Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

Regularized hidden Markov modeling with applications to wind speed predictions in offshore wind

Anna Haensch [a], Eleonora M. Tronci [b], Bridget Moynihan [b], Babak Moaveni [b,*]

[a] *Data Intensive Study Center, Tufts University, Medford, USA*
[b] *Department of Civil and Environmental Engineering, Tufts University, Medford, USA*

ARTICLE INFO

Communicated by M. Beer

Keywords:
Hidden Markov models
Wind speed
Offshore wind
Tykhonov Regularization
Cross-validation learning

ABSTRACT

Offshore wind power is rapidly becoming an essential component of the transition to clean energy. As turbine design capacities continue to increase, there is a growing interest in monitoring both individual turbines and entire wind farms to ensure their performance while also reducing the levelized cost of energy. However, obtaining reliable and comprehensive data on these structures can be challenging, as it often requires costly and potentially dangerous installation procedures and significant computational resources. Therefore, it is critical to predict the needed information to properly assess the performance of offshore wind turbines when not available. To address these challenges, this paper introduces a modified Hidden Markov Model (HMM) framework-based strategy and a companion Python library, `Hela`. The proposed HMM-based framework incorporates a "smart" initialization strategy and regularization to overcome some of the limitations of applying HMMs to experimental cases. This Python library introduced in this paper is a highly flexible and customizable HMM library for training and inference. This work is all carried out within the use-drive context of offshore wind.

1. Introduction

As the pursuit of climate solutions gains momentum, offshore wind power is poised to become an increasingly vital component of the transition to cleaner energy sources [1]. The United States Offshore Wind (OSW) pipeline currently boasts a potential generating capacity of over 40,000 MW, necessitating the construction of thousands of OSW turbines [1]. Furthermore, in order to maximize energy production from wind, turbine designs are growing in size and capacity [2]. This trend towards larger turbines and increased installations has created a pressing need for enhanced structural reliability to ensure safe and affordable clean energy. One area of particular importance in wind turbine engineering is the extension of turbine lifetimes, which involves utilizing condition monitoring techniques to detect and prevent faults and failure mechanisms [3–6]. By extending the lifespan of turbines through monitoring campaigns, the levelized cost of energy can be reduced, further facilitating and accelerating climate solutions [7].

Instrumentation of civil infrastructures allows for continuous monitoring and analysis of the health and behavior of structural systems starting from their acquired measurements response [8]. Known as Structural Health Monitoring (SHM), this field aims to monitor the structural condition of infrastructures and to detect damage in order to extend the lifetimes of structures [9].

SHM applications rely on structural instrumentation to monitor and record the response of infrastructure to different loading, operational, and environmental conditions. Wind turbine instrumentation campaigns typically include the installation of accelerometers and strain gauges to measure deformation, material loads, and bending moments [10]. These instruments collect continuous

[*] Corresponding author.
E-mail address: babak.moaveni@tufts.edu (B. Moaveni).

https://doi.org/10.1016/j.ymssp.2024.111229
Received 4 April 2023; Received in revised form 5 December 2023; Accepted 4 February 2024
Available online 17 February 2024

measurements from the turbine structure at high frequency, providing a real-time and continuous stream of information. Additionally, the wind turbine Supervisory Control and Data Acquisition (SCADA) system provides synchronous data on environmental conditions and turbine operational conditions such as wind speed and power output. Altogether, such instrumentation can be used in condition monitoring methods in order to track the behavior of the turbine under varying environmental loads and to detect and prevent faults [10,11]. Both physics-based and data-driven methods rely on this monitoring information, and they have been introduced and developed for applications to SHM of wind turbines [12,13].

The availability of various data types (dynamic quantities, environmental and operational information), as mentioned earlier, enables robust physics-based modeling and model updating as well as system identification and performance assessment of offshore wind turbines [14]. However, not all data channels may be readily available for use. Hence, predicting and imputing data from certain channels becomes necessary to assess the potential lack of information and accomplish comprehensive SHM of a wind turbine. In particular, data from the SCADA system, such as wind speed readings, sometimes contain missing periods of data. Wind speed is a crucial parameter for the operational analysis of wind turbines as it reflects the incident loading on the turbine and is directly linked to turbine power output and other controller settings. Thus, wind speed measurements play a vital role in describing and predicting the performance of wind turbines. The absence of wind speed data during certain periods prevents the analysis of the structure with respect to environmental loading conditions, disabling a detailed assessment of the structure performance.

In addition to missing data, some turbines provide only low-resolution wind speed readings, such as 10-minute average values. Higher resolution readings would enable further understanding of the incident loading and environmental conditions experienced by the turbine. Therefore, it is essential to define a strategy that can robustly and reliably predict wind speed information whenever it is not directly measurable using available high-resolution measurements such as strain or acceleration quantities.

Several works have been presented in the literature for wind speed estimation at offshore sites using various data-driven methods. Many works have focused on future prediction of wind speeds for power output forecasting [15–18]. Other works focus on farm-wide wind estimation in order to understand the effects of turbine wakes on power and turbine health [19]. None of these works are concerned with reconstructing periods of missing wind speed readings or predicting the wind speed incident on a single turbine by utilizing structural response measurements in modeling. This work presents data-driven modeling of wind speed readings using structural response measurements as input data features, with the goal of reconstructing the time-history of incident wind speed for use in SHM applications.

The Hidden Markov Model (HMM) is a type of statistical machine learning model that relies on both spatial and temporal variations. At its core, the HMM assumes that the system being modeled consists of a set of observable values and an underlying sequence of latent states, which are themselves a Markov process. That is, the latent state at any time is determined only by the latent state immediately prior. The time-based nature of HMMs makes them particularly well-suited to the task of condition monitoring and have been used with great effect as a tool for measuring mechanical wear [20,21] and electrical faults [22] specifically in rotation machines [23]. In the domain of wind power, algorithms based on HMMs have been used in fault detection in tandem with onboard control and alarm systems [24,25] as well as estimating relevant structural and environmental quantities of interest, such as power generation [26] and wind speed [27].

HMM has already shown promising results for other study cases related to wind speed tailored for forecasting applications. In [27], the authors propose a wind speed correction framework based on an enhanced HMM strategy to correct the wind speed forecasting results obtained using the weather research and forecasting model. In particular, the HMM is modified with the addition of a fuzzy C-means cluster to properly divide the hidden state space (the forecasting error) of HMM into the optimal number of discrete hidden states, and to make full use of the predicted wind speed, then the emission probability of HMM is improved as continuous using kernel density estimation.

This work introduces a new regularized Hidden Markov Model with an informed initialization strategy that makes it suitable for real data applications. The model is proposed in combination with a Python library, `Hela` [28], and presents an application of the library to models for operational wind speed prediction from an offshore wind turbine. Specifically, this work focuses on utilizing HMMs to predict wind speed readings using structural dynamic readings as model inputs (accelerations and strain-related quantities). The addressed strategy uses HMMs to perform wind speed classification in which each point in time is assigned a certain class according to wind speed magnitude. The proposed framework differs from existing methods in the literature, such as the one proposed by [28], because it is not designed for forecasting but rather for forecast-agnostic estimation derived from dynamic and structural quantities, and the number of hidden states is considered as a user-driven hyperparameter instead of a learned model parameter.

Two weeks of data are available from an offshore wind turbine instrumented with accelerometers, strain gauges, and a SCADA system, providing measurements at 25 Hz, and 10 Hz, respectively. The true wind speed readings are used to create wind speed bins for each step in the time-history, and these states are mapped onto the learned hidden states of the HMMs in order to measure model accuracy. This modeling is useful for time periods where the wind speed readings or SCADA system data are not available or only available at low sampling rates. The main contributions of this paper are as follows:

1. A randomly seeded HMM is enhanced using a cross-validation strategy that results in smart seeding of the model.
2. A regularization step is added to the HMM training to deal with ill-conditioning of the covariance matrix.
3. The `Hela` repository for hybrid HMM modeling is introduced with a clear use-driven workflow demonstration in the domain of offshore wind.

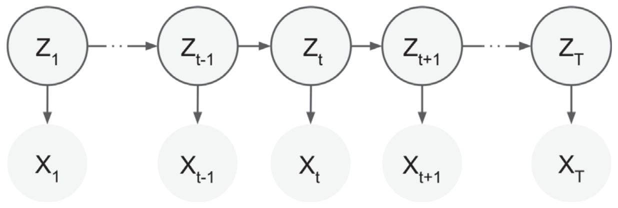


Fig. 1. HMM graphical structure.

This paper is structured as follows. Sections 2.1–2.4 will describe the theoretical underpinnings of the HMM in the context of model parametrization and proposed modified learning strategy, and Section 2.5 will describe how these tools have been implemented in the open-source `Hela` library. Section 3 will focus on the application of these tools to the domain of offshore wind, describing the datasets, experimental methodology, and results for wind speed prediction with HMMs. Finally, Section 4 will summarize the overall findings, model meta-analysis, and potential pathways forward.

2. Hidden Markov modeling

2.1. Model parametrization

Suppose we have a set of observations related to structural dynamics recorded over a period of time from $t = 1$ to $t = T$. Viewing the set of T multi-dimensional observations as random variables, $X_{1:T} := X_1, \dots, X_T$, the concrete goal of HMM training is to find the set of model parameters, θ, that maximize the likelihood function,

$$L(\theta | X_{1:T}) = p(X_{1:T} \mid \theta). \tag{1}$$

Running in parallel to the set of observations, there is also a set of hidden states, $Z_{1:T} := Z_1, \dots, Z_T$ which encode some latent behavior in the system. Marginalizing over the possible sequences of hidden states, this likelihood can also be written as

$$L(\theta | X_{1:T}) = \int p(X_{1:T}, Z_{1:T} \mid \theta)\; dZ_{1:T} \tag{2}$$

where the integrand above is called the complete data likelihood.

In an HMM, the sequence of hidden states must satisfy the Markov property; that is, the hidden state at any time t is dependent on the hidden state immediately prior but is independent of the hidden state evolution prior to time $t-1$. This conditional dependence can be viewed as a directed graph as in Fig. 1, where Z_t denotes the hidden state, and the X_t denotes the observed data at time t.

Given the underlying Markov structure, the probability in Eq. (2) can be written as

$$p(X_{1:T}, Z_{1:T} \mid \theta) = p_0(Z_1 \mid \theta) \cdot \prod_{t=2}^{T} p_a(Z_t \mid Z_{t-1}, \theta) \cdot \prod_{t=2}^{T} p_b(X_t \mid Z_t, \theta). \tag{3}$$

For a discrete set of hidden states $\{h_1, \dots, h_N\}$, the probabilities in this expression can be parameterized as

$$\begin{aligned} \pi_i &:= p_0(Z_1 = h_i \mid \theta) \\ A_{ij} &:= p_a(Z_t = h_j \mid Z_{t-1} = h_i, \theta) \\ B_i(x_t) &:= p_b(X_t = x_t \mid Z_t = h_i, \theta), \end{aligned} \tag{4}$$

where π is an N-dimensional vector, A is an $N \times N$ matrix, and B is a set of N distinct parameter families dependent on the underlying distribution of $X_{1:T}$. Therefore, the full model is parameterized as

$$\theta = \{\pi, A, B\} \tag{5}$$

where π, A, and B give the initial state, transition, and emission probabilities of the model.

In the case of dynamic system data and environmental data, a reasonable assumption is that $X_{1:T}$ consists of continuous observations drawn from a k-dimensional, M-component Gaussian mixture model (GMM). Recall for a k-dimensional GMM, the mth component has a prior mixture weight w_m and

$$x_t \sim \mathcal{N}(\mu_m, \Sigma_m) \tag{6}$$

where μ_m and Σ_m are the means and covariance matrices of the underlying distribution for the mth component. Therefore, the emission probability can be expressed more precisely as

$$B_i(x_t) = \sum_{m=1}^{M} w_{mi} \cdot \mathcal{N}(x_t; \mu_{mi}, \Sigma_{mi}) \tag{7}$$

where w_{mi} is the prior mixture weight of the mth component in the ith hidden state, μ_{mi} and Σ_{mi} are defined similarly.

Unfortunately, the integral in Eq. (2) is often quite difficult to compute in practice. Even in the cases where it is possible to compute analytically, it can still be computationally intractable. Moreover, solving the maximization problem also involves optimizing the quantity on the right-hand side, which can often involve complicated non-convex optimization. Fortunately, the Markov property and dynamic programming allow this complicated problem to be bypassed.

Rather than directly computing the marginalized integral, it is possible to use a variant of the Baum–Welch Expectation–Maximization (EM) algorithm to find the optimal set of model parameters [29]. This algorithm consists of an expectation step and a maximization step whereby it is possible to iteratively reach at a locally optimal set of model parameters. The expectation step necessitates the introduction of an auxiliary function in terms of current model parameters, θ', and new model parameters, θ,

$$Q(\theta,\theta') = \mathbb{E}_{Z_{1:T}|X_{1:T},\theta'}[\log p(X_{1:T}, Z_{1:T} \mid \theta)] \tag{8}$$

which gives the expected value of the complete data log likelihood with respect to the sequence of hidden states $Z_{1:T}$ given $X_{1:T}$ and the current model parameters. For the maximization step, a set of new model parameters, θ, is found which maximize Q as a function of θ'. In what follows, we show that iteratively maximizing this auxiliary function is sufficient to find a local maximum for the likelihood function. Since it will simplify the discussion significantly, we will work in terms of the log-likelihood function,

$$\mathcal{L}(\theta|X_{1:T}) := \log p(X_{1:T} \mid \theta) = \log \int p(X_{1:T}, Z_{1:T} \mid \theta)\, dZ_{1:T}. \tag{9}$$

Theorem 2.1. *If $Z_{1:T}$ is drawn from a discrete probability distribution, then*

$$Q(\theta,\theta') \le \mathcal{L}(\theta|X_{1:T}) \tag{10}$$

for any fixed set of model parameters θ'. Moreover,

$$Q(\theta,\theta') \ge Q(\theta',\theta') \tag{11}$$

implies

$$\mathcal{L}(\theta|X_{1:T}) \ge \mathcal{L}(\theta'|X_{1:T}). \tag{12}$$

Proof. Suppose that $Z_{1:T}$ is drawn from a discrete probability distribution. From Bayes' rule, we know that

$$\log p(X_{1:T} \mid \theta) = \log p(X_{1:T}, Z_{1:T} \mid \theta) - \log p(Z_{1:T} \mid X_{1:T}, \theta) \tag{13}$$

Multiplying both sides of this equation by $p(Z_{1:T} \mid X_{1:T}, \theta')$ and integrating over the space of all possible $Z_{1:T}$, we get

$$\mathcal{L}(\theta|X_{1:T}) = Q(\theta,\theta') - \int \log p(Z_{1:T} \mid X_{1:T},\theta) \cdot p(Z_{1:T} \mid X_{1:T},\theta')\, dZ_{1:T}. \tag{14}$$

The negative term on the right-hand side above is just the entropy of the random variables X and Z, which in the case of discrete Z is always positive, from which we obtain Eq. (10).

The equation above still holds if we replace θ with θ', in particular,

$$\mathcal{L}(\theta'|X_{1:T}) = Q(\theta',\theta') - \int \log p(Z_{1:T} \mid X_{1:T},\theta') \cdot p(Z_{1:T} \mid X_{1:T},\theta')\, dZ_{1:T}. \tag{15}$$

Subtracting these equations from one another we obtain

$$\begin{aligned}\mathcal{L}(\theta|X_{1:T}) - \mathcal{L}(\theta'|X_{1:T}) \;=\; & Q(\theta,\theta') - Q(\theta',\theta') + \cdots \\ & \cdots + \int \log \frac{p(Z_{1:T} \mid X_{1:T},\theta')}{p(Z_{1:T} \mid X_{1:T},\theta)} \cdot p(Z_{1:T} \mid X_{1:T},\theta')\, dZ_{1:T}.\end{aligned} \tag{16}$$

However, we recognize the second term on the right hand side above as the Kullback–Leibler divergence,

$$D_{KL}(p(Z_{1:T} \mid X_{1:T},\theta') \,||\, p(Z_{1:T} \mid X_{1:T},\theta), \tag{17}$$

which is always greater than or equal to 0. In particular, this shows that if Q increases, then $\mathcal{L}$ increases at least as much. □

Corollary 2.2. *If $Z_{1:T}$ is drawn from a discrete probability distribution, then repeated iterative improvements of Q will eventually lead to a local maximum for $\mathcal{L}$.*

Proof. Since Q is bounded above by 0 and since the input space of Q is compact, iteratively improving Q will eventually lead to a critical point for Q. That is, we will reach a set of model parameters, θ^*, such that

$$Q(\theta,\theta^*) \le Q(\theta^*,\theta^*) \tag{18}$$

for any choice of θ. If we consider $Q(\theta,\theta^*)$ as a function in θ, then this is equivalent to

$$\frac{d}{d\theta}\left[Q(\theta,\theta^*)\right]\Big|_{\theta^*} = 0. \tag{19}$$

On the other hand, taking the derivative of $\mathcal{L}$ with respect to θ and evaluating at θ^* gives

$$\frac{d}{d\theta}\mathcal{L}(\theta|X_{1:T}) = \frac{1}{L(\theta^*|X_{1:T})} \sum_{Z_{1:T}} \frac{d}{d\theta}\left[p(Z_{1:T}, X_{1:T} \mid \theta)\right]\Big|_{\theta^*} \tag{20}$$

Now, combining Eqs. (19) and (20) above, we obtain

$$\begin{aligned}\frac{d}{d\theta}\left[Q(\theta,\theta^*)\right]\Big|_{\theta^*} &= \frac{d}{d\theta}\left[\int \log p(X_{1:T}, Z_{1:T} \mid \theta) p(Z_{1:T} \mid X_{1:T}, \theta^*)\, dZ_{1:T}\right]\Big|_{\theta^*} \\ &= \sum_{Z_{1:T}} p(Z_{1:T} \mid X_{1:T}, \theta^*) \cdot \frac{d}{d\theta}\left[\log p(Z_{1:T}, X_{1:T} \mid \theta)\right]\Big|_{\theta^*} \\ &= \sum_{Z_{1:T}} \frac{p(Z_{1:T} \mid X_{1:T}, \theta^*)}{p(Z_{1:T}, X_{1:T} \mid \theta^*)} \cdot \frac{d}{d\theta}\left[p(Z_{1:T}, X_{1:T} \mid \theta)\right]\Big|_{\theta^*} \\ &= \frac{1}{p(X_{1:T} \mid \theta^*)} \cdot \sum_{Z_{1:T}} \frac{d}{d\theta}\left[p(Z_{1:T}, X_{1:T} \mid \theta)\right]\Big|_{\theta^*} \\ &= \frac{1}{L(\theta^*; X_{1:T})} \cdot \frac{d}{d\theta}\left[L(\theta; X_{1:T})\right]\Big|_{\theta^*} \\ &= \frac{d}{d\theta}\left[\mathcal{L}(\theta|X_{1:T})\right]\Big|_{\theta^*},\end{aligned} \tag{21}$$

from which it follows that θ^* is a critical point for $\mathcal{L}$. Therefore, we have shown that iteratively improving Q will eventually lead to a local maximum, θ^*, which will also be a critical point for $\mathcal{L}$. Moreover, from Theorem 2.1 we know that this critical point will be a local maximum. □

With successive iterations of the expectation and maximization steps, the sequence

$$\{Q(\theta^i, \theta^{i-1}) : i \in \mathbb{Z}^+\} \tag{22}$$

is bounded and monotone increasing. Since it is bounded and therefore guaranteed to converge, we will eventually reach a local maximum for $\mathcal{L}$ by Corollary 3.2. However, it still remains to be shown how to compute the expected value in the expectation step and how to carry out the maximization described in the maximization step.

2.2. The expectation step

On its face, computing the expected value in Eq. (8) appears to be quite difficult since it should involve computing a deeply nested integral of a joint probability. However, from the graphical model in Fig. 1, it can be verified using the d-separation criterion that $X_{1:t}$ and $X_{t+1:T}$ are conditionally independent given Z_t and therefore the joint probability in question can be written as

$$p(Z_t, X_{1:T} \mid \theta) = p(Z_t, X_{1:t} \mid \theta) \cdot p(X_{t+1,T} \mid Z_t, \theta) \tag{23}$$

allowing for the use of dynamic programming to avoid direct computation. To do so, define

$$\alpha_t(z_t) := p(Z_t = z_T, X_{1:t} = x_{1:t} \mid \theta) \tag{24}$$

and

$$\beta_t(z_t) := p(X_{t+1:T} = x_{t+1:T} \mid Z_t = z_t, \theta), \tag{25}$$

for a fixed set of observations $x_{1:T}$. Using Eq. (3),

$$\alpha_1(z_1) = p_0(z_1) \cdot p_b(x_1 \mid z_1) \tag{26}$$

and define remaining terms recursively as

$$\alpha_t(z_t) = p_b(x_t \mid z_t) \int p_a(z_t \mid z_{t-1}) \cdot \alpha_{t-1}(z_{t-1})\, dz_{t-1} \tag{27}$$

for $1 < t \le T$. Similarly, we initialize β with

$$\beta_T(z) = 1 \tag{28}$$

and define remaining terms recursively as

$$\beta_t(z_t) = \int p_b(x_{t+1} \mid z_{t+1}) \cdot p_a(z_{t+1} \mid z_t) \cdot \beta_{t+1}(z_{t+1})\, dz_{t+1} \tag{29}$$

for $1 \le t < T$, taken in descending order. These recursively defined values for α and β are the core of the *forward–backward algorithm* that is used to compute the expectation step [30]. The forward-backward algorithm considers the possibility of each

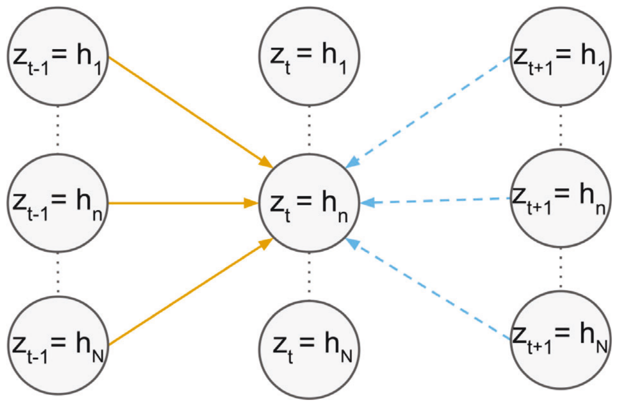


Fig. 2. This schematic shows a single step of the forward–backward algorithm in terms of the forward pass as solid orange lines and the backward pass as dashed blue lines.

hidden state, h_i, at each time, t, by computing the cumulative likelihood of arriving in state h_i given any of the possible prior states (this is the forward part, seen as solid orange lines in Fig. 2) and any of the possible subsequent states (this is the backward part, seen as dashed blue lines in Fig. 2).

With values for α and β in hand, using Eq. (23), Bayes' rule, and marginalizing over the hidden state space,

$$p(Z_t = z \mid x_{1:T}, \theta) = \frac{p(z, x_{1:T} \mid \theta)}{\int p(z', x_{1:T} \mid \theta)\, dz'} = \frac{\alpha_t(z) \cdot \beta_t(z)}{\int \alpha_t(z') \cdot \beta_t(z')\, dz'}. \tag{30}$$

Since the present discussion is only concerned with hidden states drawn from a discrete state space, define $\langle Z_t \rangle$ as the $N \times 1$ vector, whose ith entry is $p(Z_t = h_i \mid x_{1:T}, \theta)$, which can be written in terms of α and β as

$$\langle Z_t \rangle_i = \frac{\alpha_t(h_i) \cdot \beta_t(h_i)}{\sum_{j=1}^{N} \alpha_t(h_j) \cdot \beta_t(h_j)}. \tag{31}$$

Similarly, define the $N \times N$ vector $\langle Z_{t-1} Z_t \rangle$ whose ij^{th} entry is $p(Z_{t-1} = h_i, Z_t = h_j \mid x_{1:T}, \theta)$, which can be written in terms of α and β as

$$\langle Z_{t-1} Z_t \rangle_{ij} = \frac{\alpha_{t-1}(h_i) \cdot A_{ij} \cdot \prod_{k=1}^{K} p_k(x_t \mid h_j) \cdot \beta_t(h_j)}{\sum_{j'=1}^{N} \alpha_{t-1}(h_i) \cdot A_{ij'} \cdot \prod_{k=1}^{K} p_k(x_t \mid h_{j'}) \cdot \beta_t(h_{j'})}. \tag{32}$$

These quantities in Eqs. (31) and (32) are often referred to as γ and ξ, respectively, but here the notation of [31] is adopted. Note that the $\langle \cdot \rangle$ terms here are taken with respect to the θ' model parameters.

Combining all of the terms above, the expected value can now be efficiently computed as

$$\begin{aligned} \mathbb{E}_{Z_{1:T} \mid X_{1:T}, \theta'} \left[\log p(X_{1:T}, Z_{1:T} \mid \theta)\right] = &\sum_{i=1}^{N} \log \pi_i \cdot \langle Z_t \rangle_i + \cdots \\ &\dots + \sum_{t=2}^{T} \sum_{i=1}^{N} \sum_{j=1}^{N} \log A_{ij} \cdot \langle Z_{t-1} Z_t \rangle_{ij} + \cdots \\ &\dots + \sum_{t=1}^{T} \sum_{i=1}^{N} \log B_i(x_t) \cdot \langle Z_t \rangle_i, \end{aligned} \tag{33}$$

from which it becomes possible to maximize this expression by maximizing its three component parts.

2.3. The maximization step

For a fixed set of model parameters, θ', the goal is now to find a set of model parameters, θ^*, which maximize the expected value in Eq. (33). Once obtained, the current model parameters are updated, and one iteration of EM is complete. The previous section gave a simplified expression for the expected value, separated into three components describing the contribution of the initial state, the transition, and the emission probabilities to the conditional expected value. Because of the linearity of the expected value, each of these terms in the expected value can be treated as a separate optimization problem.

In some cases, certain constraints must be satisfied; for example, the initial state and the transition probabilities are subject to some additive constraints. Because it is of most interest to this work, subsequent discussion of parameter maximization will focus on updating Gaussian emission parameters (i.e. the third term on the right-hand side of Eq. (33)). Other update equations will follow similarly.

Combining Eqs. (7) and (33), the third component of the maximization problem is solved by

$$w^*, \mu^*, \Sigma^* = \underset{w,\mu,\Sigma}{\operatorname{argmax}} \sum_{t=1}^{T} \sum_{i=1}^{N} \log \sum_{m=1}^{M} w_{mi} \cdot \mathcal{N}(x_t; \mu_{mi}, \Sigma_{mi}) \cdot \langle Z_t \rangle_i \tag{34}$$

where w is an $M \times N$ array of GMM component weights, μ is a $M \times k \times k$ array of means, and Σ is an $M \times N \times k \times k$ array of variance terms. The general strategy will be to optimize these terms one by one by computing the derivative of the argument in Eq. (34) with respect to each of the GMM parameters, W, μ and Σ and then solving for the maximum.

For example, the optimal weight parameters, which are subject to the constraint,

$$\sum_{m=1}^{M} w_{mi} = 1 \tag{35}$$

are obtained by computing the gradient of

$$\sum_{t=1}^{T} \sum_{i=1}^{N} \log \sum_{m=1}^{M} w_{mi} \cdot \mathcal{N}(x_t; \mu_{mi}, \Sigma_{mi}) \cdot \langle Z_t \rangle_i - \lambda \left(-1 + \sum_{m=1}^{M} w_{mi} \right), \tag{36}$$

with respect to each of the w_{mi} and the Lagrange multiplier, λ, and solving. In this way, the optimal weight parameter is obtained,

$$w^*_{mi} = \frac{\sum_{t=1}^{T} \langle Z_t X_t \rangle_{mi}}{\sum_{t=1}^{T} \langle Z_t \rangle_i} \tag{37}$$

for $1 \le i \le N$ and $1 \le m \le M$, where

$$\langle Z_t X_t \rangle_{mi} = \frac{w_{mi} \cdot \mathcal{N}(x_t; \mu_{mi}, \Sigma_{mi}) \cdot \langle Z_t \rangle_i}{\sum_{m'=1}^{M} w_{mi} \cdot \mathcal{N}(x_t; \mu_{im'}, \Sigma_{im'})}. \tag{38}$$

In the case of Σ this update equation is given by

$$\Sigma^*_{mi} = \frac{\sum_{t=1}^{T} \langle Z_t X_t \rangle_{mi} \cdot (x_t - \mu_{im}) \cdot (x_t - \mu_{im})^\intercal}{\sum_{t=1}^{T} \langle Z_t X_t \rangle_{mi}} \tag{39}$$

where $\cdot^\intercal$ denotes the vector transpose. For the curious reader, full details of this derivation are included in Appendix A.1, but it will be pointed out here that in order to arrive at this update equation it is necessary not only that Σ_{mi} be non-singular, but that it be well-conditioned.

In the case of numerical processes such as EM, an ill-conditioned matrix (e.g. one with large condition number of order 10^n) can lead to a loss in up to n digits of numerical accuracy. This, in turn, can lead to erratic convergence behavior that appears to violate Theorem 2.1 and Corollary 2.2. To overcome this loss of precision in a numerical context, a common approach is to use Tykhonov regularization [32]. In this setting, Tykhonov regularization is equivalent to adding a small scalar multiple of the identity matrix to Σ_{im}. Therefore with regularization, the covariance matrix Σ_{im} is replaced by

$$\Sigma^{reg}_{im} = \Sigma_{im} + r \cdot \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}. \tag{40}$$

The addition of this so-called "nugget" has the effect of increasing the noise on the observation data, so care must be taken in choosing a small scalar term that simultaneously reduces the condition number without blowing up the noise. Some sample regularization terms are shown in Fig. 10.

2.4. Learning with cross-validation

To recap, the goal of EM is to iteratively maximize the conditional expected value until a set of model parameters is reached that maximizes the likelihood. A well-known shortcoming of the EM is that convergence to a locally optimal solution is all that is guaranteed under the algorithm. In practice, this means that using a truly random seed is likely to yield a model that converges to a locally, but not globally, optimal solution and takes longer than necessary to do so. Therefore, rather than using truly random seeding, it is sensible to use a "smart" seed. One way to improve the eventual likelihood is to start from a well-seeded set of model parameters. Absent any priors, suppose models $\theta_1, \ldots, \theta_C$ are initialized by randomly seeding each of the parameters and are trained on distinct observation subsets $X^1_{1:T}, \ldots, X^C_{1:T}$ (see Fig. 11). In this case, one would expect these processes to converge to a set of C distinct optimal models, $\theta^*_1, \ldots, \theta^*_C$ with a range of likelihoods. By selecting the model from among this set that has the highest complete data likelihood, say θ^*_{opt}, a second round of EM can be carried out, this time beginning from a more well-informed set of priors. In this way, a higher likelihood model, θ^*, can eventually be reached. In principle, this process could be carried out many times by taking iterative splits of the training data, but in practice the returns will eventually diminish. The process followed for the proposed cross-validation strategy is described in Fig. 3, considering a 3-fold example.

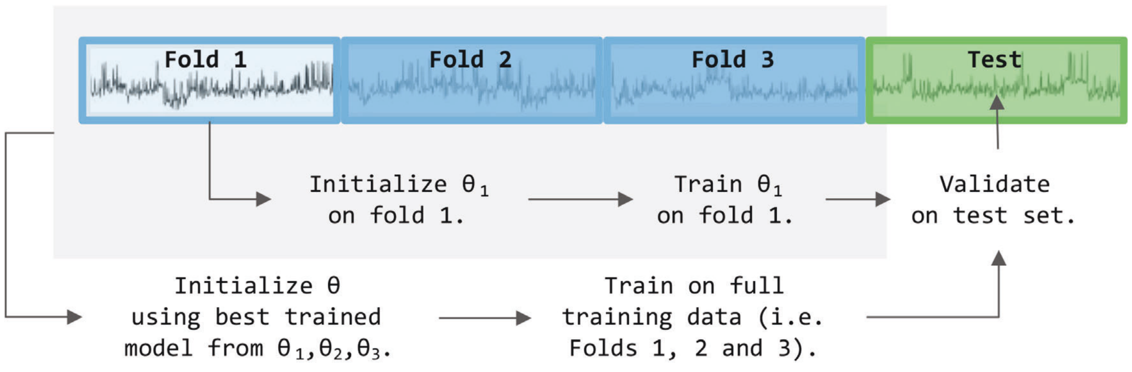

Fold 1
Fold 2
Fold 3
Test
Initialize θ₁ on fold 1.
Train θ₁ on fold 1.
Validate on test set.
Initialize θ using best trained model from θ₁,θ₂,θ₃.
Train on full training data (i.e. Folds 1, 2 and 3).

Fig. 3. 3-fold Cross Validation Schematic.

Table 1
Model specification dictionary keys and values.

Key	Value type	Value description
`n_hidden_states`	`int`	Expected number of discrete hidden states.
`observations`	`list`	Feature names, types (i.e. discrete, continuous), distributions (if continuous) , and values (if discrete).
`regularization_constant`	`int`	Size of regularization constant.
`model_parameter_constraints`	`dict`	Seed values for transition, initial state, and emission parameters, including GMM parameters.

2.5. *The* `Hela` *library*

A key contribution of this work is the introduction of the open-source `Hela` library for hidden Markov modeling [28]. This library provides a flexible framework for hybrid observation types (i.e. a combination of various continuous and discrete observations). This section will briefly describe the workflow for model initialization, training, and prediction, although the curious reader is directed to the `Hela` documentation for more complete details.

Initialization: Models are initialized via a specification dictionary. Models themselves are data agnostic but require a description of the data features to be modeled. A specification dictionary consists of the keyed values described in Table 1. These values can either be filled in with informed priors (i.e. constraints) or left blank. Using the specification, a model configuration object is generated. This object includes a complete set of model parameters, which are instantiated using constraints where they are given and a random seed where they are not. Using the model configuration, it is now possible to initialize a model object that is available for training and inference tasks. The motivation for this pipeline is reproducibility. From a single configuration object, it is possible to deterministically generate multiple instances of the same untrained model object to tune and validate training routines.

In the case that the number of GMM components is chosen to be too large, the GMM will "collapse" to one dominant component and several mixture components that are just a single point, with zero covariance. This is perfectly acceptable from the data modeling perspective. However, in the case of actually carrying out EM, it should be noted that the built-in probability density function solver in Numpy allows singular covariance matrices by computing pseudo-determinants. This leads to the optimization program drifting around, avoiding convergence. Therefore, some care must be taken in choosing the number of GMM components.

Beyond the need for a non-singular variance matrix, in order for the EM algorithm to converge, a suitable pseudoinverse for Σ_{im} must be computed without significant loss of numerical precision. In order to guarantee this convergence, the model can be adjusted using Tikhonov regularization, which in the matrix case just amounts to adding a small multiple of the identity matrix (a "nugget") to the computed covariance at the update step. Practically speaking, this addition can be interpreted as accounting for variance in the noise of the observations [32]. An alternative and classically studied method would involve performing an eigenvalue deflation and then computing the pseudoinverse using the singular value decomposition. However, this method is known to be susceptible to extreme roundoff errors, and it is not clear that this would yield improved results [33] (see Fig. 4).

Training: A model is trained by first loading an `HMMLearningAlgorithm` class object. This class had all of the methods necessary to train the model using exact EM, or EM with a variant on the maximization step using Gibbs sampling or variational inference. Model training is carried out to a specified number of iterations, and sufficient statistics are stored along the way. After a model is trained, it is possible to carry out the task of predicting hidden states on both training and testing data sets. This is carried out using the Viterbi algorithm [34] (see Fig. 5).

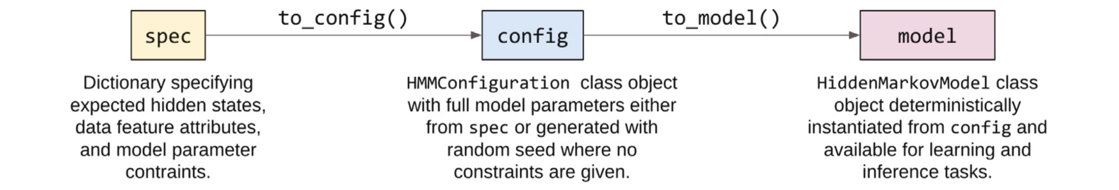

spec
to_config()
config
to_model()
model
Dictionary specifying expected hidden states, data feature attributes, and model parameter contraints.
HMMConfiguration class object with full model parameters either from spec or generated with random seed where no constraints are given.
HiddenMarkovModel class object deterministically instantiated from config and available for learning and inference tasks.

Fig. 4. Multi-state model initialization for replicable, deterministic model seeding.

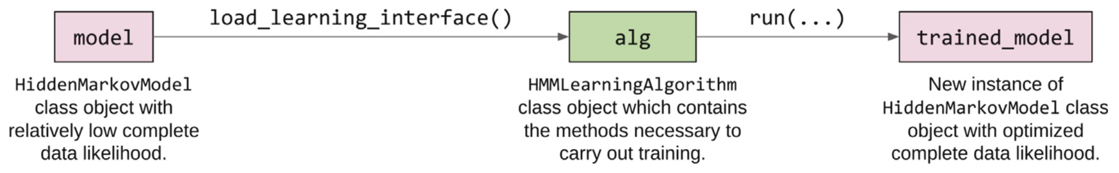

model
load_learning_interface()
alg
run(...)
trained_model
HiddenMarkovModel class object with relatively low complete data likelihood.
HMMLearningAlgorithm class object which contains the methods necessary to carry out training.
New instance of HiddenMarkovModel class object with optimized complete data likelihood.

Fig. 5. Training with multiple rounds of EM guaranteeing convergence to optimal complete data likelihood.

3. Experiments

3.1. Dataset description and preprocessing

The proposed library has been tested and validated using experimental data collected from an offshore wind turbine. The instrumented turbine is a 6 MW monopile turbine located in the North Sea in Europe. The turbine under investigation is located in the external section of the offshore wind farm, exposed to open-sea marine conditions with wind directions on average predominantly from the west during the fall season. The turbine consists of a monopile driven into the seabed and a transition piece allowing access to the wind turbine. The overall height of the turbine from mudline to the nacelle is about 140 m. One system collects dynamic data from 16 strain gauges and 12 accelerometers distributed along the tower and foundation, while the second system is a Supervisory Control and Data Acquisition (SCADA) condition monitoring system located at the top of the turbine, which collects operational and environmental information. Fig. 6 displays the location and distribution of the sensors along the structure. The strain measurements provide 8 bending moment measurement channels, and the SCADA system acquires various operational and environmental information, such as power production, rotor speed, pitch and yaw angles, and wind speed. For this study, only the wind speed information is considered, and the data from both acquisition systems are available at a 25 Hz frequency for all channels.

While high-frequency resolution is essential for capturing the dynamics embedded in the accelerometer and strain gauge response of the wind turbine, it is not needed to predict wind speed since this quantity does not change as fast. Consequently, instead of using the 25 Hz resolution data, a 1-minute average value of all data channels is preferred for the present application.

The observed variables in the following HMM-based framework are represented by the dynamic quantities (moments and accelerations), while the hidden states inferred by the model consist of two different wind speed bins. The model is built using a selection of 13 days of continuous acquisitions, and, in detail, 80% of the data is used for training, and the remaining 20% is used for testing. The dynamic data are normalized with respect to the mean and standard deviation.

The wind speed is separated into two bins considering as cutoff wind speed of 8 m/s. The following cutoff wind speed value is chosen so that the two classes are equally balanced and represented in the dataset. The alternation of the two states can be observed in Fig. 7 for the 13 days of interest.

It was observed that the use of the acceleration data did not contribute to an improvement of the observability of the hidden states, and consequently, it did not lead to better classification performance. The moments derived by the strain sensors are the most significant features for this study case. Therefore, the results presented in the following sections relate to the use of the moments as the only input observed features to the model.

3.2. Model parameters definition

The parameters that the user needs to set in the definition of the methodology using the `Hela` library are the number of states, the transition matrix, the number of GMM components, and the coefficient for the regularization of model parameters (see Table 1). The first two parameters are easy to define since they directly correlate to the number of states to identify in the data and their distribution within the analyzed dataset. The last two parameters highly depend on the features' magnitudes and the observability of the hidden states and can be set by an exploratory investigation of the dataset.

The number of states to identify in the present application is 2, with high and low wind speeds. For the transition matrix, it is possible to either initialize the model with an initial random guess of the matrix or with the matrix directly derived from the

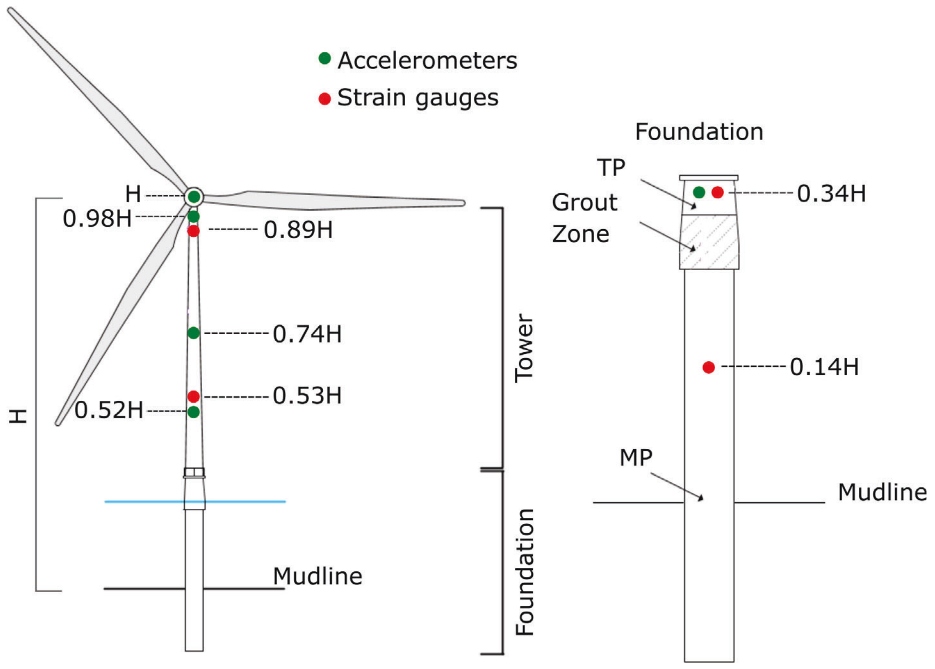
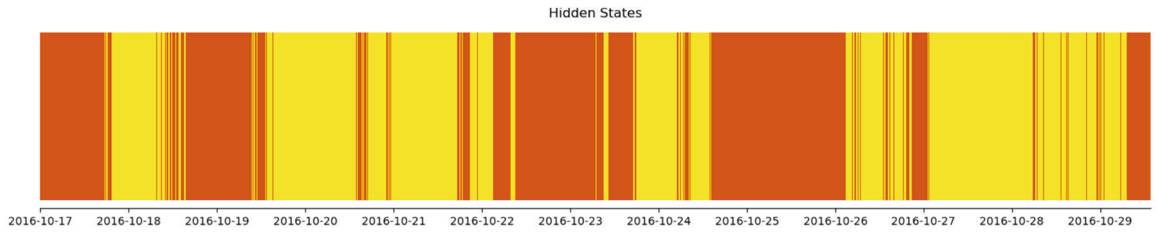
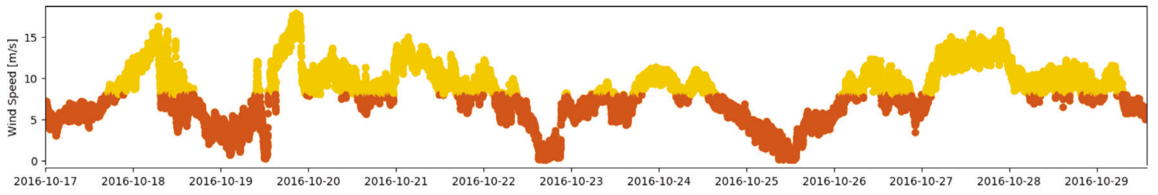


Fig. 6. Monitoring instrumentation.



(a)



(b)

Fig. 7. Wind speed bins (a) and separation of the wind speed values into bins (b).

training dataset. For the following study case, the true transition matrix, computed using the experimental data from the training set, assumes the following expression.

$$A = \begin{bmatrix} 0.931, 0.069 \\ 0.053, 0.947 \end{bmatrix} \tag{41}$$

There are three different options for choosing the number of GMM components. It is possible to have a random initialization of the HMM model without choosing any GMM component. In this case, the number of GMM components is taken to be exactly the number of hidden states, and GMM parameters are taken to be Gaussian mixture model parameters fit to the complete training data. The fit is achieved by implementing scikit-learn's `sklearn.mixture.GaussianMixture.fit()` [35], which provides an estimate of model parameters with the EM algorithm. The method fits the model n times and sets the parameters with which the model has the largest likelihood or lower bound. Within each trial, the method iterates between E-step and M-step for a maximum number of iterations until the change of likelihood or lower bound is less than a tolerance level. In general, this will not be a

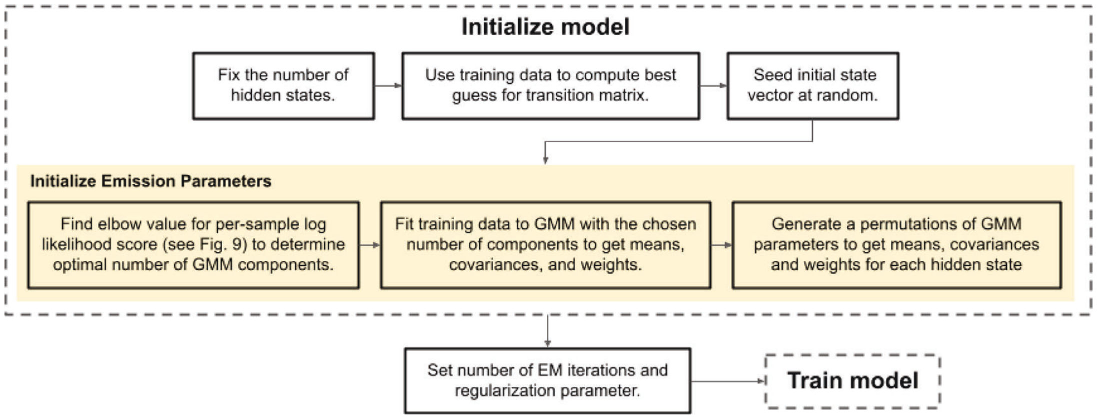

Initialize model
Fix the number of hidden states.
Use training data to compute best guess for transition matrix.
Seed initial state vector at random.
Initialize Emission Parameters
Find elbow value for per-sample log likelihood score (see Fig. 9) to determine optimal number of GMM components.
Fit training data to GMM with the chosen number of components to get means, covariances, and weights.
Generate a permutations of GMM parameters to get means, covariances and weights for each hidden state
Set number of EM iterations and regularization parameter.
Train model

Fig. 8. First the model is initialized on training data, the training parameters are set, and the model is fit to the training data.

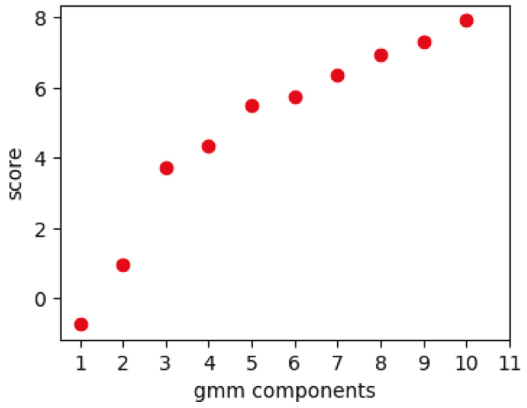


Fig. 9. Per sample log-likelihood of the observed data by number of GMM components.

particularly good initialization and will typically lead to a locally optimal convergence, which can be far from the global optimum. A schematic of the model initialization is provided in Fig. 8.

Instead of relying on a random selection of the GMM components, it is possible to select and input this parameter in two different ways. If the user knows a priori the preferred number of components, this can be directly specified in the specification function definition. An alternative solution is to choose the number of GMM components by looking at likelihood-related information. A popular method commonly used is to look for the minimum in the Akaike information criterion (AIC) and Schwarz's Bayesian Information Criterion (BIC) [36,37]. These are both penalized-likelihood information criteria. AIC is an estimate of a constant plus the relative distance between the unknown true likelihood function of the data and the fitted likelihood function of the model. In contrast, BIC is an estimate of a function of the posterior probability of a model being true under a certain Bayesian setup. An alternative option is to rely on the per-sample log-likelihood score of the observed data provided by the Gaussian Mixture model distribution for different possible options of GMM components. The two strategies lead to substantially similar results, but in the present methodology, the latter is preferred since it tends to be less conservative. The derivation of the log-likelihood is done using scikit-learns `sklearn.mixture.GaussianMixture().score` function. Fig. 9 shows the value of the score obtained testing up to 10 components. The optimal choice for the number of GMM components lies around the elbow. Given the randomness of the Gaussian Mixture Model, the optimal value might be before or after that change is slope in the score function. In the present application, the optimal value lies around 2 and 4. The three possible values (number of components equal to 2,3 and 4) were all considered, but there was no substantial improvement in the final results. Therefore, to reduce the computational effort of the model, the number of GMM components was chosen to be equal to 2.

Once the number of GMM components is set, the means, covariance matrices, and weights used to initialize the models are obtained as estimated parameters of a Gaussian mixture distribution. It is fundamental to point out that in the case of a single GMM component, a small amount of noise or variation needs to be introduced for the means of the different states in order to disambiguate the clusters in advance of learning. It is common in any clustering algorithm that the initial "centroids" need to have at least a small amount of distance between them. In the present formulation and application, the variation is introduced by changing the sign of the mean value derived by the Gaussian Mixture model algorithm.

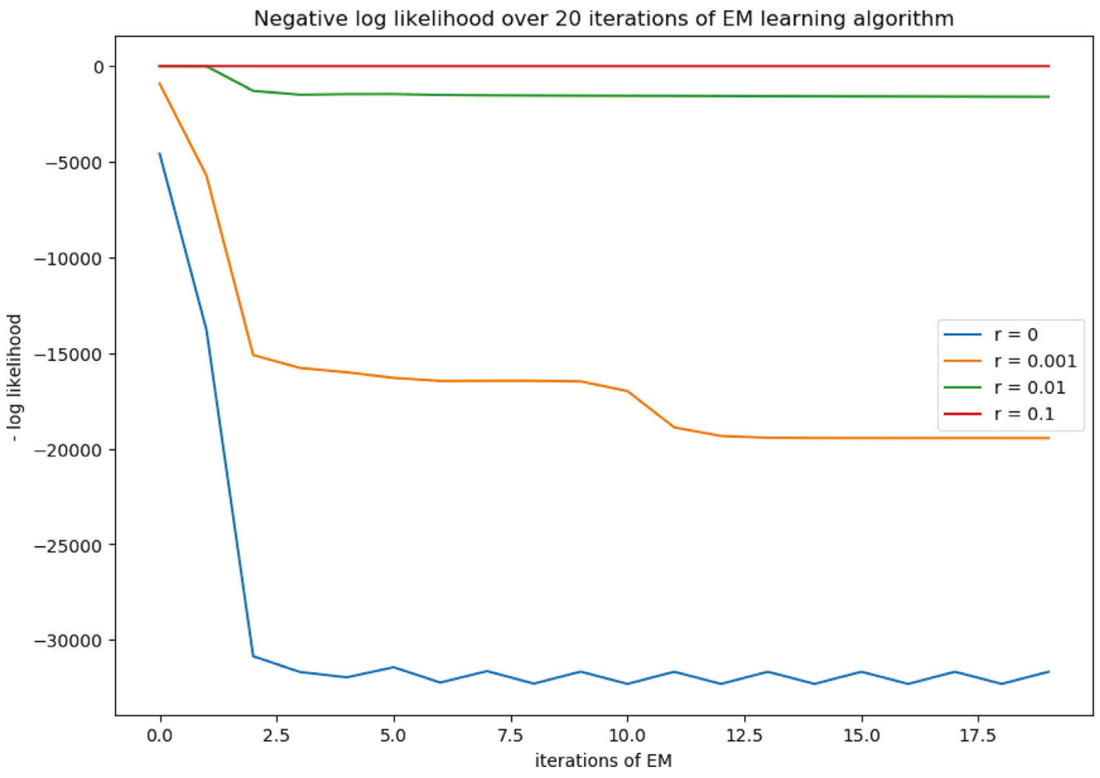


Fig. 10. Log-likelihood functions for different values of the regularization term.

As previously mentioned in Section 2.5, the regularization coefficient plays a key role in guaranteeing convergence for the EM algorithm. Fig. 10 shows the negative log-likelihood function for the different iterations of the EM algorithm considering four different regularization coefficients: 0, 0.1, 0.01, 0.001. It is evident that when the coefficient corresponds to 0, which implies that no Tikhonov regularization is implemented, the convergence of the likelihood function is compromised, and instead of setting on a stable value, it assumes alternating values. On the other hand, the introduction of a small regularization coefficient removes this unstable behavior, leading to a clearer convergence.

The positive effect of the regularization can be observed in the performance accuracy of the HMM model. When no regularization term is considered ($r = 0$ in Fig. 10), only a 77.21% accuracy is reached for the training dataset; the model also shows a poor generalization capability for the testing set where the classification accuracy goes down to 64.52%. On the other hand, the introduction of a small regularization term equal to 0.001 or 0.01 causes a substantial increase in the model performance, with accuracies respectively equal to 84.25% and 83.44% for the training dataset and 77.31% and 76.71% for the testing dataset. The coefficient needs to be properly tuned since a higher regularization coefficient ($r = 0.1$) could lead, like in this case, to moderately worse results in comparison with those obtained for smaller regularization terms (80.63% accuracy for the training and 69.24% accuracy for the testing). Despite this, including regularization terms still resulted in better outcomes compared to not using them. However, setting the regularization Tykhonov constant too high can lead to underfitting and increased bias. This happens because it can excessively shrink the model coefficients, causing the model to be less responsive to changes in the data. As a result, the model may become too simplistic and not accurately fit the data.

For the present application, the regularization term is chosen equal to 0.001.

3.3. Results with or without the initialization by cross-validation

In the first step of the study, an HMM is trained on the training dataset considering the one GMM component and regularization constant of 0.001. Fig. 10 shows the negative log-likelihood function and shows the convergence of the EM algorithm in 20 iterations. The accuracy reached on the training dataset is 84.25% and 77.31% on the testing set. The model has a good performance on the training dataset and seems to generalize fairly well on the testing.

In order to improve the generalization of the model further, the cross-validation strategy presented in Section 2.4 is implemented for the initialization of the model. The dataset used for the training is split into a number of folds, and independent HMM models are trained using the single folds, adopting the previously mentioned strategy. For each fold, the number of GMM components and the regularization term are kept the same, while the transition matrix is computed on the specific subset of the training set. Then,

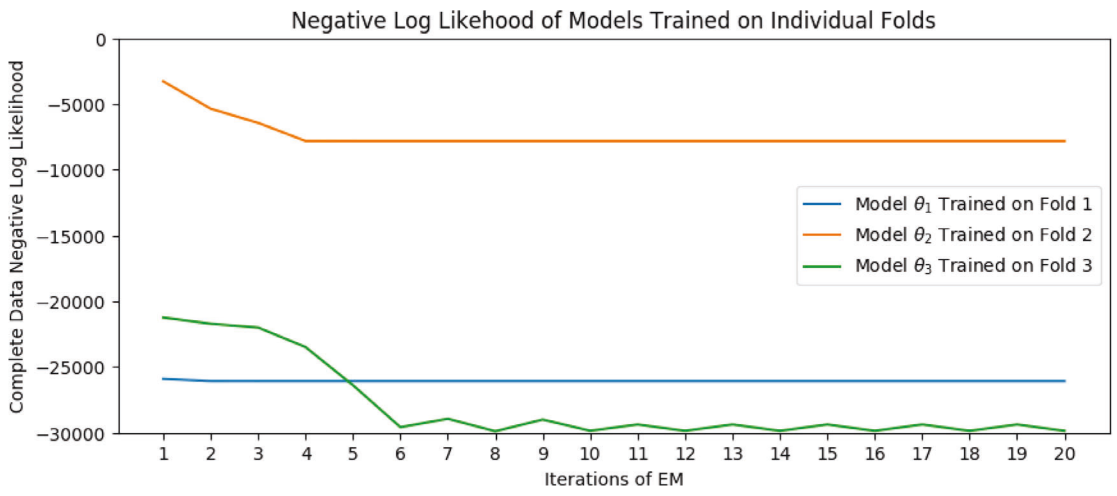


Fig. 11. Log-likelihood function for the three subset models θ_1, θ_2, θ_3 (regularization term 0.001).

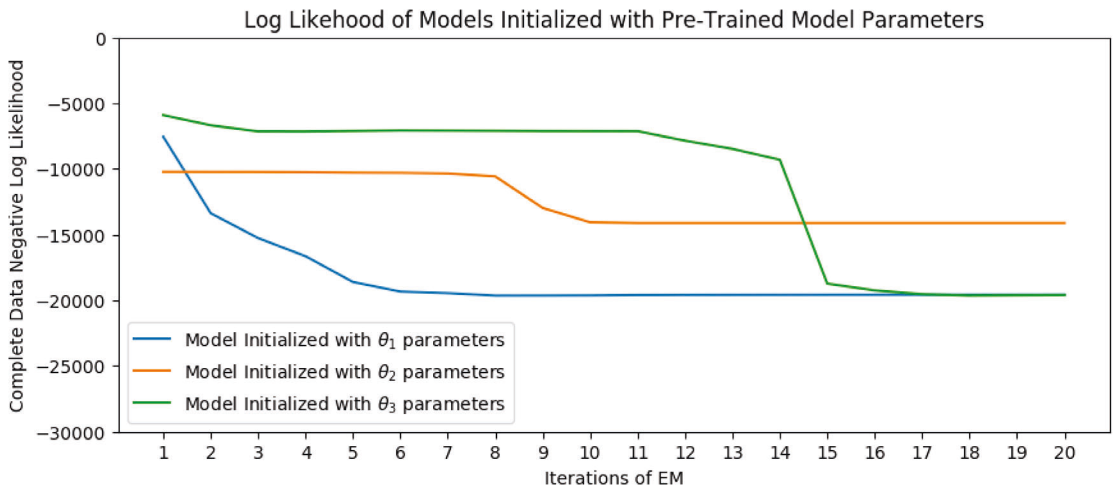


Fig. 12. Log-likelihood function for the final model initializing with the parameters from the three models (regularization term 0.001).

the model with the better performance and that maximizes the log-likelihood is used to seed and initialize the training of the final model. The initialization is performed by using the means, covariance matrices, weights, and transition matrix obtained at the end of the recursive updating training of the most performative model. These quantities are used as initial guesses for the model, bypassing their derivation from the Gaussian Mixture model distribution.

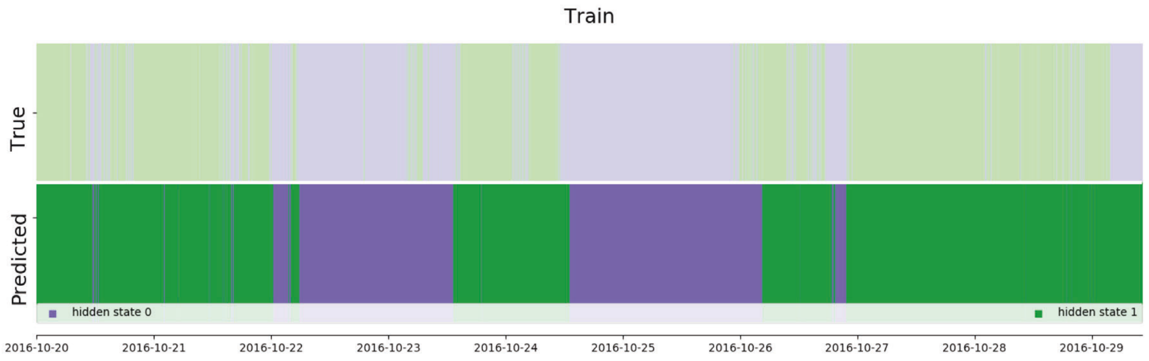
In the present study case, given the data amount and distribution, the training set is split into three folds. There is no data overlap between the folds and no influence between the different models. The performance of each independent model is later validated on the testing dataset, which consists of unseen observations. The log-likelihood function for the three subset models is presented in Fig. 11 while Fig. 12 shows the log-likelihood function obtained by the initialization of the entire model using the model parameters derived from the first fold-based model. The performance of the single models and the log-likelihood values are presented in Table 2.

Looking at Fig. 11, it is possible to compare the learning experience in the training of the three models. The log-likelihood function for the first model suggests a better and faster learning experience. This is also reflected in the training and testing accuracy values presented in Table 2 that are highest among the three, and it is also reflected in the training log-likelihood that is properly maximized. The remaining two models present, on the one hand (model 2), a very insignificant learning experience or a convergence affected by precision errors that arise from computing pseudo-inverses (model 3) not solved by the regularization term.

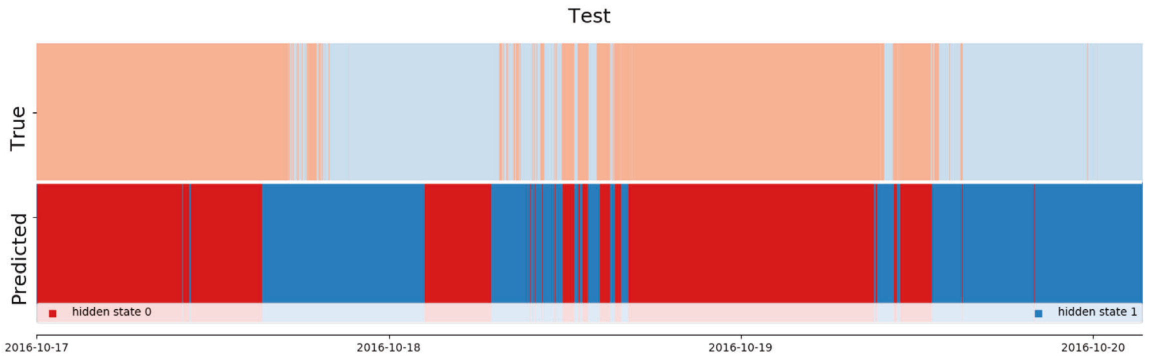
For this application, model 1 is used to initialize the training on the entire training dataset using the means, covariances, weights, and transition matrix obtained at the end of the EM iterations for model 1. In Fig. 12, it can be noticed how the convergence of the model initialized with θ_1 parameters is smoother and faster with respect to the one obtained by training the entire model without initialization (Fig. 10) or training it on the remaining two subsets (Fig. 12). The log-likelihood functions for the model initialized

Table 2
Subset models results in terms of log-likelihood and performance accuracy for the training and testing datasets.

Subset model	Log-Likelihood		Accuracy	
	Training	Testing	Training	Testing
1	250.04	−26068.86	81.08%	74.79%
2	3.95	−7810.89	53.05%	51.08%
3	77.40	−29851.40	50.75%	45.58%



(a)



(b)

Fig. 13. True and predicted hidden states for the training dataset (a) and for the testing dataset (b) using the proposed Hidden Markov Model strategy.

using the remaining two folds both show a static initial branch of the function or a “pre asymptotic region” (up to the 8th and 11th iteration respectively). This is commonly due to a different order of magnitude of the updated parameters; the mean values might be moving in different magnitude steps with respect to the covariance terms. This option in a variable step size is allowed by the use of the lagrangian multipliers in the constrained optimization problem, which removes the assumption on the classical gradient descent of having a fixed step size.

Fig. 13 presents the true hidden states and the corresponding prediction of the optimized model for the training and testing sets. While the overall performance of the model stays the same for the training set (85.43%), there is a substantial improvement in the prediction of the testing set (84.17%), moving it closer to the ability prediction shown in the training set. The model is able to properly differentiate between the low and high wind states in the majority of the cases, also for frequent and close variations.

The trained HMM model performance is compared with the classification ability of two established and commonly used unsupervised classification algorithms, k-Means [38] and Gaussian Mixture Model [39]. k-Means tries to partition the dataset into k pre-defined distinct non-overlapping clusters where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different and distant as possible. The Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mix of Gaussian distributions with unknown parameters. It is a probabilistic model where Gaussian distributions are assumed for each cluster, with means and covariances defining their parameters.

For k-Means, the number of clusters was set equal to two, and for Gaussian Mixture Models, the number of components was defined as equal to two. The k-Means algorithm performs poorly, with an accuracy of 47.26% and 69.33% for the training and testing sets, respectively. The discrepancy between the training and testing dataset also highlights a strong underfitting tendency of

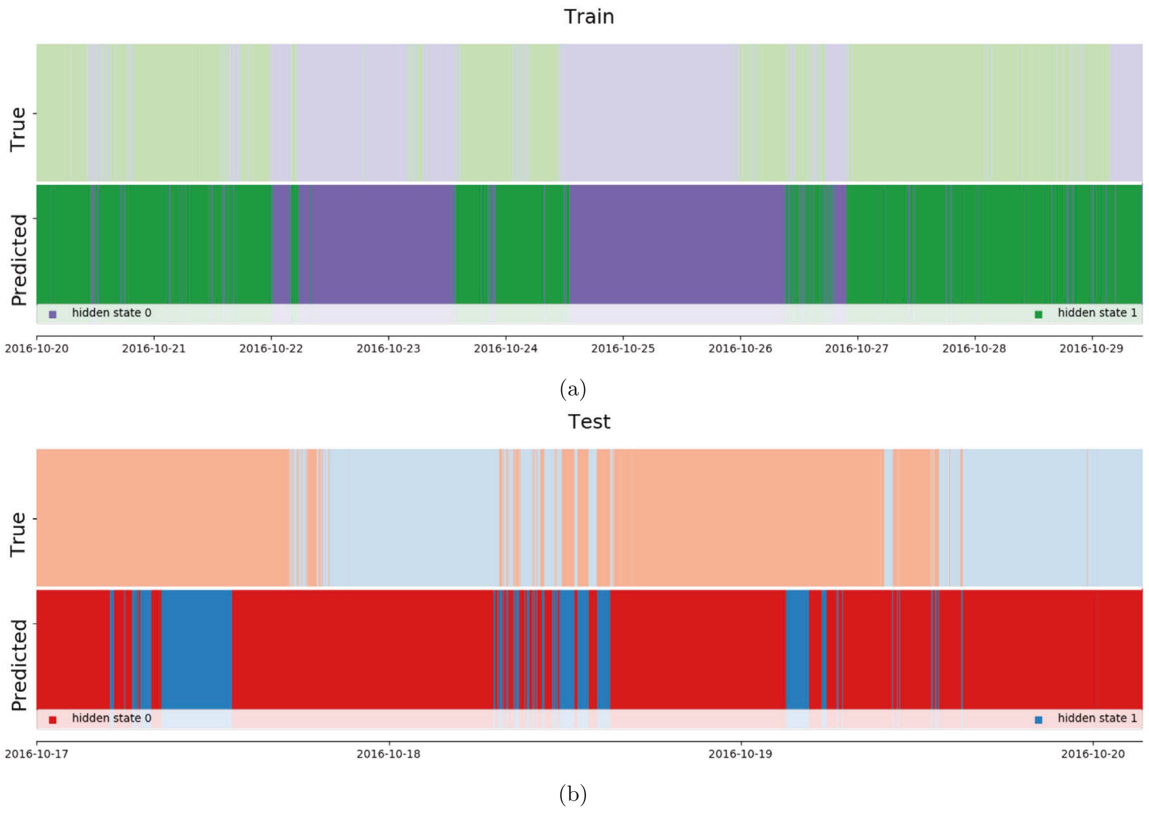


Fig. 14. True and predicted hidden states for the training dataset (a) and for the testing dataset (b) using Gaussian Mixture Model.

the *k*-Means algorithm. On the other hand, the Gaussian Mixture Model provides results that are closer to those obtained with HMM. The Gaussian Mixture Model performs better on the training dataset with a classification score of 82.28%. However, the model does not generalize as well on the testing set, where it reaches an accuracy of 43.48%, showing a dominant overfitting behavior. The graphical representation of the results for the Gaussian Mixture model is presented in Fig. 14.

The comparison between the different models' results shows how Hidden Markov Models have several advantages over Gaussian Mixture Models and *k*-Means in certain applications. Firstly, both Gaussian Mixture Models and *k*-Means fail to properly identify the two classes within the data distribution, which, in this study case, happen to show a concentric structure. These two models are unable to separate the clusters properly, given this cluster distribution. On the other hand, HMMs succeed in the task of adding additional dimensions through the covariance distribution and the temporal dependency. The latter is one of the biggest strengths of HMMs, which can model temporal dependencies between observations, something not possible with GMMs and *k*-Means. This makes HMMs particularly useful for the present application, which involves modeling time series data where the order of observations matters. Then, HMMs can model unobserved states, which is not possible with GMMs and *k*-Means. This makes HMMs useful for applications where there are hidden variables that affect the observations. HMMs can model flexible output distributions, which is not possible with *k*-Means. This makes HMMs useful for applications where the output distribution could be a mixture of Gaussians. HMMs are parameter-efficient compared to GMMs. Since HMMs model temporal dependencies, they can use fewer parameters than GMMs to model the same data. This makes HMMs more efficient for applications with large amounts of data.

4. Conclusion

The present work describes a regularized Hidden Markov Model strategy with informed initialization to predict wind speed in an experimental application of an offshore wind turbine.

Hidden Markov Models are unsupervised stochastic models that can capture hidden temporal dependencies between observations, making them useful for time series data modeling in various engineering applications. However, the implementation of HMMs in real experimental studies may result in unstable model convergence.

The proposed strategy introduces an updated version of the Hidden Markov Model algorithm that uses the Tykhonov regularization technique to overcome the possible loss of precision and the erratic convergence behavior in the learning model due to the presence of an ill-conditioned matrix in the EM stage. Additionally, to avoid random model initialization, the proposed strategy introduces an informed cross-validation method to initialize the model training using data-tailored parameters. The proposed methodology is implemented using the Python open-access library `Hela`, which can be utilized for various applications.

This paper presents the results of applying the `Hela` library to train a model that predicts wind speed information for an offshore wind turbine. The Tykhonov regularization technique improved the convergence of the model, eliminating unstable and erratic behavior during the learning phase due to ill-conditioned information. Furthermore, the informed cross-validation method produced more robust and accurate predictions of wind speed information than random initialization. HMMs outperformed two commonly used unsupervised methodologies, k-Means and Gaussian Mixture Models, and led to substantially better results.

CRediT authorship contribution statement

Anna Haensch: Conceptualization, Formal analysis, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing. **Eleonora M. Tronci:** Formal analysis, Validation, Visualization, Writing – original draft, Writing – review & editing. **Bridget Moynihan:** Formal analysis, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Babak Moaveni:** Funding acquisition, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

While the wind turbine data is protected and cannot be shared, the HMM code is available on GitHub.

Acknowledgments

The authors acknowledge partial support of this study by the National Science Foundation grant 2230630. The team is grateful to Dr. Finn Rüdinger and Dr. Ross McAdam at Orsted for their help in accessing and processing of the experimental data. The team also extends thanks to Dr. Will Vega-Brown and Isabel Ljungberg at Tagup, Inc. for contributions to the `Hela` library. Opinions, findings, and conclusions expressed in this paper are those of the authors and do not necessarily represent the views of the sponsors and organizations involved in this project.

Appendix A

A.1. Derivation of variance update equation

Computing the derivative of the argument in Eq. (34) with respect to the Σ_{mi} and solving for 0, yields

$$0 = \sum_{t=1}^{T} \frac{w_{mi} \cdot \frac{\partial}{\partial \Sigma_{mi}} \left[\mathcal{N}(x_t; \mu_{mi}, \Sigma_{mi}) \right] \cdot \langle Z_t \rangle_i}{\sum_{m'=1}^{M} W_{im'} \cdot \mathcal{N}(x_t; \mu_{im'}, \Sigma_{im'})} \tag{42}$$

for all $1 \leq i \leq N$ and $1 \leq m \leq M$. Consider

$$\frac{\partial \Sigma_{mi}^{-1}}{\partial \Sigma_{mi}} \tag{43}$$

as a matrix of partial derivatives with respect to the entries of Σ_{mi}, where the jk^{th} entry is given by

$$\left(\frac{\partial \Sigma_{mi}^{-1}}{\partial \Sigma_{mi}} \right)_{jk} = \frac{\partial \Sigma_{mi}^{-1}}{\partial \left(\Sigma_{mi} \right)_{jk}}. \tag{44}$$

Since

$$\Sigma_{mi} \cdot \Sigma_{mi}^{-1} = \mathbb{I}, \tag{45}$$

where $\mathbb{I}$ is the identity matrix, it follows from the product rule that

$$\frac{\partial \Sigma_{mi}}{\partial \left(\Sigma_{mi} \right)_{jk}} \cdot \Sigma_{mi}^{-1} + \Sigma_{mi} \cdot \frac{\partial \Sigma_{mi}^{-1}}{\partial \left(\Sigma_{mi} \right)_{jk}} = 0 \tag{46}$$

and therefore

$$\frac{\partial \Sigma_{mi}^{-1}}{\partial \left(\Sigma_{mi} \right)_{jk}} = -\Sigma_{mi}^{-1} \cdot e_j^{\intercal} \cdot e_k \cdot \Sigma_{mi}^{-1} \tag{47}$$

where e_j and e_k are elementary row vector (i.e. e_j is equal to 0 in all but the jth component, at which it is equal to 1). Using Eq. (47), compute the partial derivative

$$\begin{aligned}&\frac{\partial}{\partial\left(\Sigma_{mi}\right)_{jk}}\left[(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\right]\\&=-(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\cdot e_j^\intercal\cdot e_k\cdot\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\\&=-\left(e_k\cdot\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\right)\cdot\left((x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\cdot e_j^\intercal\right)\\&=-e_k\cdot\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\cdot e_j^\intercal,\end{aligned}\tag{48}$$

where the penultimate step works by splitting the expression into two scalars which necessarily commute. From here, it follows that the full matrix of partial derivatives with respect to Σ_{mi} is

$$\frac{\partial}{\partial\Sigma_{mi}}\left[(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\right]=-\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}.\tag{49}$$

Before fully computing the partial derivative in Eq. (42) one additional fact from matrix algebra will be necessary, namely

$$\frac{\partial\left|\Sigma_{mi}\right|}{\partial\Sigma_{mi}}=\left|\Sigma_{mi}\right|\cdot\Sigma_{mi}^{-1}\tag{50}$$

which is a consequence of Jacobi's formula, and from which it can be deduced

$$\frac{\partial}{\partial\Sigma_{mi}}\left[\left|\Sigma_{mi}\right|^{-\frac{1}{2}}\right]=-\frac{1}{2}\left|\Sigma_{mi}\right|^{-\frac{3}{2}}\cdot\left|\Sigma_{mi}\right|\cdot\Sigma_{mi}^{-1}=-\frac{1}{2}\left|\Sigma_{mi}\right|^{-\frac{1}{2}}\cdot\Sigma_{mi}^{-1}.\tag{51}$$

Now the partial derivative in the numerator of Eq. (42), can be easily computed using the product rule and the equations above to obtain

$$\begin{aligned}&\frac{\partial}{\partial\Sigma_{mi}}\left[(2\pi)^{-\frac{K}{2}}\cdot\left|\Sigma_{mi}\right|^{-\frac{1}{2}}\cdot\exp\left\{-\frac{1}{2}(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\cdot(x_t-\mu_{mi})\right\}\right]\\&=-\frac{1}{2}\cdot\mathcal{N}(x_t;\mu_{mi},\Sigma_{mi})\cdot\Sigma_{mi}^{-1}\cdot\left(1-(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\right).\end{aligned}\tag{52}$$

Substituting this into Eq. (42) and dividing out the unnecessary terms to get

$$\begin{aligned}0&=\sum_{t=1}^{T}\frac{w_{mi}\cdot\mathcal{N}(x_t;\mu_{mi},\Sigma_{mi})\cdot\left(1-(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\right)\cdot\langle Z_t\rangle_i}{\sum_{m'=1}^{M}W_{im'}\cdot\mathcal{N}(x_t;\mu_{im'},\Sigma_{im'})}\\&=\sum_{t=1}^{T}\langle Z_tX_t\rangle_{mi}\cdot\left(1-(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal\cdot\Sigma_{mi}^{-1}\right)\end{aligned}\tag{53}$$

which implies

$$\sum_{t=1}^{T}\langle Z_tX_t\rangle_{mi}\cdot\Sigma_{mi}=\sum_{t=1}^{T}\langle Z_tX_t\rangle_{mi}\cdot(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal.\tag{54}$$

This can now be solved to obtain the optimal covariance parameter

$$\Sigma_{mi}^{*}=\frac{\sum_{t=1}^{T}\langle Z_tX_t\rangle_{mi}\cdot(x_t-\mu_{mi})\cdot(x_t-\mu_{mi})^\intercal}{\sum_{t=1}^{T}\langle Z_tX_t\rangle_{mi}}\tag{55}$$

for all $1\le i\le N$ and $1\le m\le M$.

References

[1] Walt Musial, Paul Spitsen, Patrick Duffy, Philipp Beiter, Melinda Marquis, Rob Hammond, Matt Shields, Offshore Wind Market Report: 2022 Edition, Technical report, Department of Energy, 2022.

[2] Vestas Wind Systems, V236-15.0 MW™ at a glance. Vestas. https://www.vestas.com/en/products/offshore/V236-15MW.

[3] T. Rubert, G. Zorzi, G. Fusiek, P. Niewczas, D. McMillan, J. McAlorum, M. Perry, Wind turbine lifetime extension decision-making based on structural health monitoring, Renew. Energy 143 (2019) 611–621.

[4] J.H. Piel, C. Stetter, M. Heumann, M. Westbomke, M.H. Breitner, Lifetime extension, repowering or decommissioning? decision support for operators of ageing wind turbines, J. Phys. Conf. Ser. 1222 (1) (2019) 012033.

[5] T. Rubert, D. McMillan, P. Niewczas, A decision support tool to assist with lifetime extension of wind turbines, Renew. Energy 120 (2018) 423–433.

[6] A. Kazemi Amiri, R. Kazacoks, D. McMillan, J. Feuchtwang, W. Leithead, Farm-wide assessment of wind turbine lifetime extension using detailed tower model and actual operational history, J. Phys. Conf. Ser. 1222 (1) (2019) 012034.

[7] M.L. Hossain, A. Abu-Siada, S.M. Muyeen, Methods for advanced wind turbine condition monitoring and early diagnosis: A literature review, Energies 11 (5) (2018) 1309.

[8] Alejandro Moreno-Gomez, Carlos A. Perez-Ramirez, Aurelio Dominguez-Gonzalez, Martin Valtierra-Rodriguez, Omar Chavez-Alegria, Juan P. Amezquita-Sanchez, Sensors used in structural health monitoring, Arch. Comput. Methods Eng. 25 (2018) 901–918.

[9] Yuequan Bao, Zhicheng Chen, Shiyin Wei, Yang Xu, Zhiyi Tang, Hui Li, The state of the art of data science and engineering in structural health monitoring, Engineering 5 (2) (2019) 234–242.

[10] Eric M. Hines, Christopher D.P. Baxter, David Ciochetto, Mingming Song, Per Sparrevik, Henrik J. Meland, James M. Strout, Aaron Bradshaw, Sau-Lon Hu, Jorge R. Basurto, Babak Moaveni, Structural instrumentation and monitoring of the block island offshore wind farm, Renew. Energy 202 (2023) 1032–1045.
[11] R. Rolfes, S. Tsiapoki, M.W. Häckell, 19 - Sensing solutions for assessing and monitoring wind turbines, in: M.L. Wang, J.P. Lynch, H. Sohn (Eds.), Sensor Technologies for Civil Infrastructures, in: Woodhead Publishing Series in Electronic and Optical Materials, vol. 56, Woodhead Publishing, 2014, pp. 565–604.
[12] Maria Martinez-Luengo, Athanasios Kolios, Lin Wang, Structural health monitoring of offshore wind turbines: A review through the statistical pattern recognition paradigm, Renew. Sustain. Energy Rev. 64 (2016) 91–105.
[13] Mathew L. Wymore, Jeremy E. Van Dam, Halil Ceylan, Daji Qiao, A survey of health monitoring systems for wind turbines, Renew. Sustain. Energy Rev. 52 (2015) 976–990.
[14] Christof Devriendt, Filipe Magalhães, Wout Weijtjens, Gert De Sitter, Álvaro Cunha, Patrick Guillaume, Structural health monitoring of offshore wind turbines using automated operational modal analysis, Struct. Health Monit. 13 (6) (2014) 644–659.
[15] Zi Lin, Xiaolei Liu, Wind power forecasting of an offshore wind turbine based on high-frequency SCADA data and deep learning neural network, Energy 201 (2020) 117693.
[16] Mehdi Neshat, Meysam Majidi Nezhad, Ehsan Abbasnejad, Seyedali Mirjalili, Lina Bertling Tjernberg, Davide Astiaso Garcia, Bradley Alexander, Markus Wagner, A deep learning-based evolutionary model for short-term wind speed forecasting: A case study of the Lillgrund offshore wind farm, Energy Convers. Manage. 236 (2021) 114002.
[17] Xiaolei Liu, Zi Lin, Ziming Feng, Short-term offshore wind speed forecast by seasonal ARIMA - A comparison against GRU and LSTM, Energy 227 (2021) 120492.
[18] P. Flores, A. Tapia, G. Tapia, Application of a control algorithm for wind speed prediction and active power generation, Renew. Energy 30 (4) (2005) 523–536.
[19] Torben Knudsen, Thomas Bak, Mohsen Soltani, Prediction models for wind speed at turbine locations in a wind farm, Wind Energy 14 (7) (2011) 877–894.
[20] Larry P. Heck, James H. McClellan, Mechanical system monitoring using hidden Markov models, in: [Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing, IEEE, 1991, pp. 1697–1700.
[21] Hasan Ocak, Kenneth A. Loparo, Fred M. Discenzo, Online tracking of bearing wear using wavelet packet decomposition and probabilistic modeling: A method for bearing prognostics, J. Sound Vib. 302 (4–5) (2007) 951–961.
[22] Syed Sajjad H. Zaidi, Wesley G. Zanardelli, Selin Aviyente, Elias G. Strangas, Prognosis of electrical faults in permanent magnet AC machines using the hidden Markov model, in: IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society, IEEE, 2010, pp. 2634–2640.
[23] Carey Bunks, Dan McCarthy, Tarik Al-Ani, Condition-based maintenance of machines using hidden Markov models, Mech. Syst. Signal Process. 14 (4) (2000) 597–612.
[24] Sung-Hwan Shin, SangRyul Kim, Yun-Ho Seo, Development of a fault monitoring technique for wind turbines using a hidden Markov model, Sensors 18 (6) (2018).
[25] Jie Ying, T. Kirubarajan K.R., Pattipati, A. Patterson-Hine, A hidden Markov model-based algorithm for fault diagnosis with partial and imperfect tests, IEEE Trans. Syst., Man, Cybern., C (Appl. Rev.) 30 (4) (2000) 463–473.
[26] Debarati Bhaumik, Daan Crommelin, Stella Kapodistria, Bert Zwart, Hidden Markov models for wind farm power output, IEEE Trans. Sustain. Energy PP (2018) 1.
[27] Menglin Li, Ming Yang, Yixiao Yu, Wei-Jen Lee, A wind speed correction method based on modified hidden Markov model for enhancing wind power forecast, IEEE Trans. Ind. Appl. 58 (1) (2022) 656–666.
[28] Anna Haensch, Hela HMM toolkit - beta, 2022, https://github.com/annahaensch/hela.
[29] Todd K. Moon, The expectation-maximization algorithm, IEEE Signal Process. Mag. 13 (6) (1996) 47–60.
[30] Lawrence R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[31] Zoubin Ghahramani, Michael Jordan, Factorial hidden Markov models, Adv. Neural Inf. Process. Syst. 8 (1995).
[32] Hossein Mohammadi, Rodolphe Le Riche, Nicolas Durrande, Eric Touboul, Xavier Bay, An analytic comparison of regularization methods for Gaussian processes, 2016, arXiv preprint arXiv:1602.00853.
[33] Stanley C. Eisenstat, Ilse C.F. Ipsen, Relative perturbation techniques for singular value problems, SIAM J. Numer. Anal. 32 (6) (1995) 1972–1988.
[34] Andrew Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Trans. Inform. Theory 13 (2) (1967) 260–269.
[35] 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, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[36] Shuhua Hu, Akaike information criterion, Cent. Res. Sci. Comput. 93 (2007) 42.
[37] Harish S. Bhat, Nitesh Kumar, On the Derivation of the Bayesian Information Criterion, Vol. 99, School of Natural Sciences, University of California, 2010, Citeseer.
[38] Stuart Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (2) (1982) 129–137.
[39] Douglas A. Reynolds, et al., Gaussian mixture models, Encycl. Biometr. 741 (659–663) (2009).