Set	Models HMM	HSMM	HSMM-LR
3	0	Val			$\mathbf{0.1163 \pm 0.0123}$
		Test			$\mathbf{0.1047 \pm 0.0122}$
	1	Val			0.1354 ± 0.0126
		Test			0.1148 ± 0.0131
	3	Val	0.1341 ± 0.0101	0.1344 ± 0.0118	0.1588 ± 0.0144
		Test	0.1137 ± 0.0093	0.1121 ± 0.0106	0.1469 ± 0.0151
	5	Val			0.1535 ± 0.0142
		Test			0.1490 ± 0.0147
	7	Val			0.1632 ± 0.0135
		Test			0.1509 ± 0.0148

HSMM-LR model. For the 1 hour discretization, the best value for the validation set is achieved looking only at the features from the current window ($\triangle w = 0$), while, for the test set, the best value aggregates information from all seven previous windows ($\triangle w = 7$). Note that the same does not happen in Table 4.14. We believe that, at this point, we are averaging so much information that the feature vector is no longer informative. Another thing to notice is the discrepancy between the results from the validation and test sets, which is related to each signal. As one can observe in Fig. 4.15, the validation set (Fig. 4.15(a)) has some high peaks which are more difficult to estimate when compared to the test set (Fig. 4.15(b)). In Fig. 4.14, we show the training results for the HSMM-LR model with (a)$\triangle t = 1$, $\triangle w = 7$ and (b) $\triangle t = 3$, $\triangle w = 0$.[45] The results from Fig. 4.14 indicate that the less discretized dataset (Fig. 4.14(a))



(a) (b)

Figure 4.14: Training results for the HSMM-LR model with (a)$\triangle t = 1$, $\triangle w = 7$ and (b) $\triangle t = 3$, $\triangle w = 0$. The cyan line represents the true observation sequence while the magenta line represents the estimated observation sequence. The dots are the (discrete) observations. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

allows the model to better capture the outlier peaks, which can also be justified by the (higher) length of the training data. In Fig. 4.15, the prediction results for the (a) validation and (b) test sets are presented, both trained with the HSMM-LR model, while (c) and (d) are the results for the validation set only, trained with HMM and HSMM, respectively.

[45] Which are the setup that achieved lowest MNRMSE for $\triangle t = 1$h and $\triangle t = 3$h, respectively.

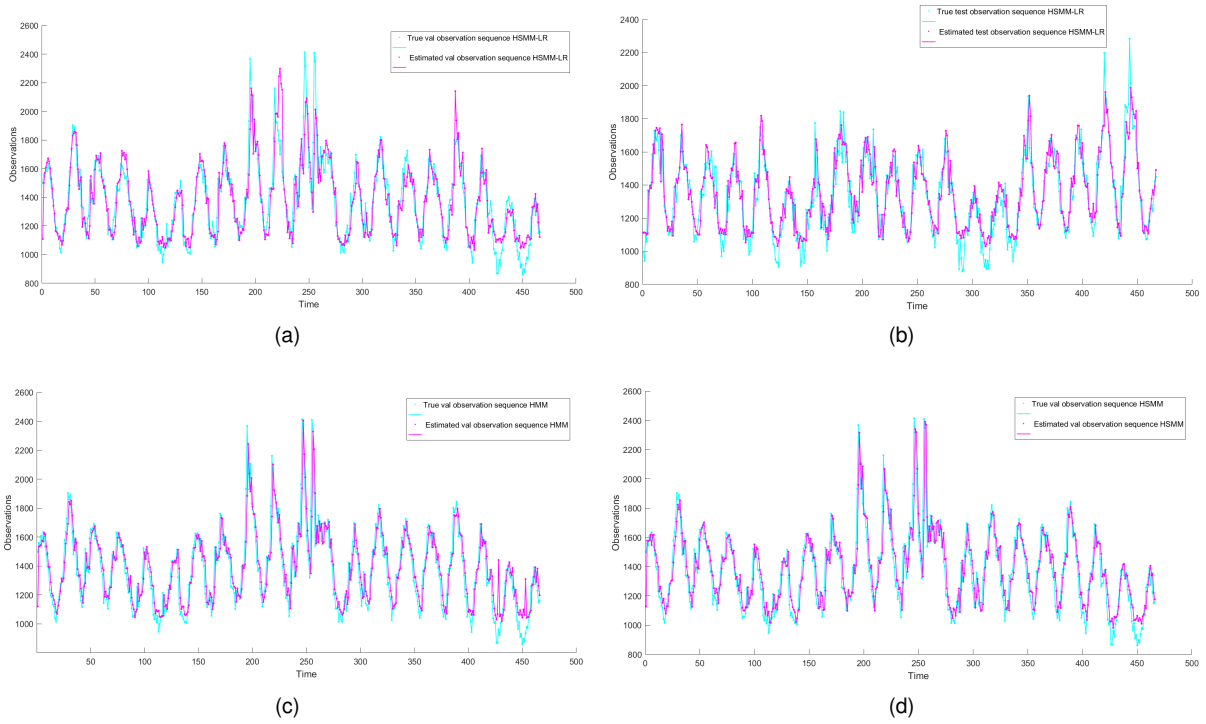


Figure 4.15: Prediction results with $\triangle t = 1$, $\triangle w = 7$ for the (a) validation set and (b) test set with model HSMM-LR; and validation set with models (c) HMM and (d) HSMM. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

Despite the notorious similarity between the HMM and HSMM results in Fig. 4.15(c) and Fig. 4.15(d), respectively, the HMM predicts an observation sequence with two abnormal peaks around $t = 425$ and $t = 450$, which compromise the MNRMSE metric. Another metric that is more sensitive to lower prediction errors and penalizes less those rare large errors is the *mean-absolute-error* (MAE), computed as

$$\text{MAE}(\hat{o}, o) = \frac{1}{n} \sum_{t=1}^{n} \mid o_t - \hat{o}_t \mid . \tag{4.44}$$

Results from this metric can be found in Table 4.15.

One can also observe that the observation sequence predicted by the HSMM-LR has more oscillations when compared to the one from the HMM and the HSMM.

Note that the signal granularity is controled by the total number of states. More states imply more Poisson rates, which may in turn cover a wider range of ”activity levels”, i.e., the counts of the total number of tweets within a window. Since the total number of states chosen by the HMM (with any of the criteria from Section 4.4.3) is always higher than the number of states chosen by the other models, we would expect to observe a higher signal granularity. Instead, we believe that the HMM, to predict the validation and test sets, is wasting some states.

Still regarding the HSMM-LR predictions, we note that the model performs poorly in capturing some high peaks and counts below 1100, even though in training (Fig. 4.14(a)) we observe that the model can properly express observations above and below that range. We can think of many reasons behind this phenomena. The first is that the external features may be forcing a state transition to a higher

Table 4.15: Weighted average (WA) and maximum-a-posterior (MAP) prediction results for all three models, two metrics MNRMSE (in eq. (4.43)) and MAE (in eq. (4.44)), for the validation and test sets of the dataset with discretization: $\triangle t = 1$, $\triangle w = 7$. The best values are identified in bold.

Metric	Model	Set	Criteria	
			WA	MAP
MNRMSE	HMM	Val	0.0881 ± 0.0087	0.1042 ± 0.0154
		Test	0.0864 ± 0.0085	0.0956 ± 0.0166
	HSMM	Val	$\mathbf{0.0876 \pm 0.0103}$	0.0956 ± 0.0151
		Test	0.0883 ± 0.0113	0.0970 ± 0.0168
	HSMM-LR	Val	0.0960 ± 0.0132	0.0959 ± 0.0157
		Test	$\mathbf{0.0839 \pm 0.0124}$	0.0934 ± 0.0155
MAE	HMM	Val	$\mathbf{88.78 \pm 0.48}$	101.55 ± 1.28
		Test	89.97 ± 0.57	97.99 ± 1.60
	HSMM	Val	89.51 ± 0.80	97.47 ± 1.53
		Test	91.79 ± 0.79	99.41 ± 1.69
	HSMM-LR	Val	92.88 ± 1.11	99.38 ± 1.52
		Test	$\mathbf{86.54 \pm 0.96}$	99.48 ± 1.59

λ. However, since we can observe that low counts are also not well captured by the other models, we discard this option. A more obvious reason is that the set of estimated Poisson rates ($\hat{\boldsymbol{\lambda}}$) has no other level closest to the true observations. In eq. (4.45), we show the Poisson rates estimated by the HSMM-LR (for $\triangle t = 1$ and $\triangle w = 7$):

$$\hat{\boldsymbol{\lambda}} = \Big[183.9, 1041, 1181.6, 1279.7, 1356.9, 1437.6, 1528.4, 1602.4, 1689.9, 1770.5, 1854.9, 1986.5, 2210.8, 2793.1\Big]. \tag{4.45}$$

We verify, from eq. (4.45), that there is a large gap between $\hat{\lambda}_1$ and $\hat{\lambda}_2$. During training, the model captures the lowest counts with $\hat{\lambda}_1 = 183.9$ and, since there are no sufficient counts in between 200 and 1000 and given the fact that the total number of states is 14, the next estimated level is $\hat{\lambda}_2 = 1041$. Adding the fact that the variance of the Poisson process corresponds to its own parameter λ, it is clear why the prediction is distorted.

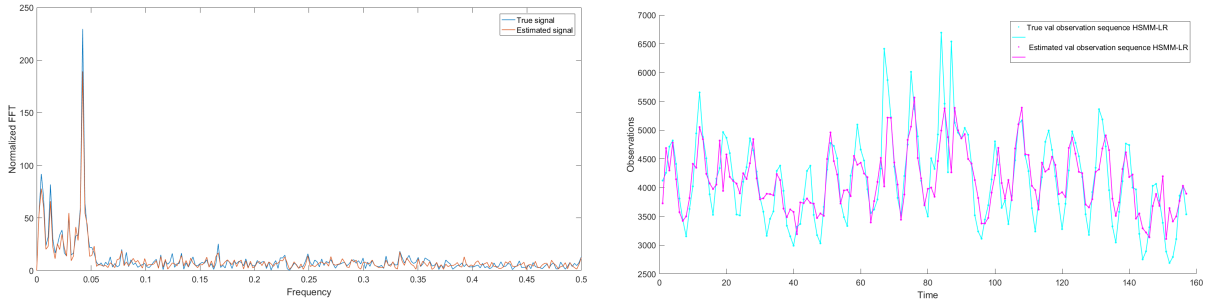
Following the same reasoning, we have also tested how the results change if the predictions are computed as a MAP prediction (see eq. (4.14)) instead of a weighted average of state posteriors (see eq. (4.15)). The results are presented in Table 4.15, where we can observe that the maximum-a-posteriori predictions perform worse than the weighted average predictions. We confirm that the MAE is less sensitive to large errors, since in the validation set, for the HSMM-LR model, the MNRMSE gives a slightly higher score to the MAP prediction, while in the MAE metric the WA is over the MAP prediction by an additive factor of 7.

In Fig. 4.16, we show the evolution of the predicted signal and the true signal in frequency, by computing the normalized fast Fourier transform (FFT) of the time signal. The FFT analysis is important to assert if the overall scope and temporal evolution of the original signal is being well modeled, while a slight phase drift is what is causing predictions errors. In Fig. 4.16, we show only the magnitude of the FFT.

However, the errors in magnitude and phase are presented in Table 4.16.

Table 4.16: Magnitude and phase errors of the FFT, measured through the SNRMSE[46]metric, for the validation and test sets of the datasets with discretizations $\triangle t = 1$, $\triangle w = 7$, for all three models, and $\triangle t = 3$, $\triangle w = 0$ only for the HSMM-LR model. The best values are shown in bold.

Metric	Component	Set	$\triangle t : \triangle w$			
			$1:7$			$3:0$
			HMM	HSMM	HSMM-LR	HSMM-LR
SNRMSE	Magnitude	Val	0.0586	**0.0575**	0.0659	0.0945
		Test	0.0624	0.0644	**0.0588**	0.0782
	Phase	Val	0.0386	0.0396	0.0393	**0.0077**
		Test	0.0435	0.0415	0.0422	**0.0083**



(a) (b)

Figure 4.16: (a) FFT of the true (in orange) and predicted (in blue) signals for the dataset with $\triangle t = 1$, $\triangle t = 7$. The predicted signal was obtained from the trained HSMM-LR model. The spacing between frequencies in the X-axis corresponds to the Nyquist frequency (i.e., half of the sampled frequency). The normalized FFT in the Y-axis is obtained as $2 \times \text{FFT}(o)/n$, where o is the (true or predicted) observation sequence and n is the number of samples. The DC component was removed from the resulting signal for a better visualization; (b) Temporal prediction results of the HSMM-LR model with $\triangle t = 3$, $\triangle w = 0$ for the validation set. The cyan line represents the true observation sequence while the magenta line represents the estimated observation sequence. The dots are the (discrete) observations. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

The metric in Table 4.16 is the SNRMSE[46] metric, computed similarly to the MNRMSE in eq. (4.43), but normalized with the standard deviation of the observation sequence instead of the mean.

From Table 4.16, we observe that, for the discretization of 1 hour, all three models have similar magnitude and phase results. On the contrary, the magnitude error for the 3 hour discretization is much higher, while the phase error reduces. To support these statements, we present, in Fig. 4.16(b), the temporal evolution of the original and predicted signals for the 3 hour discretization, from where we can observe that the observation levels are not being well captured, but there is no relevant phase drift.[47]

From inspection of Fig. 4.16(a), we observe that the magnitudes of both the original and predicted signals almost overlap, which confirm our expectations: the temporal evolution of the original signal (for $\triangle t = 1$h) is being well modeled, therefore, the reason behind the prediction errors bears in the phase.

[46] $\text{SNRMSE}(\hat{o}, o) = \dfrac{\sqrt{\frac{\sum_{t=1}^{n}(o_t - \hat{o}_t)^2}{n}}}{\sigma(o)}$.

[47] Note that the phase drift in Fig. 4.15 is not visible due to the X-axis scale (larger sequence length).

4.4.4.2 λ initialization: "Smart" vs. random

In this experiment, the setup is the same as in the model order experiment (see Section 4.4.3), with the exception that the Poisson rates are not "smartly" initialized, but randomly: $\lambda = u \times \max(\mathcal{V})$, where $u \sim \mathcal{U}([0, 1])$ is a sample from a uniform distribution between 0 and 1.

Table 4.17: Results of the $\boldsymbol{\lambda}$ initialization experiment for the time-series prediction task in social media, with a 1 hour discretization unit. The metric shown is the MNRMSE (see eq. (4.43)).

$\triangle t$ [h]	$\triangle w$	Set	Models HMM	HSMM	HSMM-LR
1	0	Val			0.0912 ± 0.0122
		Test			0.0951 ± 0.0136
	1	Val			0.0897 ± 0.0120
		Test			0.0877 ± 0.0143
	3	Val	0.0905 ± 0.0111	$\mathbf{0.0851 \pm 0.0111}$	0.0879 ± 0.0109
		Test	0.0891 ± 0.0111	0.0887 ± 0.0116	0.0897 ± 0.0115
	5	Val			0.0870 ± 0.0114
		Test			$\mathbf{0.0868 \pm 0.0115}$
	7	Val			0.0880 ± 0.0120
		Test			0.0919 ± 0.0131

The prediction results are shown in Table 4.17. Comparing Tables 4.13 and 4.17, we observe that overall the random initialization gives worse prediction results when compared to the "smart" initialization. This should come as no surprise, given that the "smart" initialization is obtained by considering a piecewise Poisson signal, with as much levels as the number of states, where the level value is computed from the observations. This is a reasonable approximation for the MLE of the Poisson rates.

4.4.4.3 Logistic regression input comparison: all features vs. a subset

In this experiment, the setup is the same as in the model order experiment (see Section 4.4.3), with the exception of the method to learn the logistic regression (LR) weights. In the previous experiments, the LR weights were learnt, in the HSMM-LR model, by feeding each L-BFGS block a partial set of the feature vector (see Section 4.2.2). Given that this approach depends significantly on the initialized parameters of the EM algorithm, we have also considered, in Section 4.2.2, a different approach, where we feed each L-BFGS block the whole features and transition probabilities as labels. The prediction results are shown in Table 4.18, for the HSMM-LR, since it is the only model that considers external features to model and predict data. Comparing Tables 4.13 and 4.18, we observe that this approach leads to worse predictions, even though the previous method was more dependent on the initialization.

4.4.4.4 State duration comparison: geometric vs. multinomial distributions

In this experiment, the setup is the same as in the model order experiment (see Section 4.4.3), but the state durations are multinomial instead of geometric distributions. The prediction results are shown in Table 4.19.

Table 4.18: Results of the logistic regression input comparison experiment for the time-series prediction task in social media, with a 1 hour discretization unit. The metric shown is the MNRMSE (see eq. (4.43)).

$\triangle t$ [h]	$\triangle w$	Set	HSMM-LR
1	0	Val	0.1101 ± 0.0138
		Test	0.0955 ± 0.0127
	1	Val	$\mathbf{0.1137 \pm 0.0115}$
		Test	$\mathbf{0.0950 \pm 0.0122}$
	3	Val	0.1282 ± 0.0140
		Test	0.0992 ± 0.0117
	5	Val	0.1443 ± 0.0148
		Test	0.1084 ± 0.0133
	7	Val	0.1499 ± 0.0170
		Test	0.1090 ± 0.0145

Table 4.19: Results of the state duration comparison experiment for the time-series prediction task in social media, with a 1 hour discretization unit. The metric shown is the MNRMSE (see eq. (4.43)).

$\triangle t$ [h]	$\triangle w$	Set	Models		
			HMM	HSMM	HSMM-LR
1	0	Val			0.0880 ± 0.0116
		Test			0.0879 ± 0.0134
	1	Val			0.0900 ± 0.0129
		Test			0.0888 ± 0.0128
	3	Val	$\mathbf{0.0872 \pm 0.0089}$	0.0880 ± 0.0120	0.0929 ± 0.0124
		Test	0.0863 ± 0.0086	0.0881 ± 0.0113	0.0864 ± 0.0129
	5	Val			0.0911 ± 0.0114
		Test			0.0840 ± 0.0127
	7	Val			0.0955 ± 0.0134
		Test			$\mathbf{0.0828 \pm 0.0127}$

Comparing Tables 4.13 and 4.19, we observe that the prediction results are quite similar with multinomial or geometric duration distributions.

We want to highlight that all previous experiments were also made for TFIDF features. We do not present the results since the conclusions are similar and due to space constraints.

4.4.4.5 Feature comparison: BERT vs. TF-IDF

In the present experiment, we compare the BERT embedding features with the TF-IDF, in particular, by varying the length of the TF-IDF features (i.e., vocabulary extent) and analyzing the effect on the prediction results. The setup is the one that, according to the previous experiments, gives the lowest MNRMSE, which corresponds to the setup from the first experiment (see Section 4.4.4.1), with "smart" initialization for the rates of the Poisson processes and geometric state duration distributions. Given that we want to evaluate the feature engineering, we only trained the model that depends on the features (HSMM-LR).

Since the BERT embeddings have a fixed length of $p = 768$, we started by extracting TF-IDF features with a $p = 768$ dimension, and we continued testing for smaller dimensions: $p = \{768, 512, 256, 128, 64\}$.[48]

[48] Recall that we reduce the feature dimension by cutting off features according to their IDF value.

In Table 4.20, we show the prediction results of the trained HSMM-LR model, using the TF-IDF features with different lengths, for $\triangle t = 1$h.

Table 4.20: Results of the feature comparison experiment for the time-series prediction task in social media, with a 1 hour discretization unit ($\triangle t = 1$). The metric shown is the MNRMSE (see eq. (4.43)).

$\triangle w$	Set	Number of features				
		768	512	256	128	64
0	Val	0.1083 ± 0.0132	0.1320 ± 0.0156	$\mathbf{0.1060 \pm 0.0128}$	0.1189 ± 0.0129	0.1231 ± 0.0150
	Test	0.1134 ± 0.0143	0.1328 ± 0.0152	$\mathbf{0.1100 \pm 0.0127}$	0.1272 ± 0.0139	0.1132 ± 0.0141
1	Val	0.1133 ± 0.0141	$\mathbf{0.1027 \pm 0.0135}$	0.1342 ± 0.0159	0.1249 ± 0.0148	0.1452 ± 0.0158
	Test	$\mathbf{0.0998 \pm 0.0127}$	0.1050 ± 0.0129	0.1276 ± 0.0145	0.1256 ± 0.0145	0.1269 ± 0.0139
3	Val	0.1198 ± 0.0139	$\mathbf{0.1119 \pm 0.0136}$	0.1467 ± 0.0168	0.1190 ± 0.0154	0.1235 ± 0.0143
	Test	$\mathbf{0.1022 \pm 0.0132}$	0.1099 ± 0.0140	0.1272 ± 0.0155	0.1080 ± 0.0135	0.1179 ± 0.0140
5	Val	0.1150 ± 0.0151	$\mathbf{0.1032 \pm 0.0147}$	0.1305 ± 0.0154	0.1126 ± 0.0134	0.1328 ± 0.0154
	Test	$\mathbf{0.1043 \pm 0.0137}$	0.1080 ± 0.0141	0.1234 ± 0.0151	0.1174 ± 0.0160	0.1271 ± 0.0169
7	Val	0.1181 ± 0.0141	$\mathbf{0.1032 \pm 0.0140}$	0.1132 ± 0.0134	0.1238 ± 0.0143	0.1277 ± 0.0154
	Test	0.1089 ± 0.0121	$\mathbf{0.1023 \pm 0.0145}$	0.1086 ± 0.0147	0.1176 ± 0.0149	0.1320 ± 0.0152

Comparing Tables 4.13 and 4.20, for the HSMM-LR model only, it is clear that the TF-IDF features are less informative than the BERT features, since the prediction results are much worse in this case. Although it is interesting to draw such a conclusion, given the powerfulness of the BERT embeddings in a variety of NLP tasks, we must note that the results are not fully comparable.

The BERT features, with 768 dimensions, are already sentence embeddings, extracted from our corpus but pre-trained on a larger corpus. In fact, the BERT large[49] is trained on plain text corpus from English Wikipedia with 2500M words.

On the other hand, TF-IDF features with a 768 dimension can be extracted from a large corpus, but the vocabulary size is only 768, which means that the vocabulary only contains the 768 most frequent words in our corpus.

It is clear the massive disadvantage that TF-IDF features have when compared to BERT embeddings, since a reasonable vocabulary size would contain, for example, at least 10000 words (we are already selecting only the most frequent ones). However, with such a large number of features, the estimation of the LR weights through the L-BFGS becomes extremely slow.

This is the biggest disadvantage of the HSMM-LR model when compared to neural networks for sequential data, such as the LSTM introduced in Section 2.2.1.5. While, with less features, this model is trained faster than LSTMs, with similar results (see the experiment in Section 4.4.4.6), when the number of features is very high, the model is not scalable.

4.4.4.6 Forecasting results: LSTM vs. parametric Markov models

The present experiment is similar to the experiment for synthetic data in Section 4.3.6, where we analyze how the amount of training data affects the prediction accuracy, with the difference that here the initialization of the parameters is unsupervised. Furthermore, we held-out the last 10% of the data for

[49] There are two pre-trained versions of BERT: the BERT base, which uses the BooksCorpus with 800M words, and the BERT large, the one used in the context of this dissertation.

the validation and test sets (5% for each). The model is trained on different percentages of the training data, ranging from 90% to 5%,[50] which, for the 1 discretization unit, corresponds to the following number of training samples: $\{4260, 3787, 3314, 2840, 2367, 1894, 1420, 947, 474, 237\}$, for $\triangle t = 1$ and $\triangle w = 0$. In this experiment, we compare the Markov models with and without features (HSMM-LR vs. HMM and HSMM) and the LSTM without features.

LSTM architecture In Fig. 4.17, we show a sketch of the architecture of our LSTM.

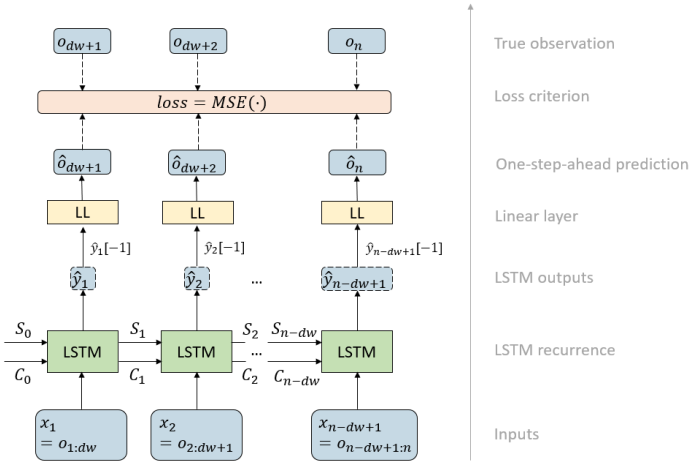


Figure 4.17: Our LSTM architecture, where x represents the inputs, S and C are the LSTM hidden and cell states, respectively, $\hat{y}$ is the LSTM output, and o and $\hat{o}$ are the true and predicted observations, respectively.

The inputs of the LSTM are sliding windows of the 1 hour discretized dataset ($\triangle t = 1$) without features, i.e., the first input x_1 corresponds to the true observations from $t = 1$ to $t = d_w = 24$, the next input is x_2 which corresponds to the previous input shifted one unit to the right, i.e., $x_2 = o_{2:25}$ and so on until all the dataset is fed into the LSTM.

The sliding windows are needed for one-step-ahead time-series forecasting in LSTMs. The use of $d_w = 24$ has only to do with our dataset. Given that each timestep represents a 1h interval, we have considered that sliding windows of dimension 24 were sufficient, since the LSTM manage to extract information even more far back by itself (through the hidden and cell states).

Furthermore, the observations, before being fed to the LSTM, are normalized using a MinMax scaler between $[-1, 1]$, fit to the training data. The validation and test sets are normalized by transforming the observations through the scaler trained using only the training data, to avoid look-ahead-bias.

We now describe the two main blocks of Fig. 4.17: the LSTM and the *Linear Layer* (LL).

The LSTM has an input size of 1, since we are not considering features, only twitter counts.[51] We considered a one-layer LSTM with a hidden layer size of 128, i.e., 128 nodes per layer.

Regarding the LL, it has an input dimension of 128 and an output size of 1.

The LSTM acts as an encoder, by upsampling the dimension of the inputs, i.e., in each timestep, we input $d_w = 24$ consecutive observations. The LSTM augments this dimension by increasing the size of

[50] The remaining (initial) parts of the observation sequence are discarded.

[51] For example, if BERT features were used, the embedding dimension (and input size of the LSTM) would be 768.

the feature space from 1 to 128. Then, it outputs a matrix of dimension 24 by 128, from where we will use only the last vector of dimension 128. We feed that vector through the linear layer[52], which acts as a decoder, since it transforms the LSTM output into a one-step-ahead prediction. Then, the linear layer weights are learnt along with the LSTM parameters so as to minimize the loss, which corresponds to the mean-squared-error between the LL output (the one-step-ahead prediction) and the true observation, as shown in Fig. 4.17.[53]

To sum up, the LSTM architecture predicts a single next observation for each sliding window input. During training, we evaluate the LSTM parameters on the validation set and, after training, on the test set. To evaluate, we use the last 24 true observations from the train set plus the initial hidden state, which acts like the state posteriors for the one-step-ahead prediction in parametric Markov models, described in Section 4.1. This allows us to predict the first observation from the validation set. Then, we append the true observation value and we shift the sliding window by one to the right, thus predicting the second observation from the validation set, and so on. The same procedure is done for the test set.

We note that, in Fig. 4.17, the parameters of the Linear Layer and the LSTM are shared for all timesteps. The initial hidden and cell states, S_0 and C_0, are initialized as random parameters that must be learnt along with the other LSTM parameters. This way, we guarantee that the dependence between sliding windows is included in the information passed by the hidden states.

We train the LSTM for 200 epochs, using full data (i.e., no batches) since it can all fit into memory.[54] We run the LSTM 10 times to account for the different random weights initialization, and we average the results. We do a backward/update pass at each iteration, and we clip the gradients according to a max gradient norm of 1.0, in order to avoid gradient exploding problems (see Section 2.2.1.5).

Ideally, the Markov models should have better estimation accuracy than the LSTM when less training data is available, and, as the amount of data increases, the LSTM is more likely to make better predictions. In general, artificial neural networks with enough training data are always better than any parametric model. In the present experiment, we want to find the lower bound, i.e., the number of training data for which the parametric Markov models surpass the LSTM in terms of prediction accuracy. We also assess what is the best model in each regime.

In Fig. 4.18, we present a graphical visualization of the MAE prediction errors, along with error bars, for the different percentages of training data. Figs.(a)(b) show the errors on the validation set, while Figs.(c)(d) refer to the test set. Moreover, Figs.(a)(c) are just a simplified version of Figs.(b)(d), without the HSMM-LR model for $\triangle t = 1$ and $\triangle w = \{1, 3, 5, 7\}$.

[52] The vector corresponds to the features of the linear layer model.

[53] We use the Adam optimizer with a learning rate of 0.001 (default).

[54] After discretization, and without features, the dataset is relatively small.

Figure 4.18: Evolution of the MAE prediction errors with different percentages of training data for the (a)(b) validation set and (c)(d) test set. The models considered in (a)(c) are: HMM, HSMM, HSMM-LR and LSTM for $\triangle t = 1$ and $\triangle w = 0$, whereas in (b)(d) are: HMM, HSMM and LSTM, all for $\triangle t = 1$ and $\triangle w = 0$ and HSMM-LR for $\triangle t = 1$ and $\triangle w = \{0, 1, 3, 5, 7\}$. Note that (a)(c) are simplified versions of (b)(d) for a better visualization. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

The X-axis in Fig. 4.18 are the number of training samples. The Y-axis are the MAE prediction errors (see eq. (4.44)).

We can observe that the LSTM achieves the lowest MAE prediction errors when we train with 1420 and 2840 data samples, in the validation set, and 1894 and 2367, in the test set. We note that, in the test set, the LSTM is the second worst model, only after the HSMM, when considering 4260 training data samples.

It is also interesting to notice that the HSMM-LR is usually better than the HMM and HSMM, but with less training data, the logistic regression weights are not properly estimated from the features, which worsens the predictions.

Regarding the HSMM-LR model, we note that the best window for the test set is $\triangle w = 3$, which is halfway in between of not averaging anything and averaging too much information.

We can also state that the HSMM has, in general, worst predictions than the HMM, which may be due to the discretization unit.

However, we note, by observing the Y-axis scale, that the variations in the MAE prediction errors between all models are not significantly relevant.

In Fig. 4.19, we show some examples of the temporal evolution of the true and predicted observation

sequences for the validation set with 2840 training samples (in Figs. 4.19(a)(b)), and the test set with 237 training samples (in Figs. 4.19(c)(d)). While Fig. 4.19(a) refers to model HSMM-LR with $\triangle t = 1$ and $\triangle w = 7$, Fig. 4.19(b) corresponds to the LSTM without features. Regarding Figs. 4.19(c)(d), the first corresponds to the HMM and the second to the LSTM, both without features.

From Fig. 4.18(b), for the validation set, we observe that the HSMM-LR 1.7 has a higher MAE than the LSTM 1.0 for 2840 samples, which is confirmed by comparing the predictions from Figs. 4.19(a)(b).

On the other hand, from Fig. 4.18(d) for the test set, we observe that the HMM has the lowest MAE for 5% of training data (which corresponds to 237 samples). In Figs. 4.19(c)(d), we compare the predicted observation sequence between the HMM and the LSTM, from which it is clear that the HMM prediction is closer to the real one, even though some high peaks are not well captured.[55]

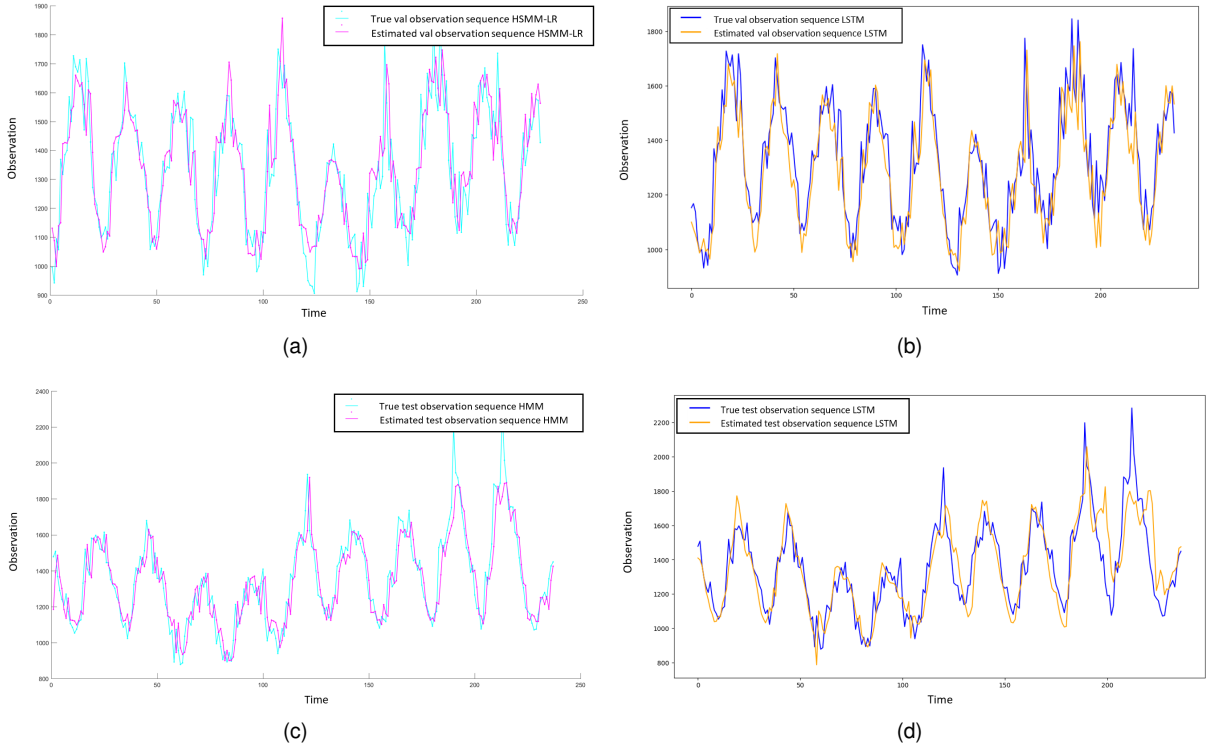


Figure 4.19: Prediction results for the (a)(b) validation set with 2840 training samples and (c)(d) test set with 237 samples. The models are: (a) HSMM-LR for the dataset with $\triangle t = 1$, $\triangle w = 7$; (b) LSTM model (c) HMM model and (d) LSTM model, all three for the dataset with $\triangle t = 1$, $\triangle w = 0$. Better seen online at `http://web.ist.utl.pt/ist181324/tese/`.

[55] Since the MAE metric does not penalize much those large rare errors, this is not reflected in the result.

Chapter 5

Conclusions

5.1 Summary of contributions

This dissertation adds two main contributions. The first regards the proposal of a context based HSMM, named HSMM-LR, which conditions state transitions on external features through logistic regression. The forecasting results indicate that, even though LSTMs are very popular nowadays, parametric models (such as HMM, HSMM and HSMM-LR) should not be underestimated as they can achieve close or even better prediction accuracy (with less training data), while remaining easy to train and more interpretable. Moreover, the parametric Markov models addressed in this dissertation have way less parameters and hyperparameters to estimate and tune, and are still able to capture a lot of data patterns. Furthermore, these models are trained faster than any artificial neural network model.[1]
However, in what the disadvantages are concerned, it is more difficult to extend or develop an end-to-end implementation of these models, when compared to neural networks. Moreover, parametric models make a lot of assumptions about the data and, in particular, the HSMM-LR model was proposed with a limitative assumption that the logistic regression weights only depend on state transitions.
Regarding the second contribution, we adapted the MMDL criterion, for order selection problems, to HSMMs. This adaptation was presented in a national pattern recognition conference - RECPAD2019.
Last but not least, the source code is publicly available at `https://github.com/filiparente/Predtweet/`.

5.2 Future Work

The main lines of future work to further improve our comparative study may explore:

- **Dataset limitations**

  Try out other real datasets and/or explore a smaller discretization unit ($\triangle t < 1$h), so that the observation signal is smoother and the features less noisy.

  Better feature engineering, i.e., collect more useful features by exploring entity recognition methods or combining textual with non-textual features.

[1] In our experiments, the Markov models converged at the minute order, compared to multiple hours in the case of the LSTM.

Regarding the TF-IDF features, it would also be interesting to try a dimensionality reduction through a principal component analysis [94], so that we are able to run the logistic regression model with a larger corpus, since the HSMM-LR is not scalable when the number of features is very high.

- **Prediction limitations**

  One of the first limitations is that we have considered a simple one-step-ahead prediction, which is not very realistic. A possible line of further work is to introduce:

  Multi-step-ahead prediction, since in real-life datasets we usually want to forecast multiple timesteps in the future; or

  Online learning, in which, instead of updating the parameters offline, they are constantly being changed whenever a new observation is available. This option is particularly interesting for social media applications, given that the textual features have temporal attention drifts over time. Online learning mechanisms allow us to capture those drifts without requiring training the model from scratch. Moreover, some predictions may not be related with past predictions since social media topics are constantly changing.

- **Model limitations**

  Another caveat in our implementation is the use of MMPP models, which can only achieve good performances when the dataset is discretized or if the observation signal is non smooth. In that regard, it would be interesting to consider models such as the GPCox and MMGCP (in Sections E.1.3 and 2.2.1.2), which combine Gaussian processes to obtain the desired smoothness. However, those models require a transformation that guarantees that the observations are positive (e.g. Log-transformation), which makes the problem intractable, forcing the use of approximate inference methods, such as MCMC (see Section E.2.1) or Variational Bayes (see Section E.2.2). A more promising idea would be to find a non-Gaussian conjugate prior with interesting and smooth properties that has, at the same time, a closed-form expression for the likelihood function.

- **Context based models**

  In this regard, it would be interesting to explore other ways to add features to parametric models. One suggestion is to consider the use of a model (e.g. MMPP, MMGCP or GPCox) together with a Cox regression approach. The model captures the temporal characteristics of the time-series while the Cox regression allows us to introduce context into the model, by conditioning it on external features. We could consider a more general state of the Markov chain which includes the information about the features. For example, for the MMGCP model, each state of the Markov chain comprises a latent function with some intensity level modeled by a Gaussian process prior, and we propose that it could also include the feature characteristics. The overall intensity function of the arrival process is modeled through conditional jumps in the states of the Markov chain, by defining the Markov process variable to be conditioned on the features, and applying Cox regression to decide to which state to move next, given the features of the arrival and the previous state.

Bibliography

[1] Y. Xie, X. Li, E. Ngai, and W. Ying. Customer churn prediction using improved balanced random forests. *Expert Systems with Applications*, 36(3):5445–5449, 2009.

[2] A. F. Dugas, M. Jalalpour, Y. Gel, S. Levin, F. Torcaso, T. Igusa, and R. E. Rothman. Influenza forecasting with google flu trends. *PloS one*, 8(2):e56176, 2013.

[3] P.-F. Pai and C.-S. Lin. A hybrid arima and support vector machines model in stock price forecasting. *Omega*, 33(6):497–505, 2005.

[4] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017.

[5] S. D. Campbell and F. X. Diebold. Weather forecasting for weather derivatives. *Journal of the American Statistical Association*, 100(469):6–16, 2005.

[6] X. Qing and Y. Niu. Hourly day-ahead solar irradiance prediction using weather forecasts by lstm. *Energy*, 148:461–468, 2018.

[7] I. Alon, M. Qi, and R. J. Sadowski. Forecasting aggregate retail sales:: a comparison of artificial neural networks and traditional methods. *Journal of retailing and consumer services*, 8(3):147–156, 2001.

[8] Z. Zhao, W. Chen, X. Wu, P. C. Chen, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

[9] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang. Short-term residential load forecasting based on lstm recurrent neural network. *IEEE Transactions on Smart Grid*, 10(1):841–851, 2017.

[10] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[11] D. R. Cox and V. Isham. *Point processes*, volume 12. CRC Press, 1980.

[12] E. N. Brown, R. E. Kass, and P. P. Mitra. Multiple neural spike train data analysis: state-of-the-art and future challenges. *Nature neuroscience*, 7(5):456–461, 2004.

[13] M. Haenggi, J. G. Andrews, F. Baccelli, O. Dousse, and M. Franceschetti. Stochastic geometry and random graphs for the analysis and design of wireless networks. *IEEE journal on selected areas in communications*, 27(7):1029–1046, 2009.

[14] M. Bertero, P. Boccacci, G. Desiderà, and G. Vicidomini. Image deblurring with poisson data: from cells to galaxies. *Inverse Problems*, 25(12):123006, 2009.

[15] R. Gallager. Poisson processes. In *6.262 Discrete Stochastic Processes*, chapter 2. Massachusetts Institute of Technology: MIT OpenCourseWare, Spring 2011. URL `https://ocw.mit.edu/`. License: Creative Commons BY-NC-SA.

[16] P. Aghion and P. Howitt. A model of growth through creative destruction. Technical report, National Bureau of Economic Research, 1990.

[17] J. Gardner and L. Knopoff. Is the sequence of earthquakes in southern california, with aftershocks removed, poissonian? *Bulletin of the Seismological Society of America*, 64(5):1363–1367, 1974.

[18] H. G. Othmer, S. R. Dunbar, and W. Alt. Models of dispersal in biological systems. *Journal of mathematical biology*, 26(3):263–298, 1988.

[19] D. C. Schmittlein, D. G. Morrison, and R. Colombo. Counting your customers: Who-are they and what will they do next? *Management science*, 33(1):1–24, 1987.

[20] M. Harchol-Balter. Lecture 9: The exponential distribution & poisson process. In *15-857 Performance Modeling & Design*. Carnegie Mellon University, October 2009. URL `ttps://pdfs.semanticscholar.org/ef08/ff8e550d118f2ffa7db21f7a31987658d5ec.pdf`.

[21] J. Virtamo. Poisson process. In *38.3143 Queuing Theory*. Helsinki University of Technology, 2005. URL `https://www.netlab.tkk.fi/opetus/s383143/kalvot/E_poisson.pdf`.

[22] P. R. Rios, D. Jardim, W. L. d. S. Assis, T. C. Salazar, and E. Villa. Inhomogeneous poisson point process nucleation: comparison of analytical solution with cellular automata simulation. *Materials Research*, 12(2):219–224, 2009.

[23] R. C. Larson and A. R. Odoni. *Urban operations research*. Number Monograph in 01. Prentice-Hall, NJ, 1981.

[24] D. Stoyan and A. Penttinen. Recent applications of point process methods in forestry statistics. *Statistical Science*, pages 61–78, 2000.

[25] A. C. Gatrell, T. C. Bailey, P. J. Diggle, and B. S. Rowlingson. Spatial point pattern analysis and its application in geographical epidemiology. *Transactions of the Institute of British geographers*, pages 256–274, 1996.

[26] H. Jing, Y. Yang, and R. M. Nishikawa. Detection of clustered microcalcifications using spatial point process modeling. *Physics in Medicine & Biology*, 56(1):1, 2010.

[27] S. Zhang, X. Liu, J. Tang, S. Cheng, and Y. Wang. Urban spatial structure and travel patterns: Analysis of workday and holiday travel using inhomogeneous poisson point process models. *Computers, Environment and Urban Systems*, 73:68–84, 2019.

[28] D. Lando. On cox processes and credit risky securities. *Review of Derivatives research*, 2(2-3): 99–120, 1998.

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

[30] P. H. Franses, P. H. B. Franses, et al. *Time series models for business and economic forecasting*. Cambridge university press, 1998.

[31] S. J. Taylor. *Modelling financial time series*. world scientific, 2008.

[32] J. D. Salas. *Applied modeling of hydrologic time series*. Water Resources Publication, 1980.

[33] J. S. Richman and J. R. Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*, 278(6): H2039–H2049, 2000.

[34] Y. Hur and S. Lim. Customer churning prediction using support vector machines in online auto insurance service. In *International Symposium on Neural Networks*, pages 928–933. Springer, 2005.

[35] T. Gunter, C. Lloyd, M. A. Osborne, and S. J. Roberts. Efficient bayesian nonparametric modelling of structured point processes. *arXiv preprint arXiv:1407.6949*, 2014.

[36] D. Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

[37] T. Rolski, H. Schmidli, V. Schmidt, and J. L. Teugels. *Stochastic processes for insurance and finance*, volume 505. John Wiley & Sons, 2009.

[38] V. S. Frost and B. Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32(3):70–81, 1994.

[39] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of artificial intelligence research*, 11:391–427, 1999.

[40] M. Dong and D. He. Hidden semi-markov model-based methodology for multi-sensor equipment health diagnosis and prognosis. *European Journal of Operational Research*, 178(3):858–878, 2007.

[41] A. Tolver. An introduction to markov chains. *Recuperado el*, 15, 2016.

[42] W. Fischer and K. Meier-Hellstern. The markov-modulated poisson process (mmpp) cookbook. *Performance evaluation*, 18(2):149–171, 1993.

[43] M. Kim. Markov modulated gaussian cox processes for semi-stationary intensity modeling of events data. In *International Conference on Machine Learning*, pages 2645–2653, 2018.

[44] C. Lloyd, T. Gunter, M. Osborne, and S. Roberts. Variational inference for gaussian process modulated poisson processes. In *International Conference on Machine Learning*, pages 1814–1822, 2015.

[45] M. J. Gales. Maximum likelihood linear transformations for hmm-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

[46] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *2012 IEEE international conference on Acoustics, speech and signal processing (ICASSP)*, pages 4277–4280. IEEE, 2012.

[47] R. Meenakshi and P. Anandhakumar. A hybrid brain tumor classification and detection mechanism using knn and hmm. *Current Medical Imaging Reviews*, 11(2):70–76, 2015.

[48] M. Zhang, X. Jiang, Z. Fang, Y. Zeng, and K. Xu. High-order hidden markov model for trend prediction in financial time series. *Physica A: Statistical Mechanics and its Applications*, 517: 1–12, 2019.

[49] J. Liu, L. Zhu, Y. Wang, X. Liang, J. Hyyppä, T. Chu, K. Liu, and R. Chen. Reciprocal estimation of pedestrian location and motion state toward a smartphone geo-context computing solution. *Micromachines*, 6(6):699–717, 2015.

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

[51] H. Zen, K. Tokuda, and A. W. Black. Statistical parametric speech synthesis. *speech communication*, 51(11):1039–1064, 2009.

[52] T. Nakamura, T. Nagai, D. Mochihashi, I. Kobayashi, H. Asoh, and M. Kaneko. Segmenting continuous motions with hidden semi-markov models and gaussian processes. *Frontiers in neurorobotics*, 11:67, 2017.

[53] S.-Z. Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2):215–243, 2010.

[54] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[55] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[56] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[57] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[58] J. Read. Hidden markov models and dynamic programming. *University of Oslo*, 2011.

[59] B.-J. Yoon. Hidden markov models and their applications in biological sequence analysis. *Current genomics*, 10(6):402–415, 2009.

[60] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell System Technical Journal*, 62(4):1035–1074, 1983.

[61] T. P. Mann. Numerically stable hidden markov model implementation. *An HMM scaling tutorial*, pages 1–8, 2006.

[62] K. P. Murphy. Hidden semi-markov models (hsmms). *unpublished notes*, 2, 2002.

[63] B. G. Leroux. Maximum-likelihood estimation for hidden markov models. *Stochastic processes and their applications*, 40(1):127–143, 1992.

[64] S.-Z. Yu and H. Kobayashi. Practical implementation of an efficient forward-backward algorithm for an explicit-duration hidden markov model. *IEEE Transactions on Signal Processing*, 54(5): 1947–1951, 2006.

[65] M. A. Figueiredo, J. M. Leitão, and A. K. Jain. On fitting mixture models. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 54–69. Springer, 1999.

[66] M. Bicego, V. Murino, and M. A. Figueiredo. A sequential pruning strategy for the selection of the number of states in hidden markov models. *Pattern Recognition Letters*, 24(9-10):1395–1407, 2003.

[67] R. B. Cooper. Queueing theory. In *Proceedings of the ACM'81 conference*, pages 119–122, 1981.

[68] T. Shinozaki. Hmm state clustering based on efficient cross-validation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I, 2006.

[69] A. Biem, J.-Y. Ha, and J. Subrahmonia. A bayesian model selection criterion for hmm topology optimization. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages I–989. IEEE, 2002.

[70] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

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

[72] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[73] V. Barbu, J. Bulla, and A. Maruotti. Estimation of the stationary distribution of a semi-markov chain. *Journal of Reliability and Statistical Studies*, 5:15–26, 2012.

[74] P. Deuflhard, W. Huisinga, A. Fischer, and C. Schütte. Identification of almost invariant aggregates in reversible nearly uncoupled markov chains. *Linear Algebra and its Applications*, 315(1-3):39–59, 2000.

[75] Z. Chen. Estimating latent attentional states based on simultaneous binary and continuous behavioral measures. *Computational intelligence and neuroscience*, 2015:26, 2015.

[76] S. Y. Xu and C. Berkely. Stock price forecasting using information from yahoo finance and google trend. *UC Brekley*, 2014.

[77] R. Rouhi, M. Jafari, S. Kasaei, and P. Keshavarzian. Benign and malignant breast tumors classification based on region growing and cnn segmentation. *Expert Systems with Applications*, 42(3): 990–1002, 2015.

[78] M. Cunha, A. R. Marcal, and L. Silva. Very early prediction of wine yield based on satellite data from vegetation. *International Journal of Remote Sensing*, 31(12):3125–3142, 2010.

[79] D. Böhning. Multinomial logistic regression algorithm. *Annals of the institute of Statistical Mathematics*, 44(1):197–200, 1992.

[80] P. Gong and J. Ye. A modified orthant-wise limited memory quasi-newton method with convergence analysis. In *International Conference on Machine Learning*, pages 276–284, 2015.

[81] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23 (4):550–560, 1997.

[82] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *The Journal of Machine Learning Research*, 11:3183–3234, 2010.

[83] A. Albert and J. A. Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1–10, 1984.

[84] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[85] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[86] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[87] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[88] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[89] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[90] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[91] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[92] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[93] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[94] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[95] R. Abbi, E. El-Darzi, C. Vasilakis, and P. Millard. Analysis of stopping criteria for the em algorithm in the context of patient grouping according to length of stay. In *2008 4th International IEEE Conference Intelligent Systems*, volume 1, pages 3–9. IEEE, 2008.

[96] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[97] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[98] S. Siegel. *Nonparametric statistics for the behavioral sciences.* McGraw-hill, 1956.

[99] T. E. Nichols and A. P. Holmes. Nonparametric permutation tests for functional neuroimaging: a primer with examples. *Human brain mapping*, 15(1):1–25, 2002.

[100] E. Maris and R. Oostenveld. Nonparametric statistical testing of eeg-and meg-data. *Journal of neuroscience methods*, 164(1):177–190, 2007.

[101] S. A. Terwijn. On the learnability of hidden markov models. In *International Colloquium on Grammatical Inference*, pages 261–268. Springer, 2002.

[102] J. P. Cunningham, K. V. Shenoy, and M. Sahani. Fast gaussian process methods for point process intensity estimation. In *Proceedings of the 25th international conference on Machine learning*, pages 192–199. ACM, 2008.

[103] R. P. Adams, I. Murray, and D. J. MacKay. Tractable nonparametric bayesian inference in poisson processes with gaussian process intensities. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 9–16. ACM, 2009.

[104] M. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

[105] A. Dezfouli and E. V. Bonilla. Scalable inference for gaussian process models with black-box likelihoods. In *Advances in Neural Information Processing Systems*, pages 1414–1422, 2015.

[106] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[107] D. Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, University of Cambridge, 2014.

[108] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[109] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[110] W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

[111] J. J. O. Ruanaidh and W. J. Fitzgerald. *Numerical Bayesian methods applied to signal processing*. Springer Science & Business Media, 2012.

[112] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[113] J. Eisner. High-level explanation of variational inference. *Department of Computer Science, Johns Hopkins University*, 2011.

[114] W. Penny, S. Kiebel, and K. Friston. Variational bayesian inference for fmri time series. *NeuroImage*, 19(3):727–741, 2003.

[115] J. E. Bronson, J. Fei, J. M. Hofman, R. L. Gonzalez Jr, and C. H. Wiggins. Learning rates and states from biophysical time series: a bayesian approach to model selection and single-molecule fret data. *Biophysical journal*, 97(12):3196–3205, 2009.

[116] N. de Freitas, P. Højen-Sørensen, M. I. Jordan, and S. Russell. Variational mcmc. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 120–127, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1. URL `http://dl.acm.org/citation.cfm?id=2074022.2074038`.

Appendix A

Supervised learning in HMMs

Proof. The joint likelihood of the state and observation sequences is given by eq. (2.10), which can be further simplified by combining repeated terms. We define the following auxiliary variables: n_{ij} is the number of transitions from state s_i to state s_j; n_{jv} is the number of times the observations take value v in state s_j; and n_{0i} is the number of times s_i is the first state in the state sequence(s).
Re-writing eq. (2.10) with these variables, yields[1]

$$\mathbb{P}(S, O; \mu) = \prod_{i=1}^{k} \pi_i^{n_{0i}} \prod_{j=1}^{k} A_{ij}^{n_{ij}} \prod_{v=0}^{+\infty} B_{jv}^{n_{jv}}. \tag{A.1}$$

In practice, the log-likelihood of the observations is preferred for numerical simplicity. Ergo, the data log-likelihood (LL) is obtained by taking the logarithm of eq. (A.1) as follows

$$\mathsf{LL} = \log\Big(\mathbb{P}(S, O; \mu)\Big) = \sum_{i=1}^{k} n_{0i} \log(\pi_i) + \sum_{j=1}^{k} n_{ij} \log(A_{ij}) + \sum_{v=\min(O)}^{\max(O)} n_{jv} \log(B_{jv}). \tag{A.2}$$

Equation (A.2) is not complete due to the fact that it allows the parameters to take any value without constraints. The constraints should be added aside. Since we are working with probabilities, we need to guarantee that they sum to 1, i.e., $\sum_{i=1}^{k} \pi_i = 1$, $\sum_{j=1}^{k} A_{ij} = 1$ and $\sum_{v=\min(O)}^{\max(O)} B_{jv} = 1$. To include these constraints, three lagrangian multipliers (ξ_0, ξ_i and ξ_j) are added to eq. (A.2), one for each respective constraint. The new log-likelihood expression, LL′, including the lagrangian multipliers, is

$$\mathsf{LL}' = \mathsf{LL} - \sum_{i=1}^{k} \xi_i \left(\sum_{j=1}^{k} A_{ij} - 1 \right) - \sum_{j=1}^{k} \xi_j \left(\sum_{v=\min(O)}^{\max(O)} B_{jv} - 1 \right) - \xi_0 \left(\sum_{i=1}^{k} \pi_i - 1 \right). \tag{A.3}$$

The maximum likelihood estimate for the observation distribution is obtained by deriving eq. (A.3) w.r.t B_{jv} and equating it to zero. That is,

$$\left. \frac{\partial \mathsf{LL}'}{\partial B_{jv}} \right|_{B_{jv}=\hat{B}_{jv}} = \frac{n_{jv}}{\hat{B}_{jv}} - \xi_j = 0 \iff \hat{B}_{jv} = \frac{n_{jv}}{\xi_j}, \qquad \forall j \in [k], \quad \forall v \in \mathcal{V}. \tag{A.4}$$

[1] In fact, in eq. (A.1) the set of observation values can be reduced from $\{0, ..., +\infty\}$ to $\{\min(O), ..., \max(O)\}$ since $n_{jv} = 0, \quad \forall v \in \{\{0, ..., \min(O)\} \cup \{\max(O), ..., +\infty\}\}$.

To solve eq. (A.4), we first need to find the expression for the lagrangian multiplier ξ_j

$$\frac{\partial \mathrm{LL}'}{\partial \xi_j} = -\left(\sum_{v=\min(O)}^{\max(O)} B_{jv} - 1 \right) \Bigg|_{B_{jv}=\hat{B}_{jv}} = 0 \iff \sum_{v=\min(O)}^{\max(O)} \hat{B}_{jv} = 1. \tag{A.5}$$

Replacing eq. (A.4) in eq. (A.5), we obtain:

$$\sum_{v=\min(O)}^{v=\max(O)} \frac{n_{jv}}{\xi_j} = 1 \iff \xi_j = \sum_{v=\min(O)}^{v=\max(O)} n_{jv} \quad \forall j \in [k]. \tag{A.6}$$

Hence, the maximum likelihood estimate for B_{jv} is obtained by replacing eq. (A.6) in eq. (A.4). That is,

$$\hat{B}_{jv} = \frac{n_{jv}}{\sum_{v=\min(O)}^{v=\max(O)} n_{jv}},$$

i.e., the ratio between the number of times the observations take value v in state s_j and the number of times state s_j appears in the state sequence, with the corresponding observation taking any value from the set of m possible values ($\mathcal{V}$). Likewise, the maximum likelihood estimate for the element A_{ij} of the transition matrix $\boldsymbol{A}$ is obtained, by following the same steps, as

$$\hat{A}_{ij} = \frac{n_{ij}}{\sum_{j=1}^{k} n_{ij}},$$

i.e., the ratio between the number of transitions from state s_i to state s_j in the given state sequence and the number of transitions from state s_i to any other state. Finally, the maximum likelihood estimate for the element π_i of the initial state distribution $\boldsymbol{\pi}$ is

$$\hat{\pi}_i = \frac{n_{0i}}{\sum_{i=1}^{k} n_{0i}},$$

i.e., the ratio between the number of times s_i is the first state for all sequences and the total number of sequences (if multiple sequences are available).

If the emission probability is not a multinomial distribution but a Poisson distribution with parameter $\boldsymbol{\lambda}$, then, B_{jv} in eq. (A.2) is replaced by the Poisson PMF from eq. (2.9), such that[2]

$$\log(B_{jv}) = -\lambda_j + v \times \log(\lambda_j) - \log(v!).$$

The maximum likelihood estimate for λ_j is therefore

$$\left.\frac{\partial LL}{\partial \lambda_j}\right|_{\lambda_j=\hat{\lambda}_j} = \sum_{v=\min(O)}^{\max(O)} n_{jv} \left(-1 + \frac{v}{\hat{\lambda}_j} \right) = 0 \quad \iff \hat{\lambda}_j = \frac{\sum_{v=\min(O)}^{\max(O)} v \times n_{jv}}{\sum_{v=\min(O)}^{\max(O)} n_{jv}} = \frac{\sum_{t=1}^{n} o_t \times \mathbf{1}(S_t = s_j)}{\sum_{t=1}^{n} 1 \times \mathbf{1}(S_t = s_j)},$$

which corresponds to the average of the observations in the corresponding state s_j. □

[2]Since the goal is to find the maximum likelihood estimator of the parameters, the term $log(v!)$ can be removed because it does not depend on any of the parameters from μ. Moreover, since the only constraint is that $\boldsymbol{\lambda}$ must be positive and this is inherently guaranteed from the equations, no Lagrangian multipliers are needed for $\boldsymbol{\lambda}$.

Appendix B

Expectation-maximization algorithm

The *expectation-maximization* (EM) algorithm is used for problems with double dependency. For example, in a HMM, the posterior over the hidden state sequence depends on the HMM parameters μ and the parameters depend on the hidden state. For this reason, the optimization problem that aims to find the optimal parameters can not be solved in closed form, instead, an iterative method, such as the EM, must be used. The EM algorithm comprises two steps: the *E(xpectation)-step* and the *M(aximization)-step*, summarized as follows:

1. **Initialization:** Start with an initial set of estimated parameters $\hat{\mu}$;

2. **Iterate**, hoping to converge on optimal values for the model parameters $\hat{\mu}$:

   E-step: Using the (new) parameters $\hat{\mu}$, feed the observations through the current model and estimate expectations/weights for each model parameter.
   For example, in HMMs, we compute the expected transition and emission counts in the data.[1] Then, we infer the posterior distribution over the hidden states [50] given the observations and the model parameters.

   M-step: Re-estimate the parameters $\hat{\mu}$ with the expectations/weights from the E-step.
   The hidden states are treated as "observed" according to the state posterior distribution found in the E-step and the parameters are recomputed taking this information into account. The model parameters are estimated, using the "observed" states, by pseudo counting and normalizing.

   Termination: If the algorithm did not converge yet, go back to step 2.

The initialization and stop criteria are crucial for the performance of the algorithm.

Stop criteria Regarding the stop criteria, the algorithm must stop when one of the following conditions is met:

- **Log-likelihood relative change**: The idea is to detect how much the log-likelihood has increased from one iteration to the next and stop the algorithm when only small improvements are being

[1] In HMMs, it is required to apply the *forward-backward algorithm* (found in Section 2.2.2.4) to ease the computational burden.

made, based on a threshold value. Hence, the stop criterion is the relative change in the log-likelihood values of two consecutive iterations, i.e., $\left|\frac{\mathsf{LL}_{t+1} - \mathsf{LL}_t}{\mathsf{LL}_{t+1}}\right| < \epsilon \approx 10^{-10}$. However, this criterion does not guarantee that the algorithm has reached a maximum of the log-likelihood [95]. Even though, this criteria is preferred over the next ones.

- **Number of iterations**: Specify *a priori* the maximum number of iterations before applying the algorithm. This criterion is problem dependent [95];
- **Parameters' relative change**: Similarly to the log-likelihood relative change, the parameters' estimated value is compared between consecutive iterations and the algorithm is stopped when the relative change is smaller than ϵ. As there is more than one parameter, the criterion becomes: $\max\left(||\hat{\mu}_{i_{t+1}} - \hat{\mu}_{i_t}, \quad \forall \hat{\mu}_i \in \hat{\mu}||\right) < \epsilon$, where $||\cdot||$ is a norm (e.g. ℓ_1 or ℓ_2-norms).

Initialization Regarding the initialization, usually the parameters are initialized at random. For example, in Section 2.2.2.3, $\hat{\alpha} \in [0, 1]$ is sampled from a uniform distribution over the interval $[0, 1]$, whereas $\hat{\lambda}_1$ and $\hat{\lambda}_2$ can be initialized as the mean of the combined process with a small distortion $\epsilon > 0$ in opposite directions, i.e., $\hat{\lambda}_1 = \bar{o} + \epsilon$ and $\hat{\lambda}_2 = \bar{o} - \epsilon$.

To sum up, the EM algorithm is an iterative procedure to approximate a posterior distribution over missing data. It is analogous to Variational inference (see Section E.2.2), in which the variational parameters are adjusted so that the proposal distribution increases the ELBO. Here, the ELBO is the data (log-)likelihood and in the E-step the observations are fractionally associated to each state in order to maximize the data (log-)likelihood. In the M-step, the parameters are adjusted accordingly.

In the following paragraph we will adress some implementation issues and solutions.

Implementation details One implementation issue that can arise when using the EM algorithm concerns *state assignments*. For example, consider a model with Poisson emissions that, after running the EM algorithm, gives the following estimates for the rates of the Poisson processes:
$\hat{\boldsymbol{\lambda}} = [1.4725, 14.9980, 2.6674]$, while the true parameters are $\boldsymbol{\lambda} = [1, 3, 15]$. In fact, the true state s_3 is assigned as state s_2 in the EM algorithm, and vice-versa. This occurs because the algorithm has no information about which parameter corresponds to each state. To correctly compare the results, we need to permute the states. If the number of states k is small, we can do the permutations by hand. If not, the complexity of computing all permutations is $\mathcal{O}(k!)$, which can easily become intractable. An efficient method, first proposed by Harold Kuhn [96] in 1955 and later reviewed by James Munkres [97] in 1957, is called the *Munkres assignment algorithm*. It is a combinatorial optimization algorithm to efficiently solve the assignment problem, by building a squared nonnegative cost matrix $\boldsymbol{C} \in \mathbb{R}^{k \times k}$, which contains a certain cost for each pair of *(algorithm state, true state)*. For example, in the supervised setting, where we have acess to the state sequence, we can compute the cost as the number of correspondences between each algorithm state and each true state in the predicted and true state sequences, respectively.

In this case, we want the optimal state assignment that *maximizes* the overall cost of this matrix. Since the algorithm works as a minimization problem, we minimize the negative of the cost matrix.[2] Munkres solves this problem using matrix algebra operations such as row and column reduction, which reduce the complexity from $\mathcal{O}(k!)$ to $\mathcal{O}(k^3)$.[3] As an example of another cost value, if we have acess to the true parameters, we can use the ℓ_2-norm between the predicted and true parameters and minimize that cost.

Another implementation detail regards *probability smoothing*. In the EM algorithm, it is recommended to smooth the probabilities at each iteration, i.e., add a small positive term (e.g. $\epsilon \approx 1e-100$), which avoids numerical errors that are likely to occur. Also, it is advantageous when the total number of states (k) is large, for example, when performing state pruning with a large initial number of states k_{max} (see Algorithm 2). In that case, some states may not be used at all, and the transition matrix will have rows that sum to zero instead of one. By smoothing the transition matrix at each iteration and then normalizing it, those cases end up in uniform distributions.

A third and last implementation detail consists of how to adapt the EM algorithm for multiple observation sequences. Instead of having one observation sequence with n samples, we have N observation sequences, where each sequence has a number of samples equal to $n_\nu, \quad \forall \nu \in [N]$. The required adaptations consist of replacing the summation $\sum_{t=1}^{n}$ by $\sum_{\nu=1}^{N}\sum_{t=1}^{n_\nu}$ in the computation of the expectations in the E-step and in the re-estimation of the parameters in the M-step (eqs. (2.39), (2.40) and (2.50) for HMMs, MMPPs and HSMMs, respectively).

[2] Note that we cannot assign two algorithm states to the same true state.

[3] In the case where the total number of algorithm states does not match the total number of true states, we add zeros to the cost matrix to make it square, and discard those state assignments from the solution.

Appendix C

Equivalence between HSMMs

This appendix contains the proof of the equivalence between a HSMM μ with k states, where the duration probability of each state is a mixture of geometric distributions and another HSMM μ' with more states (k') where the duration probability of each state is a single geometric distribution.[1]

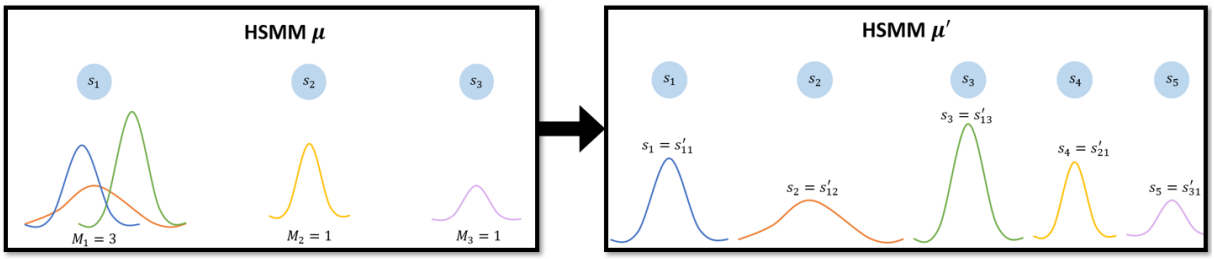


Figure C.1: Simple drawing that illustrates the equivalence between HSMM models μ and μ'. The original model has 3 states: state 1 contains a mixture of 3 state duration Gaussian distributions; while state 2 and 3 contain a single Gaussian. The equivalent model has 5 states, one for each of the Gaussian components of the mixtures in the original model.

Figure C.1 illustrates the HSMM model μ, with Poisson emission probability given by eq.(2.9). The duration probability in state s_i is a mixture of M_i geometric distributions

$$D_{id} = p_i(d) = \sum_{m=1}^{M_i} c_{im} \text{ Geom}(d|\theta_{im}), \tag{C.1}$$

with the m^{th} component in state s_i having mixing weight c_{im} and geometric probability mass function given by eq.(2.11), with parameter θ_{im} instead of A_{ii}.

The HSMM model μ, shown in Figure C.1, with emission and duration probabilities given by eqs. (2.9) and (C.1) is equivalent to another HSMM μ' with $k' = \sum_{i=1}^{k} M_i$ states where each state has only one geometric distribution to specify its duration. The equivalence is defined in terms of the likelihood of observations $O = O_{1:n}$, i.e., $\mathbb{P}(O;\mu) = \mathbb{P}(O;\mu')$. In section C.1, we describe how model μ' is built and, in section C.2, we present the proof of the equivalence between both models.

[1] The geometric distribution is used as the discrete counterpart of the exponential distribution, given that the durations are in fact the number of observations produced while in a state and, consequently, are discrete random variables.

C.1 Building HSMM μ'

Given the original HSMM $\boldsymbol{\mu} = \{\mathbf{s}, \boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{B}, \boldsymbol{D}\}$, its equivalent model $\boldsymbol{\mu}' = \{\mathbf{s}', \boldsymbol{A}', \boldsymbol{\pi}', \boldsymbol{B}', \boldsymbol{D}'\}$ is built by splitting each state s_i into M_i states, one for each geometric distribution of the mixture. Thus we obtain $k' = \sum_{i=1}^{k} M_i$ states for model μ'. The set of states of the new model is $\mathbf{s}' = \{s'_1, ..., s'_{k'}\} = \{s'_{11}, ..., s'_{1M_1}, s'_{21}, ..., s'_{2M_2}, s'_{31}, ..., s'_{kM_k}\}$, where the double index notation in s'_{kM_k} represents the $M_k{}^{\text{th}}$ geometric distribution of the original state s_k.
The emission probability of each state s'_{im}

$$b'(o_t|s'_{im}) = b'_{im}(o_t) = b(o_t|s_i) = b_i(o_t), \tag{C.2}$$

is the same as in eq. (2.9) due to the fact that the emission probability density function is conditionally independent on the component of the mixture, given the corresponding state. Regarding the state transition probability, using the double index notation, we have

$$A'_{ikz,jmd} = A'_{ik,jmd} = A_{ij} c_{jm} p'_{jm}(d), \tag{C.3}$$

representing the probability of going from state s'_{ik} with duration z to state s'_{jm} with duration d.
In eq. (C.3): the explicit duration assumption is exploited meaning that a state transition is independent of the previous duration; A_{ij} is the original state transition probability from state s_i to state s_j (as in HMMs); c_{jm} is the mixing weight of the m^{th} component from the original state s_j; and $p'_{jm}(d)$ is the probability of duration d in state s'_{jm}, given by the geometric probability mass function in eq. (2.11) with parameter θ_{jm}. Notice that

$$\sum_{jmd} A'_{ikz,jmd} = \sum_{j=1}^{k} \sum_{m=1}^{M_j} \sum_{d=1}^{d_{max}} A_{ij} c_{jm} p'_{jm}(d), \qquad \forall i \in [k]. \tag{C.4}$$

Replacing eq. (2.11) in (C.4), yields the following result

$$\sum_{jmd} A'_{ikz,jmd} = \sum_{j=1}^{k} A_{ij} \sum_{m=1}^{M_j} c_{jm} \sum_{d=1}^{d_{max}} \text{Geom}(d|\theta_{jm}) = 1,$$

which proves that $A'_{ikz,jmd}$ normalizes to 1, as required.
Finally, regarding the initial state probability, we define

$$\pi'_{s'_{jm}} = \pi_{s_j} c_{jm}, \tag{C.5}$$

as the probability that state s'_{jm} is the first state in the state sequence, which also normalizes to 1.

C.2 Equivalence between HSMMs μ and μ'

The proof of the equivalence between HSMM models μ and μ' uses the *forward-backward* procedure (see Section 2.2.2.4) to compute the likelihood of the observations, namely the forward variable of the explicit duration HSMM, defined in eq.2.41 and iteratively computed according to eq.2.42.
The likelihood of the observations for this model is computed by marginalization[2]

$$\mathbb{P}(O;\mu) = \sum_{j=1}^{k} \mathbb{P}[O, S_{\rightarrow n|} = s_j;\mu] = \sum_{j=1}^{k} \alpha_n(s_j). \tag{C.6}$$

Notice that the likelihood has the same expression as for ordinary HMMs [50].
To simplify notation, we will re-write eq. (2.42) (evaluated at $t = n$) as

$$\alpha_n(s_j) = \sum_{i=1}^{k} \sum_{d=1}^{\min(d_{max},n)} \alpha_{n-d}(s_i) A_{ij} p_j(d) u_n(s_j, d), \tag{C.7}$$

where $u_t(s_j, d) = \prod_{\tau=t-d+1}^{t} b_j(\tau)$ and $p_j(d) = D_{jd}$.
Similarly, the likelihood of the observations for model μ' is

$$\mathbb{P}(O;\mu') = \sum_{j=1}^{k'} \alpha_n(s'_j) = \sum_{j=1}^{k} \sum_{m=1}^{M_j} \alpha_n(s'_{jm}), \tag{C.8}$$

that is, using the double index component notation introduced in Section C.1.
In order to prove the equivalence between models μ and μ', in terms of the likelihoods of eqs. (C.6) and (C.8), we introduce the following variable

$$\alpha'_n(s_j) = \sum_{m=1}^{M_j} \alpha_n(s'_{jm}). \tag{C.9}$$

The two models are equivalent iif[3] $\alpha'_n(s_j) = \alpha_n(s_j), \forall j \in [k]$.
The proof of this relation is derived by induction on the length (n) of the observation sequence.

Proof. For $n = 1$, we have

$$\alpha_1(s_j) = \sum_{i=1}^{k} \alpha_0(s_i) A_{ij} p_j(1) u_1(s_j, 1) = \sum_{i=1}^{k} \pi_{s_i} A_{ij} p_j(1) b_j(o_1), \tag{C.10}$$

for model μ and

$$\alpha'_1(s_j) \underset{def}{=} \sum_{m=1}^{M_j} \alpha_1(s'_{jm}) = \sum_{m=1}^{M_j} \sum_{i=1}^{k} \sum_{l=1}^{M_i} \pi'_{s'_{il}} A'_{il,jm1} b'_{jm}(o_1), \tag{C.11}$$

for model μ'. Replacing in eq. (C.11) the definitions from the previous section (eqs. (C.2), (C.3) and

[2] The notation $\alpha_t(s_j)$ is equivalent to $\alpha_t(j)$. Likewise, $b_j(t) \equiv b_j(o_t)$ and $\pi_i \equiv \pi_{s_i}$.
[3] If and only if.

(C.5)), yields

$$\alpha'_1(s_j) = \sum_{m=1}^{M_j} \sum_{i=1}^{k} \sum_{l=1}^{M_i} \pi_{s_i} c_{il} A_{ij} c_{jm} \text{Geom}(1|\theta_{jm}) b_j(o_1). \tag{C.12}$$

Re-arranging terms and noticing that $\sum_{l=1}^{M_i} c_{il} = 1$ and that $\sum_{m=1}^{M_j} c_{jm} \text{Geom}(1|\theta_{jm}) = p_j(1)$ (eq. (C.1)), we finally get

$$\alpha'_1(s_j) = b_j(o_1) \left(\sum_{m=1}^{M_j} c_{jm} \text{Geom}(1|\theta_{jm}) \right) \sum_{i=1}^{k} \pi_{s_i} A_{ij} \left(\sum_{l=1}^{M_i} c_{il} \right) = \sum_{i=1}^{k} \pi_{s_i} A_{ij} p_j(1) b_j(o_1). \tag{C.13}$$

Therefore, we can conclude that $\alpha'_1(s_j) \equiv \alpha_1(s_j), \forall j \in [k]$.
To show the recursion, we need to prove the following implication

$$\alpha_{n+1-d}(s_j) = \alpha'_{n+1-d}(s_j), \quad \forall d \implies \alpha_{n+1}(s_j) = \alpha'_{n+1}(s_j). \tag{C.14}$$

From eq. (C.7), we know that

$$\alpha_{n+1}(s_j) = \sum_{i=1}^{k} \sum_{d=1}^{\min(d_{max}, n+1)} \alpha_{n+1-d}(s_i) A_{ij} p_j(d) u_{n+1}(s_j, d). \tag{C.15}$$

Likewise, for model μ', we have

$$\alpha'_{n+1}(s_j) \underset{def}{=} \sum_{m=1}^{M_j} \alpha_{n+1}(s'_{jm}) = \sum_{m=1}^{M_j} \sum_{i=1}^{k} \sum_{l=1}^{M_i} \sum_{d=1}^{\min(d_{max}, n+1)} \alpha_{n+1-d}(s'_{il}) A'_{il,jmd} u_{n+1}(s'_{jm}, d). \tag{C.16}$$

Invoking eq. (C.3) and noticing that, from eq. (C.2), we can conclude that $u_{n+1}(s'_{jm}, d) = u_{n+1}(s_j, d)$, equation (C.16) simplifies to

$$\alpha'_{n+1}(s_j) = \sum_{i=1}^{k} \sum_{d=1}^{\min(d_{max}, n+1)} u_{n+1}(s_j, d) A_{ij} \left(\sum_{m=1}^{M_j} c_{jm} \text{Geom}(d|\theta_{jm}) \right) \left(\sum_{l=1}^{M_i} \alpha_{n+1-d}(s'_{il}) \right). \tag{C.17}$$

Again, equation (C.17) can be further simplified by taking advantage of eqs. (C.1) and (C.9), yielding

$$\alpha'_{n+1}(s_j) = \sum_{i=1}^{k} \sum_{d=1}^{\min(d_{max}, n+1)} \alpha'_{n+1-d}(s_i) A_{ij} p_j(d) u_{n+1}(s_j, d). \tag{C.18}$$

Comparing (C.15) and (C.18), it is clear that the implication in eq. (C.14) is true. For example, evaluating (C.14) for $n = 1$ means that (for $d = 1$, the only allowed duration) if $\alpha_1(s_j) = \alpha'_1(s_j)$ then $\alpha_2(s_j) = \alpha'_2(s_j)$. The former equality was deducted in the first part of our proof, which implies the latter. Subsequently, evaluating (C.14) for $n = 2$ means that (for $d = [1, 2]$) if $\alpha_1(s_j) = \alpha'_1(s_j)$ and $\alpha_2(s_j) = \alpha'_2(s_j)$ then $\alpha_3(s_j) = \alpha'_3(s_j)$. Both premises are guaranteed from the first proof and from the conclusions of the previous implication. Thus, the conclusion is met and the recursion continues until the condition is met for all n. This concludes our proof that $\alpha'_n(s_j) = \alpha_n(s_j)$ and, consequently, the models are likelihood-equivalent. □

Appendix D

Algorithms' pseudo-code

Algorithm 2 Sequential pruning strategy

Set k_{min} and k_{max}
$k \leftarrow k_{max}$
Initialize the parameters $\hat{\mu}_k$ randomly
while $k \geq k_{min}$ **do**
    $\hat{\mu}_k \leftarrow \text{EM}(o, \hat{\mu}_k)$
    $C_k \leftarrow \text{Criterion}(\hat{\mu}_k)$
    LPS $\leftarrow \arg\min(\boldsymbol{p_\infty})$
    Prune the LPS from the estimated parameters: $\hat{\mu}_k[LPS] \leftarrow []$ and normalize
    $k \leftarrow k - 1$
end while
$k^* \leftarrow \arg\max(C_k)$
$\mu^*_{k^*} \leftarrow \mu_{k^*}$

Algorithm 3 Sequential standard strategy

Set k_{min} and k_{max}
$k \leftarrow k_{max}$
while $k \geq k_{min}$ **do**
    Initialize the parameters $\hat{\mu}_k$ at random
    $\hat{\mu}_k \leftarrow \text{EM}(o, \hat{\mu}_k)$
    $C_k \leftarrow \text{Criterion}(\hat{\mu}_k)$
    $k \leftarrow k - 1$
end while
$k^* \leftarrow \arg\max(C_k)$
$\mu^*_{k^*} \leftarrow \mu_{k^*}$

Algorithm 4 Forecasting algorithm: main()

```
Define setup
for △t ∈ {1, 3, 4, 6, 12, 24, 48} do
  for △w ∈ {0, 1, 3, 5, 7} do
    if △w = 0 then
      k*_HMM ← Pruning HMM(k_min_HMM, k_max_HMM)
      k*_HSMM, d*_HSMM ← Pruning HSMM(k_min_HSMM, k_max_HSMM)
    else
      k_max_HMM ← k*_HMM + 5
      k_max_HSMM ← k*_HSMM + 5
    end if
    run(args, k*_HMM, k*_HSMM, d*_HSMM)
  end for
end for
```

Algorithm 5 Forecasting algorithm: run(args, k^*_{HMM}, k^*_{HSMM}, d^*_{HSMM})

```
Get observations and features
Pre-processing steps
n ← 1
while n ≤ 10 do
  for model ∈ {HMM, HSMM, HSMM-LR} do
    μ_{k*_model} ← Initialize EM parameters randomly
    best_MSE ← ∞
    μ̂*_{k*_model} ← {}
    if model = HSMM-LR then
      if n = 1 then
        φ̂ ← Binary-search
      end if
      Train: μ̂_{k*_model} ← EM(μ_{k*_model})+L-BFGS(φ̂)
    else
      Train: μ̂_{k*_model} ← EM(μ_{k*_model})
    end if
    MSE_Val ← Compute the MSE of the predictions in the validation set, with the trained model
    if MSE_Val < best_MSE then
      Store best parameters: μ̂*_{k*_model} ← μ̂_{k*_model}
      best_MSE ← MSE_Val
    end if
  end for
  n ← n + 1
end while
evaluate(args, μ̂*_{k*_model})
```

Algorithm 6 Forecasting algorithm: evaluate(args, $\hat{\mu}^*_{k^*_{\text{model}}}$)

for model $\in \{\text{HMM}, \text{HSMM}, \text{HSMM-LR}\}$ **do**
  $n \leftarrow 1$
  $\text{best}_{\hat{S}_{\text{Test}}} \leftarrow []$
  $\text{best}_{\text{MSE}_{\text{Test}}} \leftarrow \infty$
  while $n \leq 50$ **do**
    $\text{MSE}_{\text{Test}} \leftarrow$ compute the MSE of the predictions in the test set, with the best parameters $\hat{\mu}^*_{k^*_{\text{model}}}$
    if $\text{MSE}_{\text{Test}} < \text{best}_{\text{MSE}_{\text{Test}}}$ **then**
      Store: $\text{best}_{\hat{S}_{\text{Test}}} \leftarrow \hat{S}_{\text{Test}}$
    end if
    $n \leftarrow n + 1$
  end while
  Get mean and std of the 50 MSE_{Test}
end for

Appendix E

Non-parametric models and estimation

E.1 Non-parametric models

Non-parametric models are widely used in numerous fields, for example, behavioral science [98], neuroimaging [99] and medicine [100].

For time-series modeling, non-parametric models are used to model the underlying intensity function of the time-series by either specifying a distribution with unknown parameters (e.g. Gaussian with unspecified mean and covariance) or by building a distribution-free model. In the latter, instead of fixing the model structure, we use latent variables that grow as necessary to fit the data. The former is more frequently found in the literature, where the underlying function is estimated by computing an approximation of the posterior distribution[1] - the distribution of the model parameters given the training data. To build such an approximation, the Bayesian method (see Section E.1.1) is combined with powerful inference methods that have been recently developed such as Markov chain Monte carlo (see Section E.2.1) and Variational Bayes (see Section E.2.2).

E.1.1 Bayesian method

The Bayesian method is used to solve supervised learning problems, i.e., problems in which both observations and labels are available to train the model. The model is trained using n pairs of continuous observations $o_t \in \mathbb{R}^p$ (with p features) and labels y_i, i.e., $\{o_t, y_t\}_{t=1}^n$. The model parameters are $\mu \in \mathbb{R}^p$. The Bayesian training phase consists of computing the posterior distribution using Bayes' rule,

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}} \iff \mathbb{P}[\mu|\boldsymbol{O}, y] \underset{Bayes}{=} \frac{\mathbb{P}[y|\boldsymbol{O}, \mu]\mathbb{P}(\mu)}{\mathbb{P}[y|\boldsymbol{O}]}, \qquad \boldsymbol{O} = [o_1, ..., o_n] \in \mathbb{R}^{p \times n}, \quad \text{(E.1)}$$

i.e., the distribution of the model parameters given the training data. In eq. (E.1), $\boldsymbol{O}$ is the matrix of all training data, also called *design matrix*. Whenever it is possible, some authors assume a *prior* dis-

[1] When it is not possible to compute the actual posterior distribution.

tribution over the parameters, to ease the training, which represents an *a priori* expectation about the parameters before taking a closer look at the observations.[2] Besides the prior, the posterior computation also depends on the model *likelihood* - the probability of the labels given the parameters and the observations. It represents how well the observations and chosen parameters (μ) support the labels.
The marginal likelihood, also called normalizing constant or evidence, is given by

$$\mathbb{P}[y|\boldsymbol{O}] = \int \text{likelihood} \times \text{prior} = \int \mathbb{P}[y|\boldsymbol{O}, \mu]\mathbb{P}(\mu)d\mu, \tag{E.2}$$

which corresponds to a marginalization over all possible parameter assignments.
To sum up, the posterior learns satisfactory properties for the parameters given the prior and the model likelihood so that later those parameters can be used to make valid predictions on unseen test data.

In non-Bayesian methods, the objective is to obtain the optimal parameters μ according to some criterion, usually through *maximum-likelihood estimation* (MLE). For example, in the *maximum-a-posteriori* criterion (MAP), the parameter μ chosen is the one that maximizes the posterior distribution. That is,

$$\hat{\mu}_{\text{MAP}} = \arg\max_{\mu} \quad \mathbb{P}[\mu|\boldsymbol{O}, y]. \tag{E.3}$$

However, in most models (e.g. in HMMs, Section 2.2.1.3), it is unfeasible to find the MLE analytically [101]. In those cases, numerical methods must be used. After obtaining the parameter estimates, we can use them to make predictions for unseen test data.

In Bayesian settings, on the other hand, we are not interested in obtaining estimates but rather in the exact posterior or an approximation of it. However, in most models, it is not easy to compute the posterior in eq. (E.1) given the complexity of the model likelihood. Besides, the computation of the evidence in eq. (E.2) is intractable in general. The high dimensionality of the integral[3] makes it unfeasible. These reasons have motivated researchers to seek an approximate distribution instead of computing the actual posterior. This is what *inference* does - given the combination of the prior and the training data, it chooses an approximate posterior distribution over the state of all functions considered.
One solution is to discretize the time domain, thus replacing high-dimensional integrals with summations [102]. Recently, some refined inference methods have been proposed, such as *Markov chain Monte Carlo* (MCMC) [103], which exploits the idea of thinned-based sampling, and *Bayesian variational inference* [44] [104] [105]. While Monte Carlo methods take advantage of randomization techniques to estimate complex integrals, variational Bayes tries to approximate integrals using different variational approaches. A brief theoretical overview of these two methods is given in Section E.2.1 and E.2.2, respectively.
After obtaining the exact posterior (or an approximation of it), in order to make predictions (y_*) about unseen test inputs (o_*), the *predictive distribution* is computed by averaging over all possible parameter

[2] When the prior is not restricted to the interval $[0, 1]$, in order to transform it into probabilities, a squashing function (e.g. the logistic function) should be used to restrict the outputs to that interval.
[3] The latent variable that represents the parameters grows exponentially when enumerating all possible latent assignments.

values, weighted by the posterior distribution. That is,

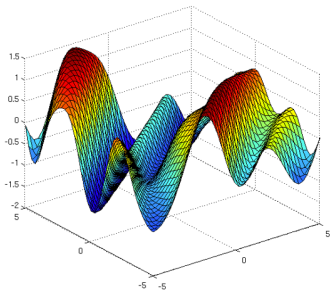
$$\mathbb{P}[y_*|o_*, \boldsymbol{O}, y] = \int \mathbb{P}[y_*|o_*, \mu]\mathbb{P}[\mu|\boldsymbol{O}, y]d\mu. \tag{E.4}$$

Non-parametric Bayesian inference methods can be applied to many distributions (e.g Gaussian, Gamma and Dirichlet distributions). Usually, supervised learning regression problems in areas such as machine learning and statistics use Gaussian processes, detailed in the next section, to model the data by taking advantage of their tractability property along with the Bayesian method to make inference [106].

E.1.2 Gaussian process

A Gaussian process (GP) is a stochastic process that consists of a set of random variables, any finite number of which have a joint Gaussian distribution [106].

Formally, a Gaussian process f with observations $o \in \mathbb{R}^p$ is completely specified by its mean $m(o) \in \mathbb{R}^p$ and covariance functions $k(o, o') \in \mathbb{R}^{p\times p}$. Figure E.1 illustrates an example of a Gaussian process in $\mathbb{R}^3$. If the index set of $f(o)$ is not infinite[4], then the Gaussian process is simply a single Gaussian distribution. In other words, GPs can be viewed as an infinite collection of Gaussians, which is mathematically achieved by defining its mean and covariance as functions instead of scalars.



$$\boxed{f(o) \sim \mathcal{GP}(m(o), k(o, o'))}$$

$$\begin{cases} m(o) = \mathbb{E}[f(o)] \\ k(o, o') = \mathbb{E}[f(o) - m(o)]\mathbb{E}[f(o') - m(o')] \end{cases}$$

Figure E.1: Example of a Gaussian process [107].

The Bayesian inference method previously explained is used to make inference in GPs [106] and the GP posterior is actually a distribution over functions[5]. After assigning prior probabilities to every function, the hyperparameters of the covariance function[6], which are responsible for the characteristic-length scale[7] of the GP, are "adjusted" by the inference method.

GP regression models are very popular due to their smoothing properties, that allow them to adapt to different data, and for their tractability, which means that the evidence in eq. (E.2) can be computed analytically and so does the posterior and predictive distributions in eqs. (E.1) and (E.4), respectively. The details about these computations, for GP regression models, can be found in [106].

In the next section we present a novel approach for time-series modeling, that combines both Gaussian processes introduced in Section E.1.2 and Cox processes described in Section 2.1.3.

[4]The index set is not infinite if o is not represented by $\mathbb{N}$ (countable infinite index set) nor $\mathbb{R}$ (uncountable infinite index set).

[5]The mean and covariance functions of the GP.

[6]The hyperparameters are the free parameters of the covariance function.

[7]The characteristic-length scale of a function defines its rapid/slow variations in time. A *short* length-scale is associated with quickly varying functions with very low noise level, which can lead to problems like *overfitting*, whereas a *long* length-scale is related with slowly varying signals with high noise level, which can cause *underfitting*.

E.1.3 Gaussian process modulated Cox processes

Gaussian process modulated Cox processes (GPCox) [35] [43] are non-parametric Bayesian models that can be used for time-series data. The event arrivals are modeled by a non-homogeneous Poisson process (see Section 2.1.2) with intensity function $\lambda(t)$.

The Cox process (see Section 2.1.3) is used to model the uncertainty in the intensity function as a continuous latent function[8] $f(t)$, i.e., $\lambda(t) = \rho(f(t))$, where $\rho(\cdot)$ is any non-negative squashing function, e.g., the sigmoid function. This explains the "Cox" part in "GPCox".

The "GP" part is due to the fact that a Gaussian process prior is placed in $f(t)$.

That is, the GP prior over n events of the temporal point process, represented by the latent function $\{f(o_t)\}_{t=1}^{n}$, is a multivariate Gaussian distribution, defined by its mean and covariance functions.

These assumptions make GPCox a notably useful model to capture the evolution of the intensity function when it exhibits smooth variations in time. For their flexibility, a lot of attention has been given to these models, in particular, when applied to machine learning problems [43]. However, these type of models cannot properly capture drastic changes in the intensity function, so they are not adequate for stationary or semi-stationary data. In Fig. 2.6(b), a sketch of a GPCox model is shown, where it is visible the smoothness of the intensity function.

The inference in GPCox models is done by approximating the posterior of the latent function $f(\cdot)$. In [35], the authors use a Markov Chain Monte Carlo with Metropolis-Hastings [108] [109]. Variational inference methods (see Section E.2.2) can also be exploited.

To recap, in Sections E.1.2 and E.1.3, we have shown two examples of non-parametric Bayesian models that can be used for time-series modeling. While the posterior inference in the first example was analytically tractable, the second requires additional methods to estimate the posterior distribution. Recall that the computation of the posterior distribution typically has no closed form solution because it requires high dimensional integration for the evidence in eq. (E.2). The next section addresses non-parametric estimation for models where inference about the posterior cannot be done analytically.

E.2 Non-parametric estimation

In this section we will cover two widely used methods for non-parametric estimation: Markov chain Monte Carlo and Variational Bayes.

E.2.1 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) combines Monte Carlo integration with *Markov chains*, introduced in Section 2.2.1. Monte Carlo integration is a sampling-based method for estimating expectations w.r.t an unknown posterior distribution[9]. For example, in the Bayesian method (see Section E.1), we

[8] Since we do not know the shape of the intensity function given only event data, GPCox uses a Cox process to model the intensity nonparametrically.

[9] Monte Carlo integration can only be used if we are able to sample from this unknown posterior.

are interested in obtaining expectations w.r.t the posterior, that can be expressed in terms of posterior expectations of a function $f(\mu)$ of the parameters μ. That is,

$$\mathbb{E}[f(\mu)] = \int f(\mu)\mathbb{P}[\mu|\boldsymbol{O}, y]d\mu = \frac{\int f(\mu)\mathbb{P}[y|\boldsymbol{O}, \mu]\mathbb{P}(\mu)d\mu}{\int \mathbb{P}[y|\boldsymbol{O}, \mu]\mathbb{P}(\mu)d\mu} \approx \frac{1}{n}\sum_{t=1}^{n} f(\mu_t), \tag{E.5}$$

where we replaced the posterior distribution by its definition from eq. (E.1).

Monte Carlo integration evaluates eq. (E.5) by drawing samples $\{\mu_t\}_{t=1}^{n}$ from the required posterior distribution $\mathbb{P}[\mu|\boldsymbol{O}, y]$ and then approximating the expectation by sample averaging (RHS of eq. (E.5)). The samples are drawn by a Markov chain that has the posterior as its stationary distribution[10] [110]. The trick is to find good properties for the transition matrix ($\boldsymbol{A}$) of the Markov chain such that sampling from $\boldsymbol{A}$ will eventually be the same as to sample from the posterior distribution $\mathbb{P}[\mu|\boldsymbol{O}, y]$. The transition $A(\mu_t \leftarrow \mu_{t-1})$ represents A_{ij} in eq. (2.7) and the states are the parameters of the posterior distribution (μ). One of the most commonly used methods to find the "ideal" transition matrix is the *Metropolis-Hastings* (MH) algorithm [108] [109], where we estimate the expectation by sampling the Markov chain recursively and accepting the samples according to an acceptance ratio. The advantage of MH is that no normalization is needed but, on the other hand, the drawback is that it takes a lot of time. Although MCMC is a widely used method for non-parametric estimation, because no normalization is needed, it takes too much time to converge for complex models or large datasets. One alternative is to use Variational Bayesian inference methods, that do not require sampling and therefore are much faster.

E.2.2 Variational Bayes

Variational Bayes is an approximate Bayesian inference method, which has boosted the inference in fields such as signal processing [111] and machine learning [112]. Approximate Bayesian inference methods approximate the complicated posterior $\mathbb{P}[\mu|\boldsymbol{O}, y]$ with a simpler proposal distribution $q_* \equiv q_*(\mu)$ [11], found as a result of an optimization problem

$$q_* = \underset{q\in\mathcal{Q}}{\arg\min}\Big(g\big(q(\cdot), \mathbb{P}[\mu|y]\big)\Big), \tag{E.6}$$

where the posterior $\mathbb{P}[\mu|\boldsymbol{O}, y]$ is reduced to $\mathbb{P}[\mu|y]$ because we are fixing a given input $\boldsymbol{O}$ and $g(\cdot)$ is a non-symmetric function that measures the "distance"[12] between two distributions. The optimization problem tries to find a distribution q_*, belonging to a set $\mathcal{Q}$ of special distributions, that is the closest to the exact posterior[13]. Then, q_* is used as an approximation of the exact posterior, for example, to predict future observations [113]. The set $\mathcal{Q}$ consists of a family of distributions (e.g. exponential distributions) where the parameters are easy to compute. In Variational Bayes, function $g(\cdot)$ is the *Kullback-Leibler*

[10] The stationary distribution is, roughly, the distribution the Markov chain will eventually converge to when the states are steady, after being sampled for a long time.

[11] The proposal distribution q_* depends only on the parameters μ, for a given input $\boldsymbol{O}$.

[12] It is not an actual distance but rather a divergence between the exact posterior and the approximate distribution q considered.

[13] In general, the exact posterior $\mathbb{P}[\mu|y]$ doesn't belong to set $\mathcal{Q}$.

(KL) divergence in a particular direction $d\mu$, defined as

$$g = \mathsf{KL}(q(\cdot)||\mathbb{P}[\cdot|y]) = \mathbb{E}_q\left[log\frac{q(\mu)}{\mathbb{P}[\mu|y]}\right] = \int_\mu q(\mu)log\left(\frac{q(\mu)}{\mathbb{P}[\mu|y]}\right)d\mu. \tag{E.7}$$

Assuming that the exact posterior $\mathbb{P}[\mu|y]$ cannot be obtained in closed form, we find the approximate distribution q that maximizes the *Evidence Lower Bound* (ELBO) instead. That is, eq. (E.7) can be further manipulated using Bayes' rule, the properties of logarithms and Jensen's inequality[14] to obtain[15,16]

$$g \propto_\mu -\mathsf{ELBO} \equiv -\int_\mu q(\mu)log\left(\frac{\mathbb{P}[\mu, y]}{q(\mu)}\right)d\mu = -\Big(\mathbb{E}_q\big[log(\mathbb{P}(\mu, y))\big] - \mathbb{E}_q\big[log(q(\mu))\big]\Big). \tag{E.8}$$

Since g is positive, minimizing the KL divergence in eq. (E.6) is the same as maximizing the ELBO. We can increase the ELBO by adjusting the proposal distribution q through the variational parameters. A common approximate Bayesian inference method is the *mean-field variational Bayes* (MFVB), which is widely used for time-series modeling and sequence prediction in many areas including neuroimaging [114] and biophysics [115]. MFVB methods use low-dimensional distributions for set $\mathcal{Q}$, defined as $\mathcal{Q} := \{q : q(\mu) = \prod_{j=1}^{J} q_j(\mu_j)\}$ and the KL divergence in eq. (E.7). In low-dimensional distributions, the parameter vector μ breaks down into components $\mu_1, ..., \mu_J$ and the approximate distribution $q(\mu)$ factorizes over these components.

The difference between approximate Bayesian inference methods (e.g. MFVB) and MCMC is that the latter will asymptotically get the right posterior as the Markov chain will always converge. On the contrary, the former is always an approximation. Other algorithms have been introduced over the years, which combine the advantages of MCMC and variational approximation [116].

[14] Jensen's inequality states that $log\big(\mathbb{E}(\cdot)\big) \geq \mathbb{E}\big(log(\cdot)\big)$.

[15] All elements in eq. (E.8) are known: $q(\mu)$ is a distribution from set $\mathcal{Q}$ and $\mathbb{P}(\mu, y)$ is the product of the likelihood with the prior.

[16] The remaining term is the logarithm of the marginal probability of y, $log\big(\mathbb{P}(y)\big)$. This term is not represented because it does not depend on q and is discarded in the optimization problem of eq. (E.6).

Appendix F

Other synthetic experiments

F.1 Analysis of the effect of the number of sequences

The aim of the test performed in this section is to assess how the number of sequences (N) affect the estimation accuracy. The synthetic data consists of sequences of $n = 100$ observations, sampled from a HSMM with $k = 3$ and $d_{max} = 4$, Poisson emissions and non-parametric state duration distributions. setup from eq. (4.31). To increase statistical significance, the estimation error is compared based on 100 independent Monte Carlo runs. In each run, the original matrices are the same, but by sampling from them we obtain different state and observation sequences. The number of sequences tested was $N = \{1, 2, 5, 10, 100\}$. To compare the results, we present the estimation error of the state and observation sequences separately, given by eq. (4.32) in Chapter 4, Section 4.4.4.3. In Table F.1, we present both the state and observation sequence estimation errors (from eq. (4.32)), the hit rate (from eq. (3.16)) and the percentage of total overlap between the distributions.[1]

Table F.1: Results of the sequence estimation with increasing number of sequences N, presented in the format mean $\pm$ std for the 100 Monte Carlo runs.

N	err(S)	err(O)	Hit rate	Overlap [%]
1	0.1372 ± 0.0631	3.7392 ± 0.3874	0.9771 ± 0.0162	0.0991
2	0.1257 ± 0.0412	3.8568 ± 0.2533	0.9825 ± 0.0105	0.1200
5	0.1229 ± 0.0253	3.8450 ± 0.1849	0.9843 ± 0.0062	0.1136
10	0.1217 ± 0.0199	3.8756 ± 0.1042	0.9848 ± 0.0046	0.1255
100	$\mathbf{0.1200 \pm 0.0047}$	3.8729 ± 0.0441	$\mathbf{0.9856 \pm 0.0011}$	0.1202

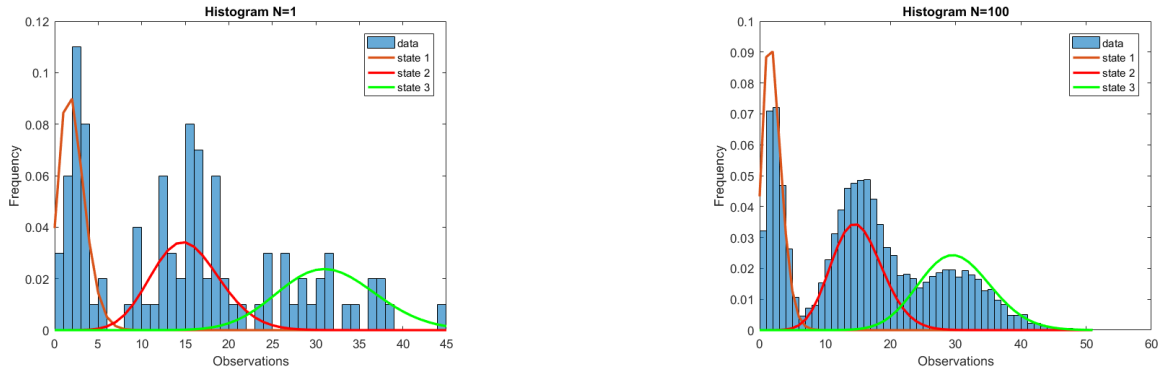
From inspection of Table F.1, we can conclude that as the number of sequences increases, the estimation error for the state sequence reduces and, simultaneously, the hit rate becomes closer to 1. We also observe small oscillations in the mean of the observation sequence estimation error. The percentage of overlap[2] between the Poisson distributions of all three states is the main reason for this error. To support

[1] These percentages were calculated for the first Monte Carlo run, as well as the corresponding histograms of Fig. F.1.
[2] Which is not directly changing with N, but rather with the observations from the sampled data.

Table F.2: Results of the parameter estimation with increasing number of sequences N.

N	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$	MSE($\hat{\boldsymbol{A}}$)	MSE($\hat{\boldsymbol{D}}$)	MSE($\hat{\boldsymbol{\pi}}$)
1	2.32 ± 0.31	15.04 ± 0.59	30.24 ± 1.19	5.41	27.21	198.03
2	2.33 ± 0.22	15.08 ± 0.46	29.99 ± 0.74	11.89	7.46	31.45
5	2.32 ± 0.12	14.98 ± 0.25	29.90 ± 0.51	0.41	3.62	11.97
10	2.32 ± 0.09	15.02 ± 0.19	30.04 ± 0.40	0.46	0.69	4.81
100	2.33 ± 0.03	15.02 ± 0.06	29.98 ± 0.12	**0.30**	**0.44**	**2.63**

these statements, Fig. F.1 shows the normalized histograms of the observations for $N = 1$ and $N = 100$. From Table F.1, we can observe that a larger total overlap is linked with higher errors in the observation sequence estimation, as expected. In Table F.2 we present the predicted rates of the Poisson distribu-



(a) (b)

Figure F.1: Histograms of the observations for: (a) $N = 1$, and (b) $N = 100$. The bars represent the frequency of the respective value in the observation sequence(s). The solid lines are the three Poisson probability distributions (one for each state) of the respective simulated data.

tions of each state and the mean-squared-error between the predicted multinomial distributions and the true distributions. Again, with increased N, the predicted means are not getting closer to the true means due to the oscillations in the total overlap percentage of the simulated data. We can observe that the estimate of $\hat{\boldsymbol{\pi}}$ is the one that takes more advantage of the increasing number of sequences N in terms of accuracy. This should come as no surprise, given that with higher N we have more examples of initial states which provide a better estimation of $\boldsymbol{\pi}$. The lowest MSE occurs, for all parameters, at $N = 100$.

F.2 Analysis of the effect of the sequence length

The aim of the test performed in this section is to assess how the sequence length (n) affects the estimation accuracy. The HSMM herein considered have the same setup as before and the synthetic data consists of one sequence ($N = 1$). In the conducted experiment, we vary the number of observations in the sequence: $n = \{10, 100, 1000, 10000\}$. The effect of increasing the sequence length is analyzed in terms of the estimation errors and hit rate (Table F.3) and the accuracy in the estimation of the Poisson

rates and mean-squared-error of the estimated parameters (Table F.4).

Table F.3: Results of the sequence estimation with increasing sequence length n.

n	err(S)	err(O)	Hit rate
10	0.2752 ± 0.2255	3.5675 ± 1.5045	0.8847 ± 0.1026
100	0.1372 ± 0.0631	3.7392 ± 0.3874	0.9771 ± 0.0162
1000	0.1218 ± 0.0186	3.8689 ± 0.1210	0.9848 ± 0.0045
10000	$\mathbf{0.1205 \pm 0.0061}$	3.9069 ± 0.0367	$\mathbf{0.9854 \pm 0.0015}$

From inspection of Table F.3, one can conclude, as foreseen, that the sequence length improves the estimation accuracy: the higher the number of observations, the smaller the state sequence estimation error is and, consequently, the hit rate approaches 1. As for the observation sequence estimation error, we can again conclude that it does not depends neither on N nor on n, but rather on the observations sampled from the data and the overlap percentage of the resulting Poisson distributions.

Table F.4: Results of the parameter estimation with increasing sequence length n.

n	$\hat{\lambda}_1$	$\hat{\lambda}_2$	$\hat{\lambda}_3$	MSE$(\hat{\boldsymbol{A}})$	MSE$(\hat{\boldsymbol{P}})$	MSE$(\hat{\boldsymbol{\pi}})$
10	6.27 ± 7.37	14.98 ± 2.26	24.53 ± 8.10	26.80	9.10	4.30
100	2.32 ± 0.31	15.04 ± 0.59	30.24 ± 1.19	0.05	0.27	2.00
1000	2.33 ± 0.09	15.00 ± 0.16	30.06 ± 0.32	0.00	0.02	2.00
10000	2.33 ± 0.03	15.01 ± 0.05	30.00 ± 0.11	$\mathbf{0.00}$	$\mathbf{0.00}$	$\mathbf{0.66}$

It is interesting to notice, from Table F.4, that the HSMM only needs $n = 100$ training samples to obtain a fair estimate of the Poisson means. In fact, the only values which are clearly disparate are the ones for $n = 10$, for which the lack of data does not allow a fair estimation of the means. Also, the MSE of the state transition matrix in Table F.4 is the one that takes more advantage of the increasing sequence length n in terms of accuracy. Moreover, by comparing these results for $n = 10000$ and $N = 1$ with the results for $N = 100$ and $n = 100$ from the previous section, we conclude that, except for the estimate of $\boldsymbol{\pi}$, all results are better with just one sequence. The total number of observations (given by $N \times n$) is the same, however, 100 transitions are "lost" in the setting from the previous section, which worsen the results.