Expert Systems With Applications 184 (2021) 115558

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Feature selection in jump models

Peter Nystrup [a,b,*], Petter N. Kolm [c], Erik Lindström [a]

[a] Centre for Mathematical Sciences, Lund University, 221 00 Lund, Sweden
[b] Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[c] Courant Institute of Mathematical Sciences, New York University, 10012, NY, USA

ARTICLE INFO

Keywords:
High-dimensional
Sequential data
Time series
Clustering
Unsupervised learning
Regime switching

ABSTRACT

Jump models switch infrequently between states to fit a sequence of data while taking the ordering of the data into account We propose a new framework for joint feature selection, parameter and state-sequence estimation in jump models. Feature selection is necessary in high-dimensional settings where the number of features is large compared to the number of observations and the underlying states differ only with respect to a subset of the features. We develop and implement a coordinate descent algorithm that alternates between selecting the features and estimating the model parameters and state sequence, which scales to large data sets with large numbers of (noisy) features. We demonstrate the usefulness of the proposed framework by comparing it with a number of other methods on both simulated and real data in the form of financial returns, protein sequences, and text. By leveraging information embedded in the ordering of the data, the resulting sparse jump model outperforms all other methods considered and is remarkably robust to noise.

1. Introduction

Jump models are a general class of latent variable models useful for describing systems characterized by abrupt, persistent changes of state dynamics. In such systems, relevant information is contained both in data values *and* in their sequential order. The jump models we consider in this article generalize the hidden Markov model (HMM) by extending classical, order-independent clustering approaches. Unlike classical clustering, by incorporating the ordering of the observations, jump models are able to detect changes in the type of model state the data were produced in an unsupervised fashion (Bemporad et al., 2018).

By way of illustration, Fig. 1 depicts the states resulting from clustering a sequence of 100 samples from a bivariate Gaussian distribution using K-means and a jump model, respectively. A mean shift of the first feature after the first 50 samples defines two distinct states. Because the two conditional distributions overlap, it is impossible to recover the true state sequence *without* taking the ordering of the observations into account. In particular, as illustrated in the figure, K-means has two issues. First, it separates the two states based on both features, although only the first feature is relevant for this purpose. Second, it is unable to recover the overlapping clusters because of its linear decision boundary. While the first issue can be addressed through feature selection, the second issue requires that the sequential ordering of the observations is taken into account. In contrast, the jump model recovers the true, persistent state sequence, despite the overlapping conditional distributions and one irrelevant feature.

State estimation can often be improved through careful selection of exogenous variables or features. In machine learning applications with vast amounts of noisy data, it has become standard practice to reduce the number of input variables by determining the most relevant features. Feature selection works well in many statistical methods, including clustering, classification, and regression models. It is perhaps most common in supervised learning, where labeled training data is available to determine the distinguishing features. Yet, it is equally important in unsupervised applications, such as jump models, where the underlying states are unknown and might not even exist (Dy & Brodley, 2004).

In this article, we present a feature selection framework for jump models inspired by Witten and Tibshirani (2010). They proposed a feature selection approach for clustering based on alternating between selecting feature weights and clustering a weighted version of the data. We extend their approach to jump models. By replacing standard, order-independent clustering with a jump model, we obtain an efficient and powerful approach to clustering high-dimensional sequential data while taking its ordering into account.

We make three main contributions. First, to the best of our knowledge, this is the first article to propose a general framework for joint feature selection, parameter and state-sequence estimation in jump

* Correspondence to: Centre for Mathematical Sciences, Lund University, Box 118, 221 00 Lund, Sweden.
E-mail addresses: peter.nystrup@matstat.lu.se (P. Nystrup), petter.kolm@nyu.edu (P.N. Kolm), erik.lindstrom@matstat.lu.se (E. Lindström).

https://doi.org/10.1016/j.eswa.2021.115558
Received 24 August 2020; Received in revised form 5 June 2021; Accepted 2 July 2021
Available online 13 July 2021

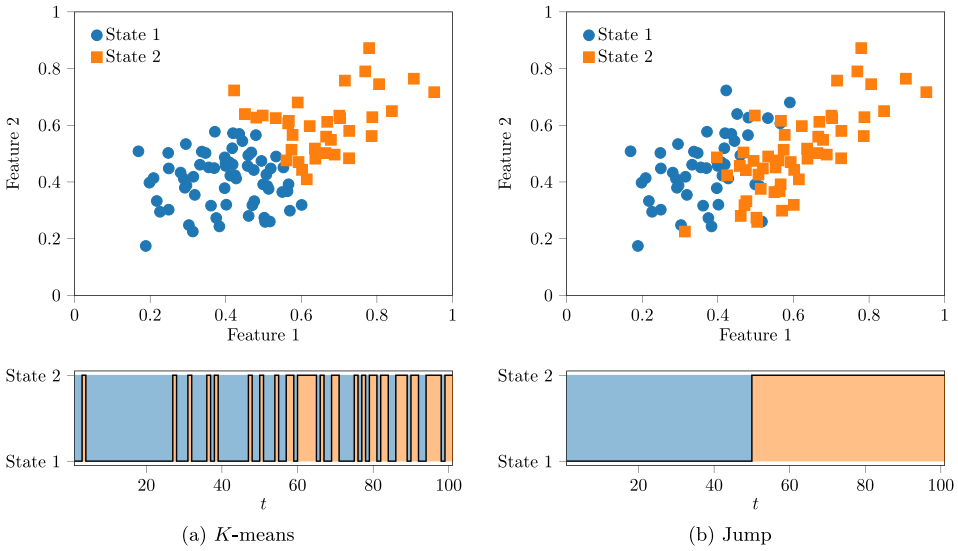

Fig. 1. Illustration of the states resulting from clustering a sequence of $t = 1, \dots, 100$ observations of two features using K-means and a jump model. By taking the ordering of the observations into account, the jump model is able to recover the true state sequence.

models. With our framework it is possible to determine which features characterize the states. Separately, jump models and feature selection are not new ideas; our contribution is to bring them together.

Second, we develop and implement a coordinate descent algorithm that alternates between selecting features and estimating the model parameters and state sequence. The new algorithm is a combination of the algorithms proposed by Witten and Tibshirani (2010) and Bemporad et al. (2018) for feature selection in clustering and for fitting jump models, respectively. By dividing the problem into simpler subproblems, each of which can be solved efficiently, the proposed algorithm scales well to high-dimensional data sets with large numbers of noisy features.

Third, through computational experiments on two simulated and three real data sets, we show that the jump model with feature selection is remarkably robust to noise and is able to select relevant features. In particular, it yields more accurate state estimates compared to a number of other common methods including HMMs. Our data and implementation in Python is available online as supplementary material.

The outline of this article is as follows. We discuss related work in Section 2. We provide an introduction to jump models in Section 3. In Section 4, we describe our framework for feature selection in jump models. We compare the framework with a number of other methods on simulated data in Section 5. In Section 6, we compare states estimated from time series of financial returns. In Section 7, we consider feature selection and clustering of protein sequences. In Section 8, we evaluate the ability of the jump model with and without feature selection to identify breakpoints between articles from Wikipedia. Finally, Section 9 concludes.

2. Related work

Fitting a jump model is related to change-point detection (Nystrup et al., 2016; Oh & Han, 2000; Ross et al., 2011), segmentation (Hallac et al., 2019; Katz & Crammer, 2015), and trend filtering (Kim et al., 2009) with the fundamental difference that the states are assumed to be recurring. The HMM is a special case of a jump model where the probability distribution that generates an observation depends on the state of an unobserved Markov chain. It is popular in many areas with applications ranging from face recognition (Nefian & Hayes, 1998) and natural language processing (Gales & Young, 2008; Kang et al., 2018) to wind- and solar-power forecasting (Bhardwaj et al., 2013; Pinson et al., 2008), identification of volcanic regimes (Bebbington, 2007), marketing (Netzer et al., 2017), fraud detection (Robinson & Aria, 2018), risk and return modeling in financial and energy markets (Dias et al., 2015; Nystrup et al., 2018; Petropoulos et al., 2016), and genome annotation in computational biology (Choo et al., 2004).

Unlike jump models in general, the HMM is based on specific assumptions about the distributions of data and sojourn times. Being a generalization of HMMs, jump models have recently been shown to be much more robust to high-dimensional settings, limited sample sizes, initialization, and misspecification as compared to HMMs (Nystrup, Kolm et al., 2020; Nystrup et al., 2020b). They can be efficiently estimated using the coordinate descent algorithm proposed by Bemporad et al. (2018) that alternates between fitting the model parameters and state sequence. Given their robustness and scalability, jump models are a promising alternative to traditional HMMs, especially in high-dimensional settings.

It is difficult, if at all possible, to expand the number of features in an effort to improve the HMM while maintaining persistence in the estimated state sequence, which is critical in many applications. Despite their popularity, feature selection in HMMs has attracted surprisingly little attention (Adams & Beling, 2017). A likely reason for this is that the maximum likelihood estimator for the HMM does not work well in high-dimensional settings, where feature selection is most commonly applied (Fiecas et al., 2017).

A practical solution is to apply a dimensionality-reduction technique, such as principal component analysis (PCA), prior to fitting the HMM (e.g., Filion et al., 2010; Georgoulas et al., 2013). However, the objective of dimensionality reduction is different from that of feature selection, and there is no guarantee that a reduced feature space retains the information necessary to distinguish between states (Chang, 1983).

Adams et al. (2016) extended the EM algorithm for estimating HMMs to include binary feature saliency parameters. Unfortunately, assumptions about the distribution of features and their conditional independence given the state typically do not hold in practice. At the same time, their estimation algorithm does not scale well to high-dimensional data.

Feature selection is critical when the number of features is large compared to the number of observations per state and the underlying states differ only with respect to a subset of the features. It yields interpretable results by establishing which features are responsible for the observed differences between the states. In addition, it can reduce computational complexity as fewer features are required to assign each new observation to a preexisting state (Witten & Tibshirani, 2010).

Several approaches to unsupervised feature selection have been proposed in the literature. Common approaches include removing redundancy based on a measure of feature similarity (Mitra et al., 2002; Zhao & Liu, 2007), ranking features based on scores computed independently for each feature (He et al., 2005), projection onto a subspace with lower dimension (Tang et al., 2020, 2021), and feature weighting (de Amorim, 2019). We note that feature selection cannot be applied prior to fitting an HMM as a means to alleviate dimensionality issues, because it requires an estimate of the underlying state sequence. Consequently, in jump models feature selection, parameter and state-sequence estimation need to be done jointly. This is the main motivation for the framework we present in this article. The ability to leverage information embedded in the ordering of the data to improve accuracy separates it from other approaches to unsupervised feature selection proposed in the past.

3. Jump models

A jump model switches infrequently between a finite collection of states or submodels to fit a sequence of data while taking the ordering of the data into account. Bemporad et al. (2018) proposed to fit jump models with K states by minimizing the objective function

$$\sum_{t=1}^{T-1}\left[\ell\left(\boldsymbol{y}_t,\boldsymbol{\mu}_{s_t}\right)+\lambda\mathbb{I}_{s_t\neq s_{t+1}}\right]+\ell\left(\boldsymbol{y}_T,\boldsymbol{\mu}_{s_T}\right) \tag{1}$$

over the model parameters $\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_K\in\mathbb{R}^P$ and the state sequence $s_1,\dots,s_T$, where $\boldsymbol{y}_1,\dots,\boldsymbol{y}_T\in\mathbb{R}^P$ is a sequence (e.g., a time series) of T observations of P (standardized) features, $\ell(\boldsymbol{y},\boldsymbol{\mu})$ is a loss function, $\lambda\geq 0$ is a regularization parameter, and $\mathbb{I}$ is the indicator function.

In this article, we consider the squared Euclidean distance $\ell(\boldsymbol{y},\boldsymbol{\mu})=\|\boldsymbol{y}-\boldsymbol{\mu}\|^2$ as loss function, which results in the objective function (1) for $\lambda=0$ being the same as that for K-means clustering (Lloyd, 1982). It is not surprising that this loss function is useful for fitting jump models in light of K-means clustering being a successful initialization strategy for maximum likelihood estimation of hidden Markov and semi-Markov models (Maruotti & Punzo, 2021). We refer to the resulting model as the jump model or standard jump model. Fig. 1 illustrates the difference between states estimated using a jump model and K-means clustering.

3.1. Jump penalty

Intuitively, the objective function (1) describes a tradeoff between fitting the data and prior assumptions about the persistence of the state sequence. That the state sequence changes infrequently simply means that at any point the model is more likely to remain in the same state than to jump to another state.

Fixed cost. The persistence is determined by the regularization parameter λ, a fixed-cost penalty for jumps. For λ large enough, the jump model results in a single state model. When $\lambda=0$, the model reduces to splitting the dataset in at most K states and fitting one model per state, thereby generalizing the K-means algorithm. Hence, the jump model allows us to infer the number of states from the data. This is in contrast to standard K-means where the number of states or clusters is chosen a priori and model complexity is determined by rerunning the estimation for different values of K (Nystrup, Kolm et al., 2020). In practice, the jump penalty needs to be selected based on the specific application in mind, for example, by cross-validation.

Transition probability. Bemporad et al. (2018) proved that the HMM is a special case of a jump model where the loss function is given by a negative log-likelihood function and the jump penalty λ is the negative logarithm of the transition probability. Thus, there is a direct link between the jump penalty and transition probabilities, which can be exploited to forecast future states. In fact, it is possible to estimate HMMs using the objective function (1) with the squared Euclidean distance as loss function applied to simple endogenous features such as rolling means and standard deviations (Nystrup, Kolm et al., 2020; Nystrup et al., 2020b). It is straightforward to extend (1) to include separate jump penalties for each possible transition. Although this increases the complexity of the model by imposing additional prior assumptions, it does not increase the computational complexity of the estimation problem.

Persistence. In many applications, it is desirable that the state sequence has a certain level of persistence. If an HMM is misspecified (e.g., because the Markov assumption or the conditional distributions are incorrect) or not properly estimated (e.g., due to limited data being available, high-dimensionality, nonstationarity, or failure to converge to the global optimum in the estimation process), typically it leads to unstable and impersistent estimates of the underlying state sequence and the introduction of unnecessary states (Bulla, 2011; Fiecas et al., 2017; Fox et al., 2011). When the state sequence contains far too many jumps the model reduces to a mixture model with no sequential information. Penalizing jumps is a natural way to ensure persistence of the estimated state process and improve stability of the model.

3.2. Estimation

Jump models can be fit by a coordinate descent algorithm that alternates between finding the model parameters that minimize the loss function for a given state sequence (i.e., fitting the model) and finding the state sequence that minimizes the objective function (1) given the parameters (i.e., fitting the state sequence) (Bemporad et al., 2018). This process is repeated 10 times at the most or until the state sequence does not change, which usually happens after less than five iterations.

While each iteration improves the value of the objective function (1), there is no guarantee that the solution found is the global optimum. The solution found depends on the initial state sequence, similar to standard K-means clustering. To improve the quality of the solution, we run the coordinate descent algorithm from several different initial state sequences (in parallel) and keep the model that achieves the lowest objective value (Arthur & Vassilvitskii, 2007).

The parameters $\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_K$ are the conditional means of the features assigned to each of the K states. The most likely sequence of states is found using dynamic programming (Bellman, 1957). Define

$$V(T,s)=\|\boldsymbol{y}_T-\boldsymbol{\mu}_s\|^2, \tag{2}$$

$$V(t,s)=\|\boldsymbol{y}_t-\boldsymbol{\mu}_s\|^2+\min_j\left\{V(t+1,j)+\lambda\mathbb{I}_{s\neq j}\right\},\quad t=T-1,\dots,1. \tag{3}$$

The most likely sequence of states is then given by

$$s_1=\operatorname*{argmin}_j V(1,j), \tag{4}$$

$$s_t=\operatorname*{argmin}_j\left\{V(t,j)+\lambda\mathbb{I}_{s_{t-1}\neq j}\right\},\quad t=2,\dots,T. \tag{5}$$

Note that (2)–(5) become the Viterbi (1967) algorithm if the sequential order of operations is reversed.

Complexity. Finding the most likely state sequence requires $O(TK^2)$ operations plus $O(TPK)$ operations to evaluate the loss function for each t and k, which is the same as one forward and backward pass with the EM algorithm. While the jump estimator and EM algorithm have the same computational complexity per iteration, the jump estimator converges faster in practice. Hard clustering algorithms, such as the jump estimator, generally converge faster than soft clustering algorithms (Bottou & Bengio, 1995). In our simulation study, typically the jump estimator requires less than five iterations, whereas the EM algorithm needs between 50 and 100 iterations to converge, depending on the signal-to-noise ratio.[1] Fitting jump models also requires significantly fewer

[1] We note that a number of methods have been proposed to accelerate the sometimes slow convergence of the EM algorithm, such as those using conjugate gradient and modified Newton methods (Bulla & Berzel, 2007; Jamshidian & Jennrich, 1997).

iterations compared to K-means, which has time complexity $O(TPK)$ per iteration using Lloyd's (1982) algorithm, because the jump penalty reduces the possibilities for the state assignments to change.

4. Feature selection

To make the jump model more robust to noisy data and high dimensionality, the objective function (1) can be extended by adding a regularization penalty on the parameters $\boldsymbol{\mu}$ (Bemporad et al., 2018). For example, quadratic ridge regularization, $\sum_{k=1}^{K} \|\boldsymbol{\mu}_k\|^2$, corresponds to assuming a Gaussian prior, whereas the lasso penalty, $\sum_{k=1}^{K} \|\boldsymbol{\mu}_k\|_1$, corresponds to a Laplace prior. This is one way to implement feature selection in jump models; but it comes at the cost of not being able to compute analytically the optimal model parameters for a given state sequence. It also does not guarantee that the features selected are the same in all states. Therefore, this approach is not useful for determining which features are responsible for the differences between the states, which is our goal in this article.

4.1. Sparse K-means clustering

A different approach to feature selection is to introduce a weighting of each feature that is the same across all states. However, a weighted version of the usual K-means objective of minimizing the within-cluster sum of squares (WCSS)

$$\begin{aligned} &\text{minimize} && \boldsymbol{w}' \textstyle\sum_{t=1}^{T} \left(\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}\right)^2 \\ &\text{subject to} && \|\boldsymbol{w}\|^2 \leq 1, \qquad w_p \geq 0 \quad \forall p, \end{aligned} \tag{6}$$

with respect to the feature weights $w_1, \dots, w_P$, where the squared value is elementwise, has the trivial solution $w_1 = \cdots = w_P = 0$.

Numerous approaches to feature weighting in K-means have been proposed in the literature (see de Amorim, 2016, for a survey). We employ the criterion proposed by Witten and Tibshirani (2010) for finding clusters and feature weights based on maximizing the weighted between-cluster sum of squares (BCSS). The total sum of squares (TSS) is constant and equal to the sum of the WCSS and BCSS:

$$\sum_{t=1}^{T} \|\boldsymbol{y}_t - \bar{\boldsymbol{\mu}}\|^2 = \sum_{t=1}^{T} \|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}\|^2 + \sum_{k=1}^{K} n_k \|\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}}\|^2, \tag{7}$$

where n_k is the number of observations in state k and $\bar{\boldsymbol{\mu}}$ is the unconditional mean of the observations. Hence, minimizing the WCSS is equivalent to maximizing the BCSS. A large BCSS indicates that the clusters are spread out, while a small value indicates that they are close together.

Their sparse K-means clustering criterion can be formulated as

$$\begin{aligned} &\text{maximize} && \boldsymbol{w}' \textstyle\sum_{k=1}^{K} n_k \left(\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}}\right)^2 \\ &\text{subject to} && \|\boldsymbol{w}\|^2 \leq 1, \qquad \|\boldsymbol{w}\|_1 \leq \kappa, \\ & && w_p \geq 0 \quad \forall p, \end{aligned} \tag{8}$$

with respect to the model parameters $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ (i.e., the conditional means), state sequence $s_1, \dots, s_T$, and feature weights $w_1, \dots, w_P$. The feature weights are a measure of the importance of the features. They will be sparse for an appropriate choice of the tuning parameter $1 \leq \kappa \leq \sqrt{P}$ that controls the degree of sparsity. The squared ℓ_2 penalty serves an important role, since without it, at most one element of w would be nonzero in general when features are correlated (Zou & Hastie, 2005). Note that it is only possible to maximize the BCSS if multiple states are estimated. If $w_1 = \cdots = w_P$, then (8) reduces to the standard K-means clustering criterion.

Algorithm 1 Sparse jump model fitting.

Input: Sequence $\boldsymbol{y}_1, \dots, \boldsymbol{y}_T$ of P standardized features, number of latent states K, jump penalty λ, and feature limit κ.

1. Initialize feature weights $\boldsymbol{w}$ as $w_1^0, \dots, w_P^0 = \frac{1}{\sqrt{P}}$.
2. Iterate for $i = 1, \dots$ until $\|\boldsymbol{w}^i - \boldsymbol{w}^{i-1}\|_1 / \|\boldsymbol{w}^{i-1}\|_1 < 10^{-4}$:
   (a) Compute sequence of weighted features $\boldsymbol{z}_t = \boldsymbol{y}_t \cdot \operatorname{diag}\left(\sqrt{\boldsymbol{w}^{i-1}}\right)$, where $t = 1, \dots, T$.
   (b) Initialize state sequence $\boldsymbol{s}$ as $s_1^0, \dots, s_T^0$.
   (c) Iterate for $j = 1, \dots$ until $\boldsymbol{s}^j = \boldsymbol{s}^{j-1}$:
      i. Fit model parameters $\boldsymbol{\mu}^j = \operatorname{argmin}_{\boldsymbol{\mu}} \sum_{t=1}^{T} \|\boldsymbol{z}_t - \boldsymbol{\mu}_{s_t^{j-1}}\|^2$.
      ii. Fit state sequence $\boldsymbol{s}^j = \operatorname{argmin}_{\boldsymbol{s}} \left\{ \sum_{t=1}^{T-1} \left[\|\boldsymbol{z}_t - \boldsymbol{\mu}_{s_t}^j\|^2 + \lambda \mathbb{I}_{s_t \neq s_{t+1}} \right] + \|\boldsymbol{z}_T - \boldsymbol{\mu}_{s_T}^j\|^2 \right\}$.
   (d) Update weights $\boldsymbol{w}^i$ by solving (9) using soft thresholding while holding the parameters $\boldsymbol{\mu}^j$ and state sequence $\boldsymbol{s}^j$ fixed.

Output: Model parameters $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$, state sequence $s_1, \dots, s_T$, and feature weights $w_1, \dots, w_P$.

4.2. Sparse jump models

Bringing it all together, we propose to fit sparse jump models by solving:

$$\begin{aligned} &\text{maximize} && \boldsymbol{w}' \textstyle\sum_{k=1}^{K} n_k \left(\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}}\right)^2 - \lambda \sum_{t=1}^{T-1} \mathbb{I}_{s_t \neq s_{t+1}} \\ &\text{subject to} && \|\boldsymbol{w}\|^2 \leq 1, \qquad \|\boldsymbol{w}\|_1 \leq \kappa, \\ & && w_p \geq 0 \quad \forall p, \end{aligned} \tag{9}$$

with respect to the model parameters $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$, state sequence $s_1, \dots, s_T$, and feature weights $w_1, \dots, w_P$. For a fixed sequence of states, the two problems (8) and (9) are equivalent. Moreover, if $w_1 = \cdots = w_P$, then (9) reduces to the standard jump fitting criterion (1) with the squared Euclidean distance as loss function. Note that although we focus on the squared Euclidean distance the framework will work with any (convex) dissimilarity measure as loss function.

Convergence. Algorithm 1 outlines how (9) can be solved using a coordinate descent algorithm. First, holding $\boldsymbol{w}$ fixed, we optimize (9) with respect to $\boldsymbol{\mu}$ and $\boldsymbol{s}$. Second, holding $\boldsymbol{\mu}$ and $\boldsymbol{s}$ fixed, we optimize (9) with respect to $\boldsymbol{w}$. In general, we do not achieve a global optimum of (9) using this iterative approach since the problem is nonconvex and the first optimization involves an application of the standard jump fitting algorithm to a weighted version of the data, which is not guaranteed to find a global optimum. However, we are guaranteed that each iteration increases the objective function. We finish after a maximum of 10 updates of the feature weights or once $\|\boldsymbol{w}^i - \boldsymbol{w}^{i-1}\|_1 / \|\boldsymbol{w}^{i-1}\|_1 < 10^{-4}$, where $\boldsymbol{w}^i$ denotes the weights obtained at iteration i. The convergence of the feature weights is abrupt, as they do not change from one iteration to the next if the state sequence has not changed. This usually takes less than five iterations, as illustrated in Fig. 2, where the convergence of the feature weights is shown for each iteration when fitting sparse jump models to 100 simulated series of $T = 500$ observations using the same setup as in the simulation study in Section 5.3.

Random initialization. In Step 2b, the initial state sequence is generated randomly using K-means++ (Arthur & Vassilvitskii, 2007), before fitting a jump model in Step 2c. This process is repeated 10 times, similar to standard jump model fitting, with the important exception

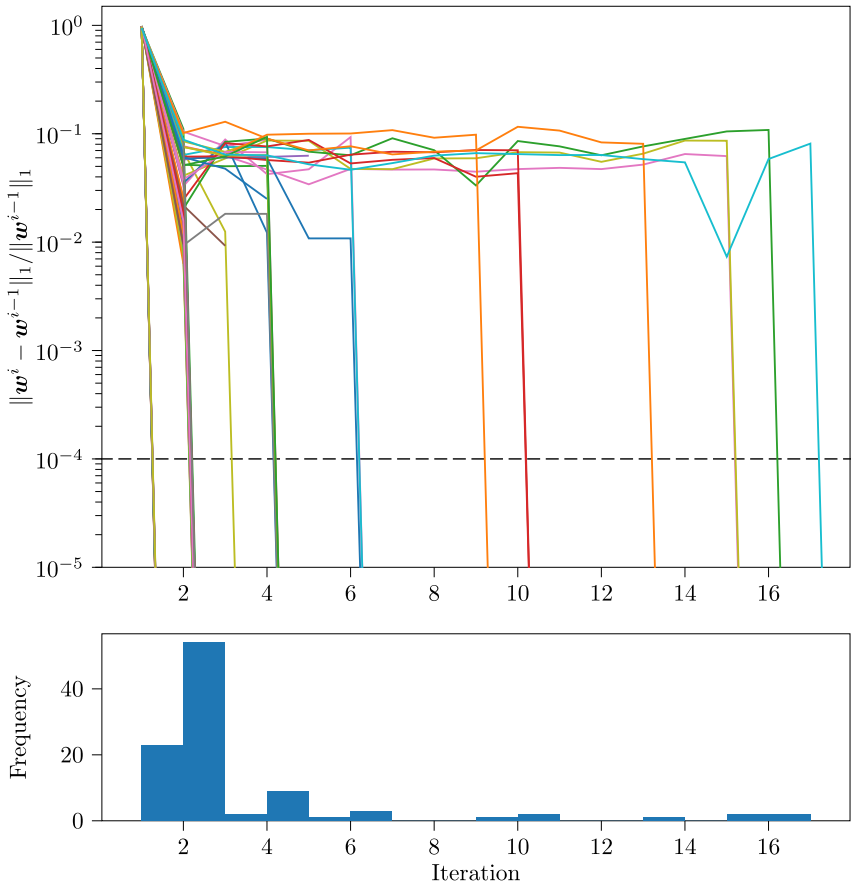

Fig. 2. Convergence of feature weights along with a histogram showing the number of iterations required when fitting sparse jump models to 100 simulated series of $T = 500$ observations with $\mu = 1$ and $P = 60$. The dashed line shows the threshold used as stopping criterion.

that we keep the best state sequence from iteration $i - 1$ and use that as one of the 10 initial state sequences. This ensures that the objective function always improves. Continuing random initialization beyond iteration $i = 1$ increases computational complexity (although it can be done in parallel), but it also reduces the risk of getting stuck in a local optimum. In simulations we find that this approach leads to significantly more accurate state estimates for both sparse K-means and the sparse jump model. It is a significant advantage of this approach to fitting jump models that it is far more robust to initialization than traditional maximum likelihood estimation of HMMs (Maruotti & Punzo, 2021). Similar to other studies, we find that K-means++ performs well in combination with repetitions (Celebi et al., 2013; Fränti & Sieranoja, 2019). We leave it for future work to consider more advanced initialization techniques.

Updating w. In Step 2d, solving (9) with respect to $\boldsymbol{w}$ with $\boldsymbol{\mu}$ and $\boldsymbol{s}$ held fixed can be done using soft thresholding with time complexity $O(TP)$. The solution to the convex problem, which follows from the Karush–Kuhn–Tucker conditions (Boyd & Vandenberghe, 2004), is $\boldsymbol{w} = \frac{S(\boldsymbol{x}_+, \Delta)}{\|S(\boldsymbol{x}_+, \Delta)\|}$, where $\boldsymbol{x} = \sum_{k=1}^{K} n_k \left(\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}}\right)^2$ is a vector of BCSS per feature, $\boldsymbol{x}_+$ denotes the positive part of the elements in $\boldsymbol{x}$, and the soft-thresholding operator is defined as $S(\boldsymbol{x}, c) = \text{sign}(\boldsymbol{x}) \circ (|\boldsymbol{x}| - c)_+$. Here, $\Delta = 0$ if that results in $\|\boldsymbol{w}\|_1 \leq \kappa$; otherwise we choose $\Delta > 0$ to yield $\|\boldsymbol{w}\|_1 = \kappa$. This assumes that there is a unique maximal element of $\boldsymbol{x}$ and that $1 \leq \kappa \leq \sqrt{P}$ (Witten & Tibshirani, 2010).

Selecting λ and κ. Witten and Tibshirani (2010) proposed to choose the value of κ that maximizes the gap between the value of the objective function (8) computed on the data and on random permutations of the data. Because λ and κ are mutually dependent, we recommend selecting their value based on the specific application in mind by cross-validation or backtesting. For example, if the estimated states are used as input to an investment strategy, then the values that maximize risk-adjusted return subject to transaction costs in a backtest are a natural choice (Nystrup et al., 2020a). We leave more systematic hyperparameter optimization in the sparse jump model for future work.

Sparsity. We refer to models fit using Algorithm 1 as sparse jump models due to the sparsity in the features induced by the ℓ_1 penalty on $\boldsymbol{w}$. Compared with the framework for feature selection in clustering proposed by Witten and Tibshirani (2010), the jump model is well suited to replace K-means because it requires very few iterations to fit. In simulations we find that the state sequence does not change much after the first iteration. It should be clear from Algorithm 1 that in Step 2c we could just as well fit a regime-switching linear regression model (Bemporad et al., 2018), a regime-switching ARMAX model (Breschi et al., 2018), or any other kind of jump model.

Robustness. Given the nested structure of Algorithm 1, where a model is fit for every update of the feature weights, it is essential that the estimation method is robust and quick. Jump model fitting fulfills both these criteria. Nystrup, Kolm et al. (2020) and Nystrup et al. (2020b) showed that the fitting of jump models is robust to initialization, high-dimensionality, small sample sizes as well as the distributions of data and sojourn times. At the same time, the jump model is a significant extension of standard, order-independent clustering. The combination of the robustness of the jump model with a framework for feature selection has a promising potential in a number of practical applications, some of which we will explore in the following sections.

5. Simulation study

We compare the accuracy of state sequences estimated using different models with and without feature selection in a simulation study.

In the simulation study the true state sequence is known, which makes it possible to evaluate the ability of each model to correctly identify the underlying hidden states. We define accuracy as the maximum classification accuracy among all possible alignments of an inferred state sequence. It is necessary to check whether a different permutation of the states yields higher accuracy due to the risk of label switching.

Following Brodersen et al. (2010), we use balanced accuracy

$$\text{BAC} = \frac{1}{K}\sum_{k=1}^{K}\frac{tp_k}{tp_k + fn_k}, \tag{10}$$

which is the average of accuracy per observed state, to avoid inflated performance estimates on imbalanced datasets. Here, tp_k is the number of true positives and fn_k the number of false negatives in state k. If a classifier performs equally well on all states, BAC reduces to conventional accuracy. In contrast, if accuracy is above chance only because a classifier takes advantage of an imbalanced dataset, BAC then, as appropriate, drops to the reciprocal of the number of states.

5.1. Setup

We simulate data from a three-state multivariate Gaussian HMM

$$\boldsymbol{y}_t | \, s_t \sim N\left(\boldsymbol{\mu}_{s_t}, \boldsymbol{I}_P\right), \tag{11}$$

where $\boldsymbol{I}_P$ is an identity matrix of order P and s_t is a first-order Markov chain, with parameters

$$\begin{aligned} &(\boldsymbol{\mu}_1)_p = \mu \mathbb{I}_{p\leq 15}, \qquad \boldsymbol{\mu}_2 = \boldsymbol{0}, \qquad (\boldsymbol{\mu}_3)_p = -\mu \mathbb{I}_{p\leq 15}, \\ &\boldsymbol{\Gamma} = \begin{pmatrix} 0.9903 & 0.0047 & 0.0050 \\ 0.0157 & 0.9666 & 0.0177 \\ 0.0284 & 0.0300 & 0.9416 \end{pmatrix}. \end{aligned} \tag{12}$$

Since the covariance matrix in all states is the identity matrix, all information separating the states is contained in the mean values. In Section 6, we will consider an example where the information separating the states comes from the second moment.

Each series is simulated by first generating a sequence of states of desired length according to the transition probability matrix $\boldsymbol{\Gamma}$ starting from its stationary distribution. Secondly, observations are sampled from multivariate Gaussian distributions with mean parameters $\boldsymbol{\mu}$ given by the simulated states and covariance $\boldsymbol{I}_P$. In state one the first 15 features have mean value μ and in state three they have mean value $-\mu$. All other features have mean zero in all states. Hence, features beyond the first 15 are white noise.

The transition probabilities are based on an HMM of weekly stock returns, which is a common application (Nystrup et al., 2020b). By sampling the states from an underlying Markov chain, rather than simply drawing a uniform number of observations from each distribution, this is an extension of the simulation setup considered by Witten and Tibshirani (2010). Having the mean value in the middle state be equal to the mean values of the noise features increases the importance of feature selection. Naturally, we expect methods that are able to benefit from the persistence in the state process to yield higher classification accuracy.

Despite the features being conditionally independent, the sampling of the states from a Markov chain means that the first 15 features are unconditionally correlated. Consequently, the unconditional correlation structure can be used to distinguish relevant from irrelevant features and to successfully reduce the dimension of the feature space. In Section 5.4, we consider what happens when the noise features are weakly correlated and the unconditional correlation structure is no longer sufficient to make the distinction between relevant and irrelevant features.

5.2. Models

We compare the accuracy of our sparse jump model with a number of competitors on the simulated data. All methods are initialized with 10 different centroid seeds generated using K-means++ (Arthur & Vassilvitskii, 2007) to make the comparison as fair as possible. Increasing the number of random initializations beyond 10 does not significantly improve the BAC of either of the considered models and, therefore, would not change the conclusions drawn from the simulation study. For the HMM we provide better initial values for the transition probabilities to increase the probability of convergence, as detailed below. We estimate the simulated state sequences using the following six different models:

1. *Standard 3-means* using the implementation in `scikit-learn` (Pedregosa et al., 2011) with a maximum of 300 iterations per initialization.
2. *Sparse 3-means* proposed by Witten and Tibshirani (2010). Our implementation follows our implementation of the sparse jump model with the jump model replaced by 3-means. We do a maximum of 10 updates of the feature weights. For each value of μ and P we report the highest BAC from a grid search over 14 equally spaced values of κ between 1 and $\sqrt{P}$.
3. *Gaussian HMMs* estimated using the regularized EM algorithm based on Huang et al. (2001, Chapter 9.6.1) that is implemented in the `hmmlearn` package in Python with a maximum of 100 iterations, the covariance prior set to 1, and the transition probability matrix initialized with self-transition probabilities of 0.95. The state sequences are estimated using the Viterbi (1967) algorithm.
4. The *Gaussian feature saliency HMM* (FSHMM) proposed by Adams et al. (2016). Our implementation is based on the implementation of the Gaussian HMM in the `hmmlearn` package extended with an unconditional mean, unconditional variance, and feature saliency parameter for each feature. The state sequences are estimated using the Viterbi (1967) algorithm.
5. *PCAHMM* using the PCA implementation in `scikit-learn` (Pedregosa et al., 2011) to transform the data based on the first eigenvector followed by estimation of a Gaussian HMM (as above) based on the dimensionality-reduced data.
6. *Standard jump models* fit using the framework proposed by Bemporad et al. (2018) with a maximum of 10 iterations per initialization. For each value of μ and P we report the highest BAC from a grid search over 14 logarithmically spaced values of λ between 10^{-2} and 10^{4}.
7. *Sparse jump models* fit using Algorithm 1. For each value of μ and P we report the highest BAC from a grid search over the Cartesian product of seven logarithmically spaced values of λ between 10^{-1} and 10^{2} and 14 equally spaced values of κ between 1 and $\sqrt{P}$, i.e., 98 combinations in total.

5.3. Results when features are conditionally independent

Table 1 compares the standard 3-means, sparse 3-means, standard HMM, feature saliency HMM, PCAHMM, standard jump, and sparse jump model for different values of μ and for different numbers of features P when the features are conditionally independent. The reported values are the mean (and standard deviation) of the BAC over 100 simulations of $T = 500$ observations for each combination of μ and P. The entries for which the BAC of the sparse jump model is significantly higher than that of the standard jump model are shown in bold. We use the Wilcoxon (1945) signed-rank test to determine whether the improvements in BAC are statistically significant at level $\alpha = 0.05$.

Table 1
Comparison of standard 3-means, sparse 3-means, standard HMM, feature saliency HMM, PCAHMM, standard jump, and sparse jump model when all features are conditionally independent. The reported values are the mean (and standard deviation) of the BAC over 100 simulations of $T = 500$ observations for each value of μ and for each number of features P. The entries for which the BAC of the sparse jump model is significantly higher than that of the standard jump model (at level $\alpha = 0.05$) are shown in bold.

	$P = 15$	$P = 30$	$P = 60$	$P = 150$	$P = 300$
$\mu = 0.25$					
3-means	0.47 (0.07)	0.45 (0.06)	0.42 (0.05)	0.38 (0.04)	0.36 (0.04)
Sparse 3-means	0.41 (0.05)	0.40 (0.05)	0.39 (0.04)	0.37 (0.03)	0.37 (0.04)
HMM	0.59 (0.10)	0.56 (0.10)	0.52 (0.11)	0.41 (0.07)	0.38 (0.05)
FSHMM	0.60 (0.11)	0.60 (0.11)	0.58 (0.11)	0.51 (0.11)	0.45 (0.10)
PCAHMM	0.67 (0.18)	0.64 (0.19)	0.57 (0.17)	0.42 (0.12)	0.36 (0.06)
Jump	0.69 (0.14)	0.60 (0.12)	0.57 (0.12)	0.49 (0.12)	0.44 (0.10)
Sparse jump	0.63 (0.14)	**0.64 (0.12)**	**0.62 (0.15)**	**0.59 (0.15)**	**0.57 (0.13)**
$\mu = 0.5$					
3-means	0.54 (0.11)	0.54 (0.10)	0.53 (0.10)	0.51 (0.09)	0.46 (0.08)
Sparse 3-means	0.49 (0.09)	0.48 (0.08)	0.49 (0.08)	0.48 (0.08)	0.46 (0.08)
HMM	0.70 (0.19)	0.65 (0.18)	0.61 (0.15)	0.55 (0.10)	0.53 (0.12)
FSHMM	0.70 (0.13)	0.68 (0.14)	0.68 (0.11)	0.64 (0.10)	0.62 (0.12)
PCAHMM	0.86 (0.17)	0.90 (0.15)	0.89 (0.15)	0.84 (0.18)	0.80 (0.20)
Jump	0.85 (0.17)	0.84 (0.16)	0.77 (0.16)	0.67 (0.11)	0.60 (0.09)
Sparse jump	0.86 (0.16)	**0.89 (0.14)**	**0.89 (0.14)**	**0.87 (0.16)**	**0.88 (0.14)**
$\mu = 0.75$					
3-means	0.68 (0.18)	0.63 (0.17)	0.59 (0.15)	0.54 (0.11)	0.54 (0.10)
Sparse 3-means	0.66 (0.20)	0.67 (0.19)	0.68 (0.18)	0.65 (0.19)	0.67 (0.18)
HMM	0.81 (0.21)	0.75 (0.21)	0.69 (0.20)	0.59 (0.14)	0.55 (0.10)
FSHMM	0.78 (0.18)	0.77 (0.18)	0.74 (0.18)	0.65 (0.18)	0.63 (0.15)
PCAHMM	0.90 (0.16)	0.93 (0.15)	0.94 (0.12)	0.93 (0.14)	0.91 (0.15)
Jump	0.90 (0.16)	0.90 (0.16)	0.87 (0.17)	0.77 (0.16)	0.68 (0.11)
Sparse jump	0.91 (0.14)	**0.95 (0.12)**	**0.95 (0.11)**	**0.95 (0.11)**	**0.94 (0.12)**
$\mu = 1$					
3-means	0.78 (0.21)	0.74 (0.22)	0.68 (0.21)	0.58 (0.14)	0.55 (0.10)
Sparse 3-means	0.77 (0.22)	0.83 (0.21)	0.84 (0.19)	0.81 (0.21)	0.82 (0.20)
HMM	0.86 (0.20)	0.80 (0.22)	0.74 (0.22)	0.63 (0.19)	0.59 (0.13)
FSHMM	0.85 (0.18)	0.84 (0.19)	0.81 (0.20)	0.64 (0.17)	0.65 (0.18)
PCAHMM	0.91 (0.16)	0.93 (0.14)	0.95 (0.13)	0.92 (0.16)	0.93 (0.15)
Jump	0.92 (0.16)	0.93 (0.14)	0.91 (0.16)	0.86 (0.16)	0.75 (0.14)
Sparse jump	0.95 (0.13)	0.96 (0.11)	**0.97 (0.10)**	**0.97 (0.11)**	**0.96 (0.12)**

3-means. Comparing 3-means with sparse 3-means it is evident that feature selection makes the clustering more robust to noisy features. The feature selection is more efficient the larger the size of the mean values. Standard 3-means is always more accurate than sparse 3-means when there are no irrelevant features. Standard 3-means is also more accurate than sparse 3-means for all values of $P \leq 150$ when $\mu \leq 0.5$. When $\mu \geq 0.75$, sparse 3-means is more accurate than standard 3-means for $P \geq 30$, and for the largest values of P the differences in BAC are quite large.

HMM. The standard HMM is always more accurate than standard 3-means. However, sparse 3-means is more accurate compared to both HMMs when $\mu = 0.75$ and $P = 300$ and when $\mu = 1$ and $P \geq 60$. The higher signal-to-noise ratios mean that accounting for the temporal persistence makes a smaller difference. Obviously, the state sequences estimated using 3-means and sparse 3-means are far less persistent, which is a disadvantage in many applications (Nystrup, Kolm et al., 2020; Nystrup et al., 2020b).

FSHMM. The FSHMM yields higher accuracy than the standard HMM, especially when $P \geq 60$, but it is not as robust to large numbers of features as sparse 3-means. We note that the examples considered by Adams et al. (2016) included at most 30 features. The lack of robustness traces back to the sensitivity of the standard HMM to the dimension of the feature space (Fiecas et al., 2017). Augmenting the model with three additional parameters per feature does not alleviate the problem.

PCAHMM. Transforming the data based on the first eigenvector before fitting a standard HMM to the resulting one-dimensional time series in most cases leads to much higher accuracy than the HMM and FSHMM fitted to the P-dimensional time series. The difference in accuracy increases with the dimension P with the exception of $\mu = 0.25$. This shows that dimensionality reduction is only successful when the signal-to-noise ratio is high enough that the unconditional correlation structure can be used to identify the relevant direction in the feature space. PCAHMM is successful because the first eigenvector, which explains the maximum amount of variance, is a linear combination of the first 15 features due to their mutual correlation. Although PCA has a different objective, in this particular case it ends up selecting the right features.

Jump. For nearly all combinations of μ and P, the jump model, even without feature selection, is more accurate than all of the previously considered models, except PCAHMM. It is a testament of the robustness and strength of the jump model that it leads to significantly more accurate state estimates compared to the state sequences estimated using the correctly specified HMM and in most cases also the FSHMM. Even though the transition probabilities differ across states, a single, constant jump penalty is sufficient to improve accuracy.

Sparse jump. The accuracy of the jump model can be significantly improved through feature selection for all values of μ whenever $P \geq 60$. The sparse jump model also has higher accuracy when $P = 30$, but the differences are not always significant. Feature selection ensures that the BAC barely changes as P increases when $\mu > 0.25$. This is also true for sparse 3-means, only the BAC is at a much higher level. In nearly all cases the sparse jump model also leads to higher accuracy than PCAHMM, despite not benefiting from the information in the unconditional correlation structure.

Dependency on μ, T, and P. In addition to the feature means μ and the number of irrelevant features, the BAC in Table 1 is also a function of the number of relevant features and the number of observations T. Increasing either of the latter would lead to higher BAC. The ratio between P and T also makes a difference for the models that do not include feature selection nor dimensionality reduction. The BAC of the

Table 2
Comparison of standard 3-means, sparse 3-means, standard HMM, feature saliency HMM, PCAHMM, standard jump, and sparse jump model when the noise features are correlated with coefficient 0.1. The reported values are the mean (and standard deviation) of the BAC over 100 simulations of $T = 500$ observations for each value of μ and for each number of features P. The entries for which the BAC of the sparse jump model is significantly higher than that of the standard jump model (at level $\alpha = 0.05$) are shown in bold.

	$P = 15$	$P = 30$	$P = 60$	$P = 150$	$P = 300$
$\mu = 0.25$					
3-means	0.47 (0.07)	0.43 (0.05)	0.36 (0.03)	0.35 (0.03)	0.36 (0.03)
Sparse 3-means	0.41 (0.05)	0.38 (0.05)	0.37 (0.03)	0.36 (0.04)	0.37 (0.03)
HMM	0.59 (0.10)	0.50 (0.10)	0.34 (0.03)	0.35 (0.03)	0.35 (0.03)
FSHMM	0.60 (0.11)	0.48 (0.14)	0.36 (0.04)	0.35 (0.03)	0.35 (0.03)
PCAHMM	0.67 (0.18)	0.35 (0.05)	0.34 (0.04)	0.35 (0.05)	0.34 (0.04)
Jump	0.69 (0.14)	0.61 (0.13)	0.50 (0.13)	0.42 (0.11)	0.38 (0.09)
Sparse jump	0.63 (0.14)	**0.65 (0.14)**	**0.59 (0.19)**	0.44 (0.13)	**0.40 (0.10)**
$\mu = 0.5$					
3-means	0.54 (0.11)	0.54 (0.09)	0.50 (0.09)	0.36 (0.03)	0.36 (0.03)
Sparse 3-means	0.49 (0.09)	0.47 (0.09)	0.36 (0.04)	0.36 (0.04)	0.37 (0.04)
HMM	0.70 (0.19)	0.55 (0.09)	0.53 (0.11)	0.35 (0.03)	0.35 (0.03)
FSHMM	0.70 (0.13)	0.64 (0.13)	0.51 (0.15)	0.36 (0.04)	0.36 (0.03)
PCAHMM	0.86 (0.17)	0.73 (0.26)	0.35 (0.05)	0.35 (0.06)	0.34 (0.04)
Jump	0.85 (0.17)	0.78 (0.17)	0.67 (0.13)	0.54 (0.14)	0.41 (0.10)
Sparse jump	0.86 (0.16)	**0.88 (0.16)**	**0.89 (0.14)**	**0.74 (0.29)**	**0.61 (0.25)**
$\mu = 0.75$					
3-means	0.68 (0.18)	0.55 (0.09)	0.56 (0.09)	0.38 (0.06)	0.36 (0.03)
Sparse 3-means	0.66 (0.20)	0.66 (0.18)	0.64 (0.19)	0.38 (0.07)	0.37 (0.03)
HMM	0.81 (0.21)	0.59 (0.15)	0.56 (0.09)	0.37 (0.08)	0.35 (0.03)
FSHMM	0.78 (0.18)	0.69 (0.17)	0.61 (0.16)	0.40 (0.09)	0.35 (0.03)
PCAHMM	0.90 (0.16)	0.92 (0.16)	0.41 (0.14)	0.35 (0.05)	0.34 (0.04)
Jump	0.90 (0.16)	0.88 (0.17)	0.82 (0.17)	0.63 (0.08)	0.51 (0.13)
Sparse jump	0.91 (0.14)	**0.94 (0.13)**	**0.96 (0.11)**	**0.94 (0.14)**	**0.84 (0.20)**
$\mu = 1$					
3-means	0.78 (0.21)	0.61 (0.17)	0.56 (0.09)	0.50 (0.11)	0.36 (0.03)
Sparse 3-means	0.77 (0.22)	0.83 (0.20)	0.83 (0.19)	0.65 (0.25)	0.36 (0.03)
HMM	0.86 (0.20)	0.64 (0.19)	0.55 (0.07)	0.51 (0.13)	0.36 (0.03)
FSHMM	0.85 (0.18)	0.72 (0.20)	0.62 (0.17)	0.50 (0.13)	0.37 (0.05)
PCAHMM	0.91 (0.16)	0.93 (0.15)	0.74 (0.28)	0.35 (0.05)	0.34 (0.04)
Jump	0.92 (0.16)	0.91 (0.16)	0.86 (0.17)	0.68 (0.12)	0.59 (0.11)
Sparse jump	0.95 (0.13)	**0.96 (0.12)**	**0.97 (0.09)**	**0.97 (0.11)**	**0.90 (0.16)**

standard jump model, for example, does not decline much when going from $P = 15$ to $P = 60$ when $\mu \geq 0.75$, but it drops fast when going from $P = 60$ to $P = 150$. This tipping point would occur at a larger value of P, had more observations been available for estimation. On the other hand, $P = T$ is not as critical a tipping point as it is for many other estimation problems.

5.4. Results when noise features are correlated

Next, we consider what happens when the noise features are weakly correlated and the unconditional correlation structure is no longer sufficient to make the distinction between relevant and irrelevant features. To generate correlated Gaussian noise features, we first construct uncorrelated features as before, and then multiply them by a matrix C, where $CC' = \Sigma$ is the Cholesky decomposition of the desired covariance matrix Σ. The covariance matrix for the correlated noise features is an identity matrix of size $P - 15$ with off-diagonal elements populated with the desired correlation coefficient of 0.1.

Table 2 compares the standard 3-means, sparse 3-means, standard HMM, feature saliency HMM, PCAHMM, standard jump, and sparse jump model when the noise features are correlated with correlation coefficient 0.1. Compared with Table 1, the BAC of all models is a lot lower when $P \geq 150$ and $\mu \leq 0.5$. The sparse jump model stands out as being the most robust to the correlated noise. Across all values of μ, the BAC of the sparse jump model is more robust to increasing values of P than the other models. In most cases, the sparse jump model maintains the same high BAC as before.

Another noticeable difference is that the BAC of PCAHMM, which worked well when the noise was uncorrelated, declines sharply as P increases. In many cases, its BAC is lower than that of the standard HMM, and only when $P \leq 30$ does it compare to that of the jump model. PCA is not successful in the presence of correlated noise, because the linear combination of the features that captures the most variance consists of a mixture of distinguishing *and* pure noise features. This example illustrates an important difference in the objectives of feature selection and dimensionality reduction.

5.5. Computation time

The results in Tables 1 and 2 confirm the great potential of combining jump models with feature selection. Before turning to the real data applications, we compare the time required to fit each of the six models to the simulated data with conditionally independent features. The experiments are conducted on a single core of a 1.9 GHz Intel i7-8650U CPU with 16GB of memory. We note that for both 3-means, sparse 3-means, jump, and sparse jump computations could potentially be sped up by parallelizing the estimations from different initial state sequences.

Fig. 3 shows the average time in seconds it takes to fit each model as a function of the number of features P when $\mu = 1$ and each simulated series consists of $T = 500$ observations. Standard 3-means is the fastest to fit, closely followed by PCAHMM and the standard HMM. We note that these three models are fit using standard Python libraries while the other models are fit using our own implementation or a mix, which can likely be optimized for speed. The first two scale a bit better with P than the latter. The reduction in time complexity when fitting the HMM to a one-dimensional rather than P-dimensional time series more than offsets the cost of transforming the data based on the first eigenvector, especially when P is large.

Of course sparse 3-means is slower than 3-means, since it involves fitting multiple 3-means models. The time complexity of sparse 3-means is also linear in P, but with a much steeper slope compared to standard 3-means. Sparse 3-means is slower than the standard jump model. It is also slower than the sparse jump model when $P > 150$.

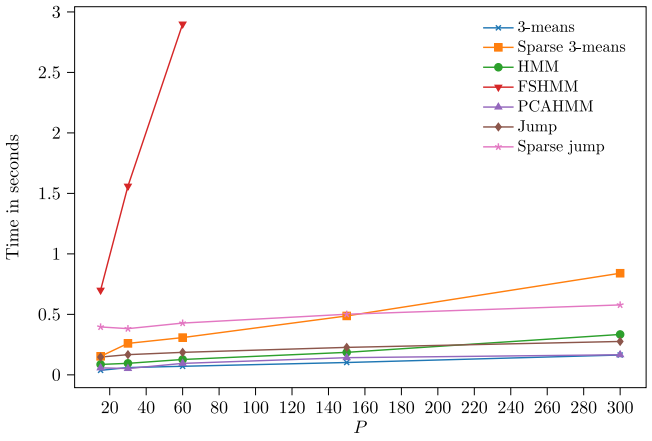

Fig. 3. Average time in seconds to fit each model as a function of the number of features P when $\mu = 1$ and $T = 500$.

The computation time for both the jump model and the sparse jump model grows more like standard 3-means than sparse 3-means as a function of P. When fitting a jump model, each iteration is time consuming and dominated by the cost of finding the most likely sequence of states, which is independent of P. The determining factor is the number of iterations that is needed. Although the time complexity of evaluating the loss function is linear in P, this is not the dominating factor. In addition, the sparse jump model is able to reduce the complexity of loss function evaluations by assigning zero weight to a number of features. As the increase in the number of iterations required is slowing as a function of P, the same is true for the computation time. This means that both the standard and sparse jump model scale well to large numbers of features.

The time complexity of the FSHMM is also linear in P, but with a much steeper slope compared to the HMM due to the computations involved in estimating the additional parameters. The computation time for the FSHMM is only shown for $P \leq 60$, in order to make the differences between the other methods visible from the plot. When $P = 15$, it takes about twice as long to fit the FSHMM as the sparse jump model. When $P = 30$, the FSHMM is a factor of four slower, and when $P = 60$ it is almost a factor of seven slower compared to the sparse jump model. The FSHMM clearly does not scale as well as the sparse jump model to large data sets.

6. Industry portfolio returns

In the first application to real data we consider time series of financial returns. Regime-switching models are popular for inferring the state of financial markets, which is useful for both asset allocation and risk modeling purposes (Ang & Timmermann, 2012; Nystrup et al., 2019; Nystrup, Hansen et al., 2015; Nystrup, Madsen et al., 2017; Yao et al., 2020). The purpose of this example is to show that jump models can be applied to data where the information separating the states comes from the second moment. Moreover, the example compares the ability of PCAHMM and the sparse jump model to select the true features.

6.1. Data description

We consider 2516 daily returns of 49 industry portfolios from 2010 through 2019. The 49 industry portfolios are constructed by assigning each NYSE, AMEX, and NASDAQ stock to an industry portfolio at the end of June each year based on its Standard Industrial Classification (SIC) code at that time. All stocks are equally weighted.[2]

In order to test the ability of the sparse jump model to select the right features we consider two different data sets. The first one consists only of the original 49 time series. The second one consist of 490 time series, where the first 49 are the original time series and the remaining 441 are constructed by nine random permutations of the rows of the original 2516×49 data matrix. By randomly permuting the rows we destroy the temporal information without altering the cross-sectional information and the distributional characteristics.

Financial returns are known to exhibit volatility clustering (Cont, 2001; Nystrup, Madsen et al., 2015). Therefore, we consider the standard deviations of the returns rather than the returns themselves. We construct $P = 49$ and $P = 490$ features, respectively, by computing rolling six-day standard deviations of each of the return series. The smoothing introduced by computing rolling moments is necessary to get persistent state sequences. We choose a window length of six days similar to Nystrup, Kolm et al. (2020).

6.2. Results

6.2.1. State estimation

Fig. 4 shows the cumulative average daily return of the 49 industry portfolios from 2010 through 2019 along with the average annualized six-day volatility in percentages and the resulting state sequences estimated using sparse K-means with $\kappa = 5$, PCAHMM based on the first eigenvector, the jump model with $\lambda = 500$, and the sparse jump model with $\lambda = 50$ and $\kappa = 5$. The values of the hyperparameters are chosen based on a qualitative assessment of the resulting state sequences. We were unable to get a reasonable state sequence using an HMM or FSHMM.

The $K = 3$ states are sorted according to the mean values of the (volatility) features. The state sequences estimated using sparse K-means show very low persistence. If the estimated state sequences were used as the basis of a trading strategy, then the large number of jumps would lead to higher transaction costs.

When only the $P = 49$ true features are included, the two state sequences based on the jump models look fairly similar. Both seem to be in good agreement with the average annualized volatility, although we have no way to measure which state sequence is more accurate as the true states are unknown. With a more inhomogeneous data set, for example returns for different asset classes (Nystrup, Hansen et al., 2017), the resulting state sequences could be quite different.

When the nine random permutations are added, bringing the number of features to $P = 490$, the state sequences estimated using PCAHMM and the jump model without feature selection change. In particular, the state sequences include several very short-lived transitions. It is likely that these transitions are driven by the random features. Comparing the state sequences estimated using the sparse jump model for $P = 49$ and $P = 490$ they look more similar, which illustrates the robustness of the model.

6.2.2. Summary statistics

Table 3 shows the mean and standard deviation of the average log returns of the 49 industry portfolios conditional on the estimated state sequences for $P = 49$ along with the implied transition probability matrices. Using sparse K-means, the states with highest and lowest volatility both have a positive mean. The two states with highest volatility are much less persistent than the state with lowest volatility.

The summary statistics conditional on the state sequences estimated using PCAHMM and the standard and sparse jump models are more

[2] The returns, which are available as supplementary material, are from Kenneth French's data library available at https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html.

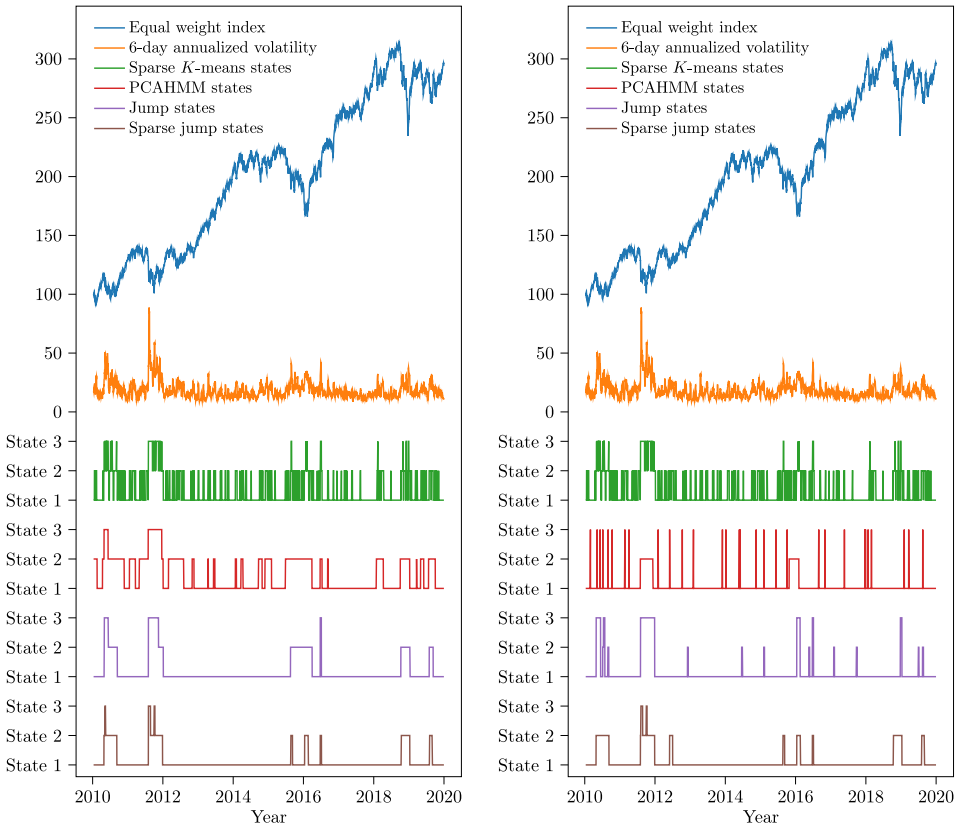

Fig. 4. Impact of random features on the inferred states for the 49 industry portfolios data set. The legends are sorted according to the order of the time series.

Table 3
Summary statistics for the average log returns of the 49 industry portfolios conditional on the estimated state sequences for $P = 49$.

	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	$\boldsymbol{\Gamma}$		
Sparse K-means	$\begin{pmatrix} 0.00066 \\ -0.00043 \\ 0.00165 \end{pmatrix}$	$\begin{pmatrix} 0.0069 \\ 0.0131 \\ 0.0248 \end{pmatrix}$	$\begin{pmatrix} 0.9245 & 0.0743 & 0.0012 \\ 0.1748 & 0.7933 & 0.0319 \\ 0 & 0.2101 & 0.7899 \end{pmatrix}$		
PCAHMM	$\begin{pmatrix} 0.00127 \\ -0.00073 \\ -0.00198 \end{pmatrix}$	$\begin{pmatrix} 0.0100 \\ 0.0152 \\ 0.0259 \end{pmatrix}$	$\begin{pmatrix} 0.9739 & 0.0261 & 0 \\ 0.0412 & 0.9561 & 0.0027 \\ 0 & 0.0175 & 0.9825 \end{pmatrix}$		
Jump	$\begin{pmatrix} 0.00065 \\ -0.00042 \\ -0.00185 \end{pmatrix}$	$\begin{pmatrix} 0.0082 \\ 0.0140 \\ 0.0251 \end{pmatrix}$	$\begin{pmatrix} 0.9971 & 0.0015 & 0.0015 \\ 0.0145 & 0.9855 & 0 \\ 0.0091 & 0.0182 & 0.9727 \end{pmatrix}$		
Sparse jump	$\begin{pmatrix} 0.00060 \\ -0.00022 \\ -0.00097 \end{pmatrix}$	$\begin{pmatrix} 0.0081 \\ 0.0134 \\ 0.0249 \end{pmatrix}$	$\begin{pmatrix} 0.9965 & 0.0020 & 0.0015 \\ 0.0146 & 0.9830 & 0.0024 \\ 0.0085 & 0.0254 & 0.9661 \end{pmatrix}$		

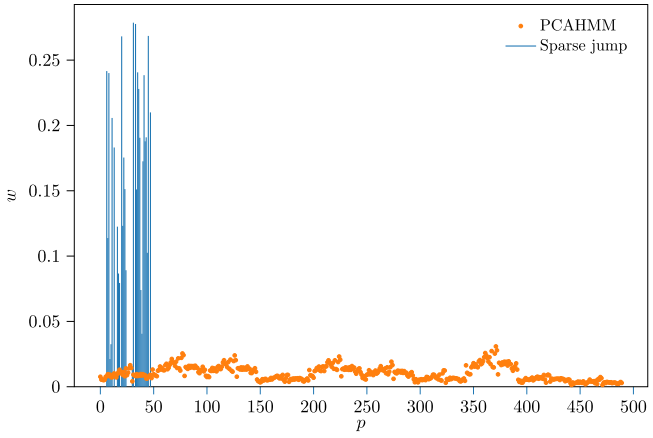

Fig. 5. Feature selection on the 49 industry portfolios data set with randomly permuted features using PCAHMM and the sparse jump model.

intuitive. Only the state with lowest volatility has a positive mean, and the most negative mean is in the state with highest volatility. All three states have much higher persistence than all of the states estimated using sparse K-means. For the jump models, the low-volatility state is the most persistent, while for PCAHMM it is the high-volatility state. Overall, the jump models clearly lead to more persistent state sequences than PCAHMM.

6.2.3. Feature selection

Even though the similarity of the estimated state sequences suggests that the sparse jump model is working well, it is of interest to examine the features selected. Fig. 5 shows the estimated feature weights for the $P = 490$ features. Since $\kappa = 5$ is smaller than $\sqrt{49}$, it only selects a subset of the 49 true features. At the same time it assigns no weight to any of the features $p > 49$. This shows that the sparse jump model is indeed able to distinguish the true features from the randomly permuted features, even in a setting with noisy financial returns, without imposing a prior on the feature weights or having a true state sequence to learn from.

In Fig. 5, for comparison, we show the absolute values of the loadings (rescaled to sum to 5) on the first principal component for the $P = 490$ features. It is evident that PCA is unable to select the relevant features, instead all the features are assigned roughly similar weights. This is another drawback of PCA: rather than selecting a sparse subset of the features, it simply transforms the features.

7. Chromatin proteins

Next, we consider an application to feature selection and order-dependent clustering of chromatin protein sequences. This is an example of sequential, non-time series data suitable for the sparse jump model. Chromatin governs gene regulation and genome maintenance. The partition of chromatin into five states is a human construct that is helpful in distinguishing different biological mechanisms. For background and details about the data we refer to Filion et al. (2010).

7.1. Data description

Filion et al. (2010) analyzed DamID profiles for 53 proteins for six different Drosophila chromosomes (chr2L, chr2R, chr3L, chr3R, chr4, and chrX).[3] We focus on the 33,662 observations for the chr2L chromosome.

Filion et al. (2010) used PCA to reduce the dimension from 53 proteins to three principal components, which together account for 58% of the total variance. The observations of the three principal components were then input to a five-state Student's-t HMM. They used a Student's t conditional distribution rather than a Gaussian as it provided a better fit to the empirical distribution of the data; otherwise outliers in the data would have led to too many state changes.

7.2. Chromatin states

Fig. 6 compares an excerpt of the chromatin states estimated by Filion et al. (2010) using their PCAHMM approach to the states from a sparse jump model fit with $\lambda = 4$ and $\kappa = 3$. We selected these values for the hyperparameters in order to maximize the overlap with their states. For the chr2L chromosome we get the same chromatin state 74% of the time with only 1984 state changes compared to the 2679 detected by Filion et al. (2010).

As the partition of chromatin into five states is a human construct, it is hard to argue whether one approach is better than the other. As for most applications, the fewer the state changes the easier it is to interpret the results.

7.3. Feature selection

Reducing the dimension of the feature space using PCA was necessary in order to successfully estimate the chromatin states using an HMM. Additionally, it is of interest to identify minimal sets of features that capture the same information. Because the experiments are very expensive, measuring as few proteins as possible keeps costs down. This is an example of an application where feature selection is of economic value. We note that PCA is not useful for this purpose, since it requires measurements of *all* proteins to compute the value of a single principal component.

Filion et al. (2010) found that a subset of only five marker proteins (H1, PC, HP1, MRG15, and BRM), which together occupy 97.6% of the genome, could recapitulate their state classification with 85.5% agreement. For comparison, Fig. 7 shows the weights of the proteins as selected by the sparse jump model. The sparse jump model gives weight to four out of the five marker proteins that Filion et al. (2010) selected, which are shown in boldface. It does not select BRM, which is quite specific to the red state. It chooses CTBP instead, and uses several additional features like SIR2 to make the distinction between the red and yellow states. The features we identify could correspond to a better description of the chromatin states; but it is not possible to draw a definitive conclusion without performing additional biological experiments.

[3] The data is available at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE22069.

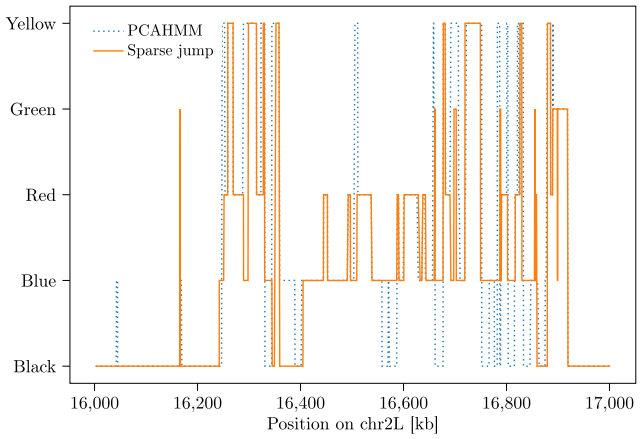

Fig. 6. Identified chromatin states based on the protein sequence.

It is a strength of the sparse jump model that without any biological knowledge, it is able to select meaningful features. In addition to providing an automated and systematic approach to feature selection, the sparse jump model makes it possible to experiment with what the necessary number of features may be. It is not a given that a set of five proteins is optimal simply because it matches the number of chromatin states.

8. Wikipedia text data

Our final example is from natural language processing (NLP) and illustrates how the sparse jump model can be applied to a different type of data set.

8.1. Data description

We look at text data from English-language Wikipedia. The data was obtained by Hallac et al. (2019) by concatenating the introductions, i.e., the text that precedes the Table of Contents section on the Wikipedia webpage, of five separate articles with titles George Clooney, Botany, Julius Caesar, Jazz, and Denmark. Here, the time series consists of the sequence of words from these five articles in order. After basic preprocessing (removing words that appear at least five times and in multiple articles), the data set consists of 1281 words, with each article contributing between 224 and 286 words, as shown in Fig. 8. Each word is converted into a 300-dimensional vector using a pretrained Word2Vec embedding of three million unique words (or short phrases), trained on the Google News data set of approximately 100 billion words. This yields a 1281×300 data matrix, which is available as supplementary material.

The question is whether the sparse jump model can detect the breakpoints between the five concatenated articles, based solely on the change in mean of the feature vectors associated with the words in the vector series. To explore robustness to noise we add additional features (i.e., columns) to the data matrix by randomly permuting the rows of the original data matrix. This ensures that the random features share the same distributional characteristics as the original features.

8.2. Results

We compare the accuracy of the standard jump model and the sparse jump model to that of the greedy Gaussian segmentation (GGS) approach proposed by Hallac et al. (2019) in determining the break points between the five articles. Hallac et al. (2019) showed that GGS compares favorably to a left-to-right HMM (Bakis, 1976) on a similar

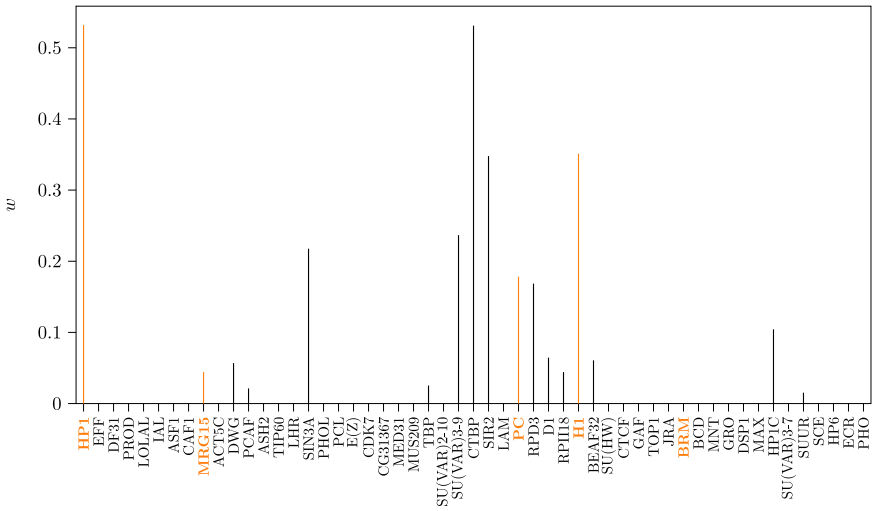

Fig. 7. Feature selection on the protein data set using the sparse jump model. The five marker proteins selected by Filion et al. (2010) are highlighted for comparison.

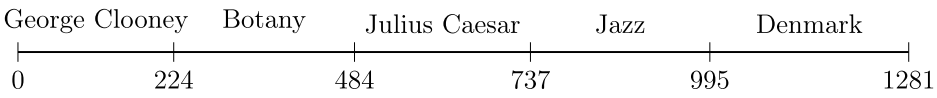

Fig. 8. Breakpoints between Wikipedia articles along with the topic of each article.

Table 4
NLP results based on 100 repetitions (only 10 for GGS) for each number of features P. The reported values are the mean (and standard deviation) of the absolute offset between the identified breakpoints and the true breakpoints.

	$P = 300$	$P = 600$	$P = 1200$	$P = 2400$
GGS	3.5 (0.0)	23 (5.2)	28 (2.5)	–
Jump	35 (81)	83 (143)	144 (185)	178 (182)
Sparse jump	0.48 (7.3)	0.39 (7.0)	0.0 (0.0)	3.5 (27)

task. Unlike jump models, the purpose of GGS is not to identify recurring states. Even though the methods are distinct and solve different problems, it is still of interest to compare their performance on a common task.

Hyperparameters. Using GGS with covariance-regularization parameter $\lambda = 10^{-3}$, the data is split into five segments every time. Unlike GGS, the jump model does not guarantee a certain number of states or break points. Instead, we fit jump models with different values of the jump penalty and select a value that yields the desired number of break points most of the time. In this way we choose the values $\lambda = 400$ for the jump model, and $\lambda = 30$ and $\kappa = 17$ for the sparse jump model, which we use for all values of P. If the jump model returns more or less than four break points, we simply fit it again from 10 new initializations of the state sequence, until we get the correct number of break points.

Accuracy. Table 4 reports the mean (and standard deviation) of the absolute offset between the identified breakpoints and the true breakpoints based on 100 repetitions (only 10 for GGS) for each value of P. For the $P = 300$ true features, GGS returns the same breakpoints every time, which on average deviate 3.5 words from the true breakpoints. The standard jump model is not as good at identifying the breakpoints with an average absolute offset of 35 words. Meanwhile, the sparse jump model is better than GGS with an average absolute offset of only 0.48 words. This shows the importance of feature selection in a high-dimensional setting, without adding any random features. The sparse jump model is able to weight the features in a way that accentuates the differences between the segments, which enables it to better detect the breakpoints.

Computation time. The sparse jump model is better than GGS at identifying the breakpoints and it is a lot faster. When $P = 300$, on average, it takes two minutes and 18 s to divide the data into five segments using GGS, compared to just 6.6 s using the sparse jump model. When $P = 600$, running GGS takes one hour and two minutes, compared to just 7.4 s to fit a sparse jump model. This is why the reported values in Table 4 are based on only 10 repetitions for each value of P for GGS. The overall time complexity of GGS is $O(TP^3K)$, which limits the dimensions of feature spaces it can be applied to Hallac et al. (2019). On the contrary, as discussed above, the sparse jump model scales well to large numbers of features.

Random features. When adding randomly permuted features to the data set, the accuracy of GGS and the jump model declines, whereas that of the sparse jump model is more or less unaffected. Even with $P = 2400$, i.e., twice as many features as observations, the average absolute offset of the breakpoints identified using the sparse jump model is only 3.5 words, which is the same as the accuracy of GGS on the $P = 300$ true features. We do not report results for GGS for $P = 2400$ because they take many hours to compute.

9. Conclusion

We proposed a general framework for joint feature selection, parameter and state-sequence estimation in jump models. With our framework it is possible to determine which features are responsible for the differences between the states by extracting information embedded in the ordering of the data. We developed and implemented a coordinate descent algorithm that alternates between selecting features and estimating model parameters and state sequence, which scales well to high-dimensional data sets with large numbers of noisy features.

Through experiments on two simulated and three real data sets, we showed that the sparse jump model is remarkably robust to noise and is able to distinguish relevant from irrelevant features. As a result, it yielded both more accurate and more persistent state estimates compared to a number of other methods.

The application to financial returns showed that jump models can be successfully applied to data where the information separating the states comes from the second moment. Moreover, the application showed

the ability of the sparse jump model to distinguish true features from random permutations.

The clustering of protein sequences emphasized the value of feature selection. It is a strength of the sparse jump model that without any biological knowledge, it was able to select meaningful features in an automated and systematic fashion.

The Wikipedia text example clearly showed the importance of feature selection for the jump model's ability to detect the breakpoints between the articles, both in absence and presence of randomly permuted features. At the same time, the example left no doubt that the sparse jump model is computationally advantageous.

In future work we plan to explore new applications of the sparse jump model as well as feature selection in other kinds of jump models. Future work should also explore the possibility to improve the accuracy of the sparse jump model through more systematic optimization of its hyperparameters, by using alternative initialization techniques, or by considering other distance measures.

A Python implementation of the sparse jump model proposed in this article is available online as supplementary material.

CRediT authorship contribution statement

Peter Nystrup: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Petter N. Kolm:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Project administration, Resources, Supervision, Validation, Writing – review & editing. **Erik Lindström:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Project administration, Resources, 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.

Acknowledgments

This work was supported by Vergstiftelsen and the Centre for IT-Intelligent Energy Systems (CITIES) project funded in part by Innovation Fund Denmark under Grant No. 1305-00027B. We thank Pierre Pinson and Tobias Rydén for helpful comments and Guillaume Filion for answering questions about clustering of chromatin proteins.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2021.115558.

References

Adams, S., & Beling, P. A. (2017). A survey of feature selection methods for Gaussian mixture models and hidden Markov models. *Artificial Intelligence Review*, *52*(3), 1739–1779.

Adams, S., Beling, P. A., & Cogill, R. (2016). Feature Selection for Hidden Markov Models and Hidden Semi-Markov Models. *IEEE Access*, *4*, 1642–1657.

de Amorim, R. C. (2016). A Survey on Feature Weighting Based *K*-Means Algorithms. *Journal of Classification*, *33*(2), 210–242.

de Amorim, R. C. (2019). Unsupervised feature selection for large data sets. *Pattern Recognition Letters*, *128*, 183–189.

Ang, A., & Timmermann, A. (2012). Regime Changes and Financial Markets. *Annual Review of Financial Economics*, *4*(1), 313–337.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM–SIAM symposium on discrete algorithms* (pp. 1027–1035).

Bakis, R. (1976). Continuous speech recognition via centisecond acoustic states. *Journal of the Acoustical Society of America*, *59*(S1), S97.

Bebbington, M. S. (2007). Identifying volcanic regimes using hidden Markov models. *Geophysical Journal International*, *171*(2), 921–942.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bemporad, A., Breschi, V., Piga, D., & Boyd, S. (2018). Fitting jump models. *Automatica*, *96*, 11–21.

Bhardwaj, S., Sharma, V., Srivastava, S., Sastry, O., Bandyopadhyay, B., Chandel, S., & Gupta, J. (2013). Estimation of solar radiation using a combination of Hidden Markov Model and generalized Fuzzy model. *Solar Energy*, *93*, 43–54.

Bottou, L., & Bengio, Y. (1995). Convergence properties of the K-means algorithms. In *Advances in neural information processing systems* (pp. 585–592).

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. New York: Cambridge University Press.

Breschi, V., Bemporad, A., Piga, D., & Boyd, S. (2018). Prediction error methods in learning jump ARMAX models. In *IEEE conference on decision and control* (pp. 2247–2252).

Brodersen, K. H., Ong, C. S., Stephan, K. E., & Buhmann, J. M. (2010). The Balanced Accuracy and Its Posterior Distribution. In *20th international conference on pattern recognition* (pp. 3121–3124).

Bulla, J. (2011). Hidden Markov models with t components. Increased persistence and other aspects. *Quantitative Finance*, *11*(3), 459–475.

Bulla, J., & Berzel, A. (2007). Computational issues in parameter estimation for stationary hidden Markov models. *Computational Statistics*, *23*(1), 1–18.

Celebi, M. E., Kingravi, H. A., & Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, *40*(1), 200–210.

Chang, W.-C. (1983). On using principal components before separating a mixture of two multivariate normal distributions. *Journal of the Royal Statistical Society. Series C. Applied Statistics*, *32*(3), 267–275.

Choo, K. H., Tong, J. C., & Zhang, L. (2004). Recent Applications of Hidden Markov Models in Computational Biology. *Genomics, Proteomics & Bioinformatics*, *2*(2), 84–96.

Cont, R. (2001). Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, *1*(2), 223–236.

Dias, J. G., Vermunt, J. K., & Ramos, S. B. (2015). Clustering financial time series: New insights from an extended hidden Markov model. *European Journal of Operational Research*, *243*(3), 852–864.

Dy, J. G., & Brodley, C. E. (2004). Feature selection for unsupervised learning. *Journal of Machine Learning Research*, *5*, 845–889.

Fiecas, M., Franke, J., von Sachs, R., & Kamgaing, J. T. (2017). Shrinkage estimation for multivariate hidden Markov models. *Journal of the American Statistical Association*, *112*(517), 424–435.

Filion, G. J., van Bemmel, J. G., Braunschweig, U., Talhout, W., Kind, J., Ward, L. D., Brugman, W., de Castro, I. J., Kerkhoven, R. M., Bussemaker, H. J., & van Steensel, B. (2010). Systematic Protein Location Mapping Reveals Five Principal Chromatin Types in Drosophila Cells. *Cell*, *143*(2), 212–224.

Fox, E. B., Sudderth, E. B., Jordan, M. I., & Willsky, A. S. (2011). A sticky HDP-HMM with application to speaker diarization. *Annals of Applied Statistics*, *5*(2A), 1020–1056.

Fränti, P., & Sieranoja, S. (2019). How much can k-means be improved by using better initialization and repeats?. *Pattern Recognition*, *93*, 95–112.

Gales, M., & Young, S. (2008). The application of hidden Markov models in speech recognition. *Foundations and Trends in Signal Processing*, *1*(3), 195–304.

Georgoulas, G., Mustafa, M., Tsoumas, I., Antonino-Daviu, J., Climente-Alarcon, V., Stylios, C., & Nikolakopoulos, G. (2013). Principal Component Analysis of the start-up transient and Hidden Markov Modeling for broken rotor bar fault diagnosis in asynchronous machines. *Expert Systems with Applications*, *40*(17), 7024–7033.

Hallac, D., Nystrup, P., & Boyd, S. (2019). Greedy Gaussian Segmentation of Multivariate Time Series. *Advances in Data Analysis and Classification*, *13*(3), 727–751.

He, X., Cai, D., & Niyogi, P. (2005). Laplacian score for feature selection. In *Advances in neural information processing systems* (pp. 507–514).

Huang, X., Acero, A., & Hon, H.-W. (2001). *Spoken language processing: a guide to theory, algorithm and system development*. New Jersey: Prentice–Hall.

Jamshidian, M., & Jennrich, R. I. (1997). Acceleration of the EM Algorithm by using Quasi-Newton Methods. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *59*(3), 569–587.

Kang, M., Ahn, J., & Lee, K. (2018). Opinion mining using ensemble text hidden Markov models for text classification. *Expert Systems with Applications*, *94*, 218–227.

Katz, I., & Crammer, K. (2015). Outlier-robust convex segmentation. In *Twenty-ninth AAAI conference on artificial intelligence*.

Kim, S.-J., Koh, K., Boyd, S., & Gorinevsky, D. (2009). ℓ_1 Trend Filtering. *SIAM Review*, *51*(2), 339–360.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129–137.

Maruotti, A., & Punzo, A. (2021). Initialization of Hidden Markov and Semi-Markov Models: A Critical Evaluation of Several Strategies. *International Statistical Review*, http://dx.doi.org/10.1111/insr.12436.

Mitra, P., Murthy, C., & Pal, S. (2002). Unsupervised feature selection using feature similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(3), 301–312.

Nefian, A. V., & Hayes, M. H. (1998). Hidden Markov models for face recognition. In *Proceedings of the 1998 IEEE international conference on acoustics, speech and signal processing, Vol. 5* (pp. 2721–2724). IEEE.

Netzer, O., Ebbes, P., & Bijmolt, T. H. A. (2017). Hidden Markov Models in Marketing. In *International series in quantitative marketing* (pp. 405–449). Cham: Springer.

Nystrup, P., Boyd, S., Lindström, E., & Madsen, H. (2019). Multi-Period Portfolio Selection with Drawdown Control. *Annals of Operations Research*, *282*(1–2), 245–271.

Nystrup, P., Hansen, B. W., Larsen, H. O., Madsen, H., & Lindström, E. (2017). Dynamic Allocation or Diversification: A Regime-Based Approach to Multiple Assets. *Journal of Portfolio Management*, *44*(2), 62–73.

Nystrup, P., Hansen, B. W., Madsen, H., & Lindström, E. (2015). Regime-Based Versus Static Asset Allocation: Letting the Data Speak. *Journal of Portfolio Management*, *42*(1), 103–109.

Nystrup, P., Hansen, B. W., Madsen, H., & Lindström, E. (2016). Detecting change points in VIX and S&P 500: A new approach to dynamic asset allocation. *Journal of Asset Management*, *17*(5), 361–374.

Nystrup, P., Kolm, P. N., & Lindström, E. (2020). Greedy Online Classification of Persistent Market States Using Realized Intraday Volatility Features. *Journal of Financial Data Science*, *2*(3), 25–39.

Nystrup, P., Lindström, E., & Madsen, H. (2020a). Hyperparameter Optimization for Portfolio Selection. *Journal of Financial Data Science*, *2*(3), 40–54.

Nystrup, P., Lindström, E., & Madsen, H. (2020b). Learning hidden Markov models with persistent states by penalizing jumps. *Expert Systems with Applications*, *150*, Article 113307.

Nystrup, P., Madsen, H., & Lindström, E. (2015). Stylised facts of financial time series and hidden Markov models in continuous time. *Quantitative Finance*, *15*(9), 1531–1541.

Nystrup, P., Madsen, H., & Lindström, E. (2017). Long Memory of Financial Time Series and Hidden Markov Models with Time-Varying Parameters. *Journal of Forecasting*, *36*(8), 989–1002.

Nystrup, P., Madsen, H., & Lindström, E. (2018). Dynamic portfolio optimization across hidden market regimes. *Quantitative Finance*, *18*(1), 83–95.

Oh, K. J., & Han, I. (2000). Using change-point detection to support artificial neural networks for interest rates forecasting. *Expert Systems with Applications*, *19*(2), 105–115.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Petropoulos, A., Chatzis, S. P., & Xanthopoulos, S. (2016). A novel corporate credit rating system based on Student's-t hidden Markov models. *Expert Systems with Applications*, *53*, 87–105.

Pinson, P., Christensen, L., Madsen, H., Sørensen, P., Donovan, M., & Jensen, L. (2008). Regime-switching modelling of the fluctuations of offshore wind generation. *Journal of Wind Engineering and Industrial Aerodynamics*, *96*(12), 2327–2347.

Robinson, W. N., & Aria, A. (2018). Sequential fraud detection for prepaid cards using hidden Markov model divergence. *Expert Systems with Applications*, *91*, 235–251.

Ross, G. J., Tasoulis, D. K., & Adams, N. M. (2011). Nonparametric Monitoring of Data Streams for Changes in Location and Scale. *Technometrics*, *53*(4), 379–389.

Tang, C., Liu, X., Zhu, X., Xiong, J., Li, M., Xia, J., Wang, X., & Wang, L. (2020). Feature Selective Projection with Low-Rank Embedding and Dual Laplacian Regularization. *IEEE Transactions on Knowledge and Data Engineering*, *32*(9), 1747–1760.

Tang, C., Zheng, X., Liu, X., Zhang, W., Zhang, J., Xiong, J., & Wang, L. (2021). Cross-view Locality Preserved Diversity and Consensus Learning for Multi-view Unsupervised Feature Selection. *IEEE Transactions on Knowledge and Data Engineering*.

Viterbi, A. J. (1967). Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260–269.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, *1*(6), 80.

Witten, D. M., & Tibshirani, R. (2010). A Framework for Feature Selection in Clustering. *Journal of the American Statistical Association*, *105*(490), 713–726.

Yao, Y., Cao, Y., Zhai, J., Liu, J., Xiang, M., & Wang, L. (2020). Latent state recognition by an enhanced hidden Markov model. *Expert Systems with Applications*, *161*, Article 113722.

Zhao, Z., & Liu, H. (2007). Spectral feature selection for supervised and unsupervised learning. In *Proceedings of the 24th international conference on machine learning* (pp. 1151–1157).

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *67*(2), 301–320.