Robust Statistical Jump Models with Feature Selection

Jonatan Persson

Master's thesis
2023:E54

Lund University

Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Abstract: A large area in statistics and machine learning is cluster analysis. This field of research concerns the design of algorithms that allow computers to automatically categorize a set of observations into different groups in a reasonable way, without any prior information about which observations belongs to which group. It is a part of the larger field of unsupervised learning within machine learning.

Many of these algorithms are designed with a specific problem-task in mind. One example are so-called statistical jump models, developed by Bemporad et al. [2018] and further by Nystrup, Kolm, et al. [2021], that are developed to be used for (amongst other things, they are quite flexible models) the clustering of time-series data.

In this thesis we have made a modification of these jump models that allows us to more freely chose how to measure distance between different observations. This opens up the possibility of designing more cluster algorithms for time series data that are more resilient to data containing many outliers, or doing clustering for categorical time series data.

Acknowledgments: I want to thank my supervisor Erik Lindström for introducing me to this project and for his encouragement and guidance. He has been an invaluable help in terms of giving me advice on what to investigate next and as a "bollplank" to bounce ideas off of.

Contents

1 Introduction

A well-studied set of problems in statistics are those of cluster analysis. In such problems, we are interested in developing methods that enable us to automatically differentiate between groups in a set of observations. An example could be if we are given information about different countries, such as their population, GDP, demographics, political systems, etc., we might want to categorize them into different groups based on how similar they are. For example, we might end up grouping them into developing or industrialized nations. Another grouping could be democratic, semi-democratic or authoritarian countries. For a human, such manual categorizations might seem trivial, but automating such tasks using computers is not.

1.1 Background

Several different approaches to find solutions to the clustering problem exists. One of the most common ones is the so-called K-means algorithm, which was first developed by Llyod [1982] in the context of signal processing. The algorithm is therefore sometimes referred to as Lloyd's algorithm. Several other methods exist, such as Gaussian mixture models with expectation-maximization or hierarchical clustering. More on these methods can be found in Hastie et al. [2017].

One issue with the clustering methods mentioned above, is that they do not work well for time series data, i.e. where there is dependence between the current observation and the observations that came before it. Attempts at reconciling clustering algorithms with time series data has been made by, e.g., Bemporad et al. [2018]. The models presented in that article are referred to as statistical jump models, and are very flexible and can be used outside of cluster analysis. One example is in Piga et al. [2020], where they fit the typical Box-Jenkins time series model into the jump model framework. Another example is the famous hidden Markov model, first developed by Baum and Petrie [1966], which can also be seen as a special case of the jump model.

In addition, Nystrup, Kolm, et al. [2021] have managed to combine the jump model with a feature selection framework originally developed by Witten and Tibshirani [2010]. This is useful when we, for example, have more features than observations, or in cases where we are not sure which features or combination of features are relevant when it comes to determining our clusters.

1.2 Purpose of Thesis

A problem with the jump models based on K-means is that they only allow for squared Euclidean distances to be considered by the model. This means that the model lacks robustness against potential outliers in the data set and that it can only consider numerical time series data. The purpose of this thesis project is to develop a jump model based on K-medoids clustering instead. K-medoids, unlike K-means, is very flexible in the choice of distance metric used, while still being computationally feasible to implement. This means that such a model could enable us to clustering of categorical time series data, or allow for more robust clustering results when outliers are present. Our new, medoids-based jump model is presented in section 4.2.

Also, we present how this new medoids-based jump model can be used in the previously mentioned feature selection framework from Witten and Tibshirani [2010]. We apply our method with a more robust way of measuring distance to simulated data with varying amounts of outliers, and compare it to the performance of the jump model based on K-means.

2 Linear Time Series Modelling

It is typical for basic courses in statistics to only consider the setting where we are dealing with *i.i.d.* observations drawn from some given distribution. However, such data is often not available to us in real-life scenarios. Instead, we have to rely on data where there is some temporal and/or spatial correlation between samples. For these cases we need statistical models that can effectively capture these correlation structures and in this section we will discuss some of these models for the case with temporal correlation. To do this, we will first have to define and briefly discuss stochastic processes. The entire section will be based on material from Jakobsson [2019].

2.1 Stationary Processes and ARMA models

A stochastic process is defined as an indexed collection of random variables $\{y_t; t \in \mathcal{T}\}$. While the standard *i.i.d* setting is an example of a very particular stochastic process, we will instead focus on processes that have some form of correlation structure between observations. In particular, we will study stochastic processes that fulfill the condition of wide-sense stationarity (WSS). A stochastic process $\{y_t; t \in \mathcal{T}\}$ is said to be WSS if the following three conditions are fulfilled:

1. $\mathrm{E}(y_t) = -\infty < m < \infty, \quad \forall t \in \mathcal{T}$.
2. $\mathrm{C}(y_t, y_s)$ can be written as a function of $(s - t)$.
3. $\mathrm{E}(|y_t|^2) < \infty$.

A stochastic process fulfilling the WSS conditions ensures that the distributional properties of the process does not depend on t.

Assuming that we have made T observations of the process, the mean of the process m can be estimated by

$$\hat{m}_T = \frac{1}{T} \sum_{t=1}^{T} y_t.$$

This estimator is unbiased and if it also consistent, i.e. $\hat{m}_T \xrightarrow{p} m$ as $T \to \infty$, we say that the process is ergodic in the mean.

The second WSS condition means that we can write the autocovariance as a function $r(k) = \mathrm{C}(y_t, y_{t-k})$ that only depends on k, not t. This autocovariance function will be symmetric, non-negative and is maximized at $k = 0$

for scalar processes. We usually estimate the autocovariance function by

$$\hat{r}_T(k) = \frac{1}{T} \sum_{t=k+1}^{T} (y_t - \hat{m}_T)(y_{t-k} - \hat{m}_T).$$

This estimate is actually not unbiased, but it is asymptotically unbiased as $T \to \infty$ if we assume that the process has mean zero. The reason that we still use this estimate is that the unbiased estimate will have increasing variance for increasing k, while the variance of the biased estimate is constant w.r.t. k.

We have now discussed the very basics of stationary processes and their descriptive statistics. However, we are most often interested in constructing an interpretable model that also allows for prediction of future values. A very common method to achieve this is to treat our observations as the output of linear time-invariant (LTI) system. If we let x_t denote the inputs and y_t the output to such a system, we can write the output as

$$y_t = \sum_{k=-\infty}^{\infty} h_k x_{t-k}$$

where we refer to h_k as the impulse response of the system.

One of the most commonly used linear models is the autoregressive moving average (ARMA) model. If we let $\{e_t\}$ denote a white noise process with zero-mean and variance σ_e^2 and treat this as the input to our system, then the ARMA model can be defined through the output of the following LTI system

$$y_t = \frac{C(z)}{A(z)} e_t$$

where

$$C(z) = 1 + c_1 z^{-1} + \ldots + c_q z^{-q}$$

and

$$A(z) = 1 + a_1 z^{-1} + \ldots + a_p z^{-p}.$$

Here, z^{-1} denotes the lag operator, i.e. we have $z^{-1} x_t = x_{t-1}$. A more common way of writing the ARMA model is

$$y_t + a_1 y_{t-1} + \ldots + a_p y_{t-p} = e_t + c_1 e_{t-1} + \ldots + c_q e_{t-q}.$$

Such an ARMA model will be stationary if the roots of the polynomial $A(z)$ have absolute value less than one. If we have the same result for the polynomial $C(z)$, we say that the process is invertible.

Several extensions of the standard ARMA model exist. For example, if we suspect that the data is non-stationary with a (stochastic) trend, an appropriate model might be an autoregressive integrated moving average (ARIMA) model, defined by the system

$$A(z)(1-z^{-1})^d y_t = C(z)e_t.$$

It is also possible to include exogenous input variables, via the system

$$A(z)(1-z^{-1})^d y_t = B(z)x_{t-d} + C(z)e_t$$

where $B(z) = b_0 + b_1 z^{-1} + \ldots + b_s z^{-s}$. Such models are referred to as ARMAX models. If we are dealing with multivariate y_t, these models can be extended by treating $A(z)$ and $C(z)$ as matrix polynomials. Such multivariate processes are often referred to as vector ARMAX models (VARMAX).

2.2 Non-Stationarity and Dynamic Linear Models

In applications such as finance, we do not typically have access to stationary data. One possible solution to this kind of problem is to instead model the data using stochastic dynamic systems. This is often done by using a state-space representation of the system, where we have a measurement equation describing the behavior of the measured data, which in turn depends on some underlying, non-observed variables that follow a so-called state equation. In particular, if we assume that the system is LTI and the uncertainty follows a Gaussian distribution, we can write the state equation as

$$x_{t+1} = Ax_t + Bu_t + v_t$$

and the measurement equation as

$$y_t = Cx_t + w_t.$$

Here $\{v_t\}$ and $\{w_t\}$ are assumed to be zero-mean multivariate Guassian processes that are independent of each other, with covariance matrices R_v and R_w respectively. Assuming that x_t, y_t and u_t are n-dimensional, m-dimensional and k-dimensional row vectors respectively, then A, B and C are $n \times n$, $n \times k$ and $m \times n$ dimensional matrices respectively.

Given that the matrices A, B, C, R_v and R_w are known and do not have to be estimated, we can estimate the unknown states using the Kalman filter. This is a recursive algorithm that requires us to pick initial "prediction" of x_1 as

$$\mathrm{E}(x_1) = \hat{x}_{1|0}$$

and the uncertainty of this initial "prediction"

$$\mathrm{Var}(x_1) = V_{1|0}.$$

Then, given the a priori prediction $\hat{x}_{t|t-1}$ and $V_{t|t-1}$, we can update this prediction using all the of the observations available up until time t to get the a posteriori updates

$$\mathrm{E}(x_t|y_1, \ldots, y_t) = \hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t|t-1})$$

and

$$\mathrm{Var}(x_t|y_1, \ldots, y_t) = V_{t|t} = (I - K_t C)V_{t|t-1}$$

Using this estimate, we can get a new a priori prediction by

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t} + Bu_t$$

and

$$V_{t+1|t} = AV_{t|t}A' + R_v.$$

Here, K_t denotes the Kalman gain, which is given by

$$K_t = V_{t|t-1}C'(CV_{t|t-1}C' + R_w)^{-1}.$$

All of these equations allow us to recursively estimate the unknown states.

The linear Gaussian state-space model presented above, allows for ARMAX processes of the form

$$A(z)y_t = B(z)u_t + C(z)e_t$$

with parameters that vary over time. We can do this by writing the model on state space form as

$$x_{t+1} = x_t + v_t$$
$$y_t = \bar{C}_t x_t + e_t$$

where v_t is white noise. The state vector x_t represents the model parameters given observations up until time t

$$x_t = [a_1 \ \ldots \ a_p \quad c_1 \ \ldots \ c_q \quad b_0 \ \ldots \ b_s]'$$

and the row vector $\bar{C}_t$ contains the given observations and noise realizations up until time t

$$\bar{C}_t = [-y_{t-1} \ \ldots \ -y_{t-p} \quad e_{t-1} \ \ldots \ e_{t-q} \quad u_t \ \ldots \ u_{t-s}].$$

Then it follows trivially how we can recursively form estimates of the time varying parameters by using the Kalman filter.

3 Regime-Switching Models

Often, we are interested in working with more complex time series data that cannot be modelled well by the linear time series models presented in section 2. Thus there is a need to develop alternative approaches, that are better able to capture non-linear complexities in the data. A common class of such non-linear models are regime-switching models. They assume that the data can be described by a finite number of different states (or regimes), each state using different time series dynamics. The jump models we will be working with are an extension of clustering algorithms, a commonly used unsupervised learning method in machine learning. We will begin by discussing a commonly used regime-switching model, the Hidden Markov Model (HMM). Then, we will show how this framework can be generalized using so-called jump models. In the final part of this section we will discuss how feature selection methods initially developed for the unsupervised learning setting in machine learning can be used to do feature selection in the previously mentioned jump model framework.

3.1 Hidden Markov Models

A relatively simple yet flexible framework (and also one of the most widely used ones) for regime-switching models is to the let the state transitions be governed by an unobservable finite-state Markov chain. Such models are referred to as HMMs and were first developed by Baum in a series of articles, the first of which being Baum and Petrie [1966]. These models have been used in a large variety of applications, of which one of the first was in the area of speech recognition (see e.g. Baker [1975]), but they have also been successfully used in, for example, bioinformatics (see e.g. Li and Stephens [2003]) and quantitative finance (see e.g. Sipos et al. [2017]).

For our illustration of HMMs we will rely on the tutorial by Rabiner [1989]. A HMM is defined by the following five properties, given a total of T observations $y_1, \ldots, y_T$ taking on discrete values from the set $\{1, \ldots, L\}$ and samples from a discrete time Markov chain $\mathbf{s} = (s_1, \ldots, s_T)$ take values in the set $\{1, \ldots, K\}$:

1. The (discrete) probability distribution of the initial state, $\pi_0 = \{\pi_{0,i}\}$. The initial probabilities are given by

$$\pi_{0,i} = P(s_1 = i), \quad i \in \{1, \ldots, K\}.$$

2. The transition matrix $\Pi = \{\pi_{i,j}\}$ of the Markov chain which governs

the state dynamics. The individual transition probabilities are

$$\pi_{i,j} = P(s_{t+1} = j | s_t = i), \quad i, j \in \{1, \ldots, K\}.$$

3. The emission probabilities for state $\boldsymbol{\beta} = \{\beta_j(v)\}$. The emission probabilities are written

$$\beta_j(v) = P(y_t = v | s_t = j, s_{t-1}, \ldots, s_1, y_{t-1}, \ldots, y_1)$$

$$= P(y_t = v | s_t = j), \quad j \in \{1, \ldots, K\} \text{ and } v \in \{1, \ldots, L\}.$$

The above equation for the emission probabilities illustrates an important detail when defining HMMs: that the current observation is only dependent on the current state, and not the previous states or observations.

Thus, we have that a HMM is completely defined by the set of parameters $\Theta = \{\Pi, \boldsymbol{\beta}, \pi_0\}$ (where K and L are defined via Π and $\boldsymbol{\beta}$).

Often times, we want to reconstruct the most likely state sequence given the HMM parameters Θ and observations $y_1, \ldots, y_T$ but with the actual state sequence missing. A common method for doing this is the algorithm by Viterbi [1967]. This algorithm is based on recursively computing the highest probability for a state sequence at time t with the current state being i, denoted

$$\delta_t(i) = \max_{s_1, s_2, \ldots, s_{t-1}} P(s_1, s_2, \ldots, s_t = i, y_1, y_2, \ldots, y_t | \Theta),$$

by noting that, since $\mathbf{s}$ is a Markov chain and only depends on the previous state, along with the fact that the current observation only depends on the current state, we can write

$$\delta_{t+1}(j) = \max_{s_1, s_2, \ldots, s_t} P(s_1, s_2, \ldots, s_{t+1} = j, y_1, y_2, \ldots, y_{t+1} | \Theta)$$

$$= \max_i \Big(\max_{s_1, s_2, \ldots, s_{t-1}} P(s_1, s_2, \ldots, s_t = i, y_1, y_2, \ldots, y_t | \Theta) P(s_{t+1} = j | s_t = i) \Big)$$

$$\times P(y_{t+1} | s_{t+1} = j) = \max_i \delta_t(i) \pi_{i,j} \beta_j(y_{t+1}).$$

If we then let $\phi = \operatorname{argmax}_i \delta_t(i) \pi_{i,j} \beta_j(y_{t+1})$ and initialize the algorithm by letting $\delta_1(i) = \pi_{0,i} \beta_i(y_1)$ for every $1 \leq i \leq K$ and $\phi_1(i) = 0$, we can compute all of these δ's and ϕ's for $2 \leq t \leq T$ and $1 \leq j \leq K$ recursively and pick

$$s_T^* = \operatorname*{argmax}_{1 \leq i \leq K} \delta_T(i)$$

$$s_t^* = \phi_{t+1}(s_{t+1}^*), \quad t = T-1, \ldots, 1$$

to reconstruct the most likely state sequence given the observations.

There are several ways to extend the HMM defined above, e.g. letting the observations and states take on continuous values. In such a case we can instead use so called sequential Monte Carlo methods (also commonly referred to as particle filters) to retrieve our state sequence. Such models are described in detail in, for example, Chopin and Papaspiliopoulos [2020]. In the next section we will discuss a further, very general extension of the HMM (which also extends other regime-switching models), known as statistical jump model.

3.2 Statistical Jump Models

In Bemporad et al. [2018], a general framework for statistical jump models is presented. First, letting X denote the $T \times P_1$ input data matrix with rows x_t and Y the $T \times P_2$ output data matrix with rows y_t, we define the fitting objective as

$$J(X, Y, \Theta, \mathbf{s}) = \sum_{t=1}^{T} \ell(x_t, y_t, \theta_{s_t}) + \sum_{k=1}^{K} r(\theta_k) + \mathcal{L}(\mathbf{s})$$

where ℓ denotes the loss function, r is called the regularizer, the vector $\Theta = (\theta_1, \ldots, \theta_K)$ denotes the model parameters, the vector $\mathbf{s} = (s_1, \ldots, s_T)$ denotes the state sequence taking values in the set $\{1, \ldots, K\}$ and $\mathcal{L}$ is called the state sequence loss. The sum of loss functions represents the regression part and indeed, if we picked the loss function to be the typical squared Euclidean loss, made the assumption that the output y_t is a scalar and modeled as a linear combination of the inputs, and that the regularizer and state loss are set to zero, we would end up with standard multiple linear regression. The regularizer r is typically picked to be the either the ridge and/or LASSO penalty term, i.e. letting $r(\theta) = \theta^2$ or $r(\theta) = |\theta|$ respectively. $\mathcal{L}$ is defined as

$$\mathcal{L}(\mathbf{s}) = \mathcal{L}^{\text{init}}(s_0) + \sum_{t=1}^{T} \mathcal{L}^{\text{state}}(s_t) + \sum_{t=1}^{T} \mathcal{L}^{\text{trans}}(s_t, s_{t-1}).$$

$\mathcal{L}^{\text{init}}$, $\mathcal{L}^{\text{state}}$ and $\mathcal{L}^{\text{trans}}$ are referred to as the initial state cost, state cost and state transition cost, respectively. The jump model is then obtained by minimizing J over the model parameters Θ and state sequence $\mathbf{s}$.

The actual fitting of the jump model is accomplished by iterating between optimizing J over the parameters Θ while keeping the state sequence $\mathbf{s}$ fixed,

and then optimizing it over $\mathbf{s}$ while keeping Θ fixed. The optimization over Θ can be accomplished using traditional convex optimization algorithms, assuming that both ℓ and r are convex functions of the parameters. For the optimization over the state sequence, we achieve a global optimum by letting V and U be matrices defined by

$$V(s, T) = \mathcal{L}^{\text{state}}(s) + \ell(x_T,\, y_T,\, \theta_s)$$

$$U_{s,t} = \arg\min_j [V(j, t+1) + \mathcal{L}^{\text{trans}}(j,\, s)]$$

$$V(s,\, t) = \mathcal{L}^{\text{state}}(s) + \ell(x_t,\, y_t,\, \theta_s) + V(U_{s,t},\, t+1) + \mathcal{L}^{\text{trans}}(U_{s,t},\, s)$$

$$V(s,\, 0) = \mathcal{L}^{\text{init}}(s) + \min_j [V(j,\, 1) + \mathcal{L}^{\text{trans}}(j,\, s)]$$

which can be computed backwards, starting from T. From these equations it is clear that the matrix V will consist of the relevant parts of the fitting objective to be minimized (since we are not minimizing over the parameters we can ignore the regularization term) and the matrix U consists of minmizing indices. The estimated state sequence is then given by

$$s_T = \arg\min_j V(j, 0)$$

$$s_t = U_{s_{t-1},t},\, t = 1,\, \ldots,\, T-1.$$

The entire iterative optimization algorithm will find at least a local optimum using just a finite number of iterations. However the optimization is dependent on the initial state sequence chosen, so we might potentially end up with a lower value for the fitting objective J if we run with a different initialization. Therefore, it is common to initialize the algorithm with multiple different $\mathbf{s}$ and then picking the resulting parameters out of these runs that results in the smallest J.

The Viterbi algorithm discussed in section 3.1 is the same as the above iterative fitting procedure for jump models, but with the entries of the matrices M and U being computed forwards in time instead of backwards. This is not the only connection between jump models and the HMM framework. In fact, Bemporad et al. [2018] showed that if we let

$$\ell(x,\, y,\, \theta_s) = -\log(\beta_{\theta_s}(y)), \quad r(\theta) = 0$$

$$\mathcal{L}^{\text{trans}}(s_t,\, s_{t-1}) = -\log\pi_{s_t,s_{t-1}}, \quad \mathcal{L}^{\text{init}}(s_0) = -\log\pi_{0,s_0}, \quad \mathcal{L}^{\text{state}}(\mathbf{s}) = 0$$

with $x_t = 1$ and $\theta_s = s$, in the above jump model framework, we retrieve the HMM. This illustrates how much more flexible jump models are compared to HMMs, where we are restricted to state sequences with Markovian dynamics

and discrete state space, just to name a few restrictions which we do not have in the jump model case.

To further illustrate the flexibility of jump models we will discuss how they can be used for clustering in an unsupervised learning setting. In cluster analysis, we are interested in assigning a label out of a finite number of different labels to each data point in a data set, where all the data points are unlabeled to begin with. To do this, the clustering algorithm will assign the same label to data points that are closer to each other than they are to other data points, using some given distance measure. Thus, the clustering algorithm will assign the same label to "clusters" of the data, hence the name. There are many algorithms that try to solve this problem, see for example Hastie et al. [2017] for a list of some of the most common ones.

We will have a closer look at one specific, common clustering algorithm, the K-means algorithm. For this algorithm, each observation is assigned one out of K labels, where K is given. This is achieved by minimizing the within-cluster sum of squares (WCSS)

$$\sum_{t=1}^{T} \|y_t - \bar{y}_{s_t}\|_2^2$$

over the s_t's, which take values in the set $\{1, \ldots, K\}$ and denotes which cluster observation t belongs to. $\bar{y}_k$ denotes the mean of all the observations belonging to cluster k. By noting the following relation between the total sum of squares (TSS), WCSS and between-cluster sum of squares (BCSS)

$$\sum_{t=1}^{T} \|y_t - \bar{y}\|_2^2 = \sum_{t=1}^{T} \|y_t - \bar{y}_{s_t}\|_2^2 + \sum_{k=1}^{K} n_k \|\bar{y}_k - \bar{y}\|_2^2 \tag{1}$$

we can achieve K-means clustering by instead maximizing the BCSS

$$\sum_{k=1}^{K} n_k \|\bar{y}_k - \bar{y}\|_2^2$$

where n_k is the total number of observations belonging to cluster k and $\bar{y}$ is the mean of the observations. We can reconcile the jump model with the K-means algorithm by letting $x_t = y_t$ and choosing $\mathcal{L}(\mathbf{s}) = r(\theta_k) = 0$ for all $k \in \{1, \ldots, K\}$ and

$$\ell(x_t, y_t, \theta_{s_t}) = \ell(y_t, \theta_{s_t}) = \|y_t - \theta_{s_t}\|_2^2$$

where $\theta_k = \bar{y}_k$. Once again, this example illustrates the generality of the jump modeling framework.

A problem with most clustering algorithms, including K-means, is that they do not work well for data with sequential dependence. It is possible to amend this by viewing the clustering problem as a jump model (as we illustrated how to do for the K-means case in the above paragraph), but then letting the state sequence loss $\mathcal{L}$ be non-zero. For example, if we keep $\mathcal{L}^{\text{init}}(s_0) = \mathcal{L}^{\text{state}}(i) = 0$ for all $i \in \{1, \ldots, K\}$ as we did before, but now instead letting

$$\mathcal{L}^{\text{trans}}(i, j) = \lambda \cdot \mathbb{I}\{i \neq j \neq 0\}$$

where λ is called the jump penalty and is a hyperparameter of our choosing, it allows us to rewrite the fitting objective as

$$\sum_{t=1}^{T-1} \Big[\ell(y_t, \theta_{s_t}) + \lambda \mathbb{I}\{s_t \neq s_{t+1}\}\Big] + \ell(y_T, \theta_{s_T}). \tag{2}$$

where we, to facilitate for more general clustering algorithms, don't necessarily let the loss function ℓ be Euclidean distance from the cluster centre. The hyperparameter λ is referred to as the jump penalty because it will increase the value of the objective function to be minimized each time there is a switch of states, and thus it penalizes jumps. In this case, letting the jump penalty $\lambda \to \infty$, we end up with a model with constant state sequence, or a single cluster, since just a single jump will lead to a very large value for the objective function. On the other hand, if we let $\lambda \to 0$ we end up with a model that does not take the temporal correlation of the data into account, i.e. we end up with a pure clustering model. In fact, if we let ℓ be the Euclidean distance, we end up with the K-means algorithm. We have now established a connection between jump models and clustering algorithms. This can be used to allow for feature selection in such jump models.

3.3 Jump Models with Feature Selection

In many applications, the number of features P is large, potentially larger than the number of observations. For those cases it is often of interest to be able to identify the most important subset of features to use for the clustering. Such feature selection allows for more interpretable results and lower computational costs when including new observations in the model. Witten and Tibshirani [2010] presented such a feature selection framework for clustering algorithms, along with a general framework for clustering methods.

Assuming we have T observations of P features, we denote the $T \times P$ data matrix by Y and feature p by $\tilde{y}_p \in \mathbb{R}^T$. Several clustering algorithms (including the K-means algorithm) can be formulated in terms of maximizing a sum of transformations of the individual features, i.e. maximizing an

objective function of the following form

$$\sum_{p=1}^{P} f_p(\tilde{y}_p, \Theta) \tag{3}$$

over some parameter space, where f_p denotes a function of the p'th feature and parameters Θ. The idea of sparse clustering is to put different weights on the different features in such a way that some of the weights will be zero (and thus that feature becomes irrelevant to the clustering procedure). This can be achieved by modifying the above objective function to get

$$\sum_{p=1}^{P} w_p f_p(\tilde{y}_p, \Theta)$$

where we once again maximize over Θ, but also additionally over the weights $\mathbf{w}$. These weights are subject to the following constraints, which we will denote ($\star$):

(i) $\|\mathbf{w}\|_2^2 \leq 1$

(ii) $\|\mathbf{w}\|_1 \leq \kappa$

(iii) $w_p \geq 0, \quad \forall p$

where $\|\cdot\|_2$ refers to the L_2-norm, $\|\cdot\|_1$ refers to the L_1-norm and $1 \leq \kappa \leq \sqrt{P}$ is a tuning parameter of our choice. Due to the constraint $\|\mathbf{w}\|_1 \leq \kappa$, which is referred to as the LASSO penalty, some of the weights will be set to zero.

In practice, we perform the optimization by iterating between optimizing over possible Θ and then optimizing over the $\mathbf{w}$'s. By results presented in Boyd and Vandenberghe [2004], this latter optimization procedure with respect to $\mathbf{w}$ has an exact solution

$$\mathbf{w} = \frac{S(\mathbf{a}_+, \Delta)}{\|S(\mathbf{a}_+, \Delta)\|_2} \tag{4}$$

where S denotes the soft-thresholding operator given by $S(x, c) = \text{sign}(x)(|x| - c)_+$, $\mathbf{a}$ is a row vector of length P with entries $a_p = f_p(\tilde{y}_p, \Theta)$, $\mathbf{x}_+$ is the positive part of $\mathbf{x}$ and we pick $\Delta = 0$ if this gives $\|\mathbf{w}\|_1 < \kappa$, otherwise we pick $\Delta > 0$ such that $\|\mathbf{w}\|_1 = \kappa$. The property of the LASSO penalty that will set certain feature weights to exactly zero can be illustrated by plotting the soft-threshold function. We have done so in figure 1, from which notice that the function will be zero in the interval $[-c, c]$. This means that the weight for feature p will be set to zero if $\max\{a_p, 0\} \in [-\Delta, \Delta]$.

A feature selection algorithm for jump models is derived in Nystrup, Kolm, et al. [2021] by combining the sparse clustering framework presented just above and the final jump model presented in 3.2. This is done by first noting that if we let $\ell(y_t, \theta_{s_t}) = \|y_t - \theta_{s_t}\|_2^2$, then minimizing the fitting objective (2) is equivalent to maximizing the following expression

$$\sum_{k=1}^{K} n_k \|\theta_k - \bar{y}\|_2^2 - \lambda \sum_{t=1}^{T-1} \mathbb{I}\{s_t \neq s_{t+1}\}$$

where n_k is the number of observations in state k and $\bar{y}$ is the mean of the observations. This follows from relation (1). We can then fit this new fitting objective into the spare clustering framework by maximizing

$$\mathbf{w}' \sum_{k=1}^{K} n_k (\theta_k - \bar{y})^2 - \lambda \sum_{t=1}^{T-1} \mathbb{I}\{s_t \neq s_{t+1}\} \tag{5}$$

with respect to $\mathbf{w}$, $\theta_1, \ldots, \theta_K$ and $\mathbf{s}$, where the square is taken element-wise and $\mathbf{w}$ is subject to the restrictions ($\star$).

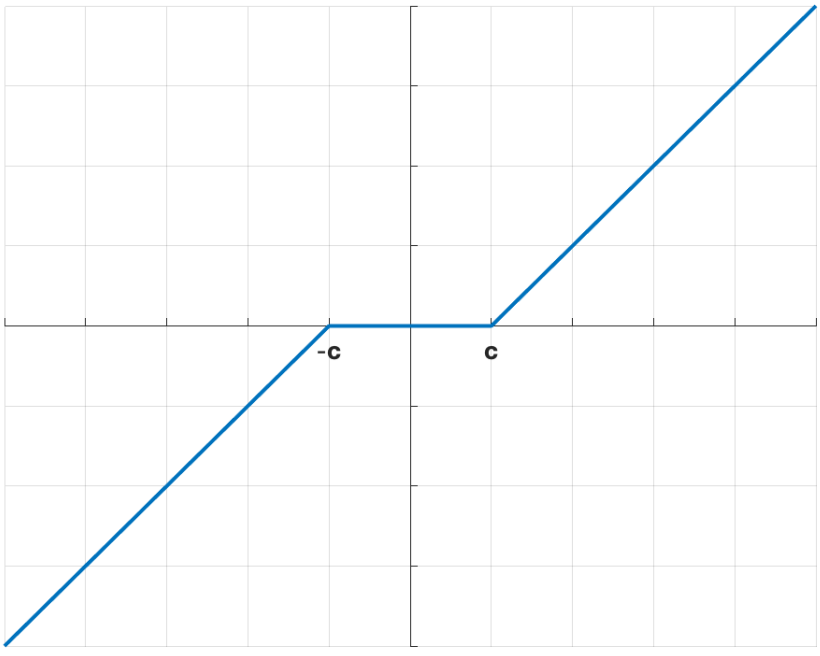

Figure 1: Plot of the soft-thresholding operator $S(x, c) = \text{sign}(x)(|x| - c)_+$ as a function of x.

Just like the sparse clustering algorithm in Witten and Tibshirani [2010], the maximization of the objective function (5) is done by iterating between optimizing (5) w.r.t. the model parameters $\theta_1, \ldots, \theta_K$ and $\mathbf{s}$, and then optimizing (5) w.r.t. the feature weights $\mathbf{w}$. Such an iterative approach is not jointly convex and thus we are not guaranteed to achieve a global optimum, but the objective function will increase at each step. Also, when optimizing over the weights, the solution is as given above of the form (4), where $\mathbf{a} = \sum_{k=1}^{K} n_k \|\theta_k - \bar{y}\|_2^2$. The entire sparse jump fitting procedure is summarized in Algorithm 1.

Note that in both the sparse clustering framework in Witten and Tibshirani [2010] and the sparse jump models in Nystrup, Kolm, et al. [2021], we assume that the features are standardized before initializing the algorithm. This is needed to ensure that the sizes of the weights are not based on the data having different scales, but only on the actual importance of each feature.

It is possible to pick λ and κ using some hyperparameter optimization routine, e.g. using grid search over some possible parameter values with some prespecified performance measure. How to pick such a performance measure follows very naturally for regression or classification problems where we have access to output data we can compare predictions to. However, in the case of clustering (and more generally in unsupervised learning) we do not have this and thus have to take more care in how to pick our performance measure. In Witten and Tibshirani [2010] they suggest using the so-called gap statistic as performance measure and in Nystrup, Kolm, et al. [2021] they suggest picking hyperparameters based on the specific applications using relevant domain knowledge. A newer approach is suggested by Cortese et al. [2023], which based on using a information criterion developed specifically for statistical jump models.

When fitting the jump model we need to provide a initial state sequence. We do this using an initialization strategy typically used for the regular K-means algorithm, called K-means++ developed by Arthur and Vassilvitskii [2007]. The development of this alternate initialization strategy was motivated by the fact that, while the standard K-means algorithm is very computationally efficient, random initialization of the algorithm can often lead to inappropriate clustering assignments that only results in a local optimum of the WCSS being found. The K-means++ initialization is done by first picking a cluster center at random amongst the available observations. The next cluster centre is picked amongst the remaining observations, where

observation y^* is picked with probability

$$\frac{\min_{\theta\in\Theta}\|y^* - \theta\|_2^2}{\sum_{t=1}^{T}\min_{\theta\in\Theta}\|y_t - \theta\|_2^2}$$

where Θ is the set of already picked centroids. We do this until we have K initial cluster centres, after which we can run the standard K-means algorithm. Thus the K-means++ initialization is also done randomly, but with weighted probabilities. Empirical results from Arthur and Vassilvitskii [2007] show that this initialization technique results in more accurate clustering results without sacrificing computational efficiency. In Nystrup, Kolm, et al. [2021] they initialize the jump model fitting 10 times each iteration, and let one of these initial state sequences be the best state sequence from the previous iteration, the others are obtained using K-means++. This results in a more robust model.

Algorithm 1 Fitting sparse jump model. (Nystrup, Kolm, et al. [2021])

Input: Standardized observations $y_1, \ldots, y_T$ of P features, number of hidden states K and tuning parameters λ, κ.

1. Initialize feature weights $\mathbf{w}$ by setting $w_1^0, \ldots, w_P^0 = 1/\sqrt{P}$.
2. For $i = 1, \ldots$ up until $\|\mathbf{w}^i - \mathbf{w}^{i-1}\|_1 / \|\mathbf{w}^{i-1}\|_1 < 10^{-4}$:
   (a) Compute weighted features $z_t = y_t \cdot \mathrm{diag}(\sqrt{\mathbf{w}^{i-1}})$, for $t = 1, \ldots, T$.
   (b) Initialize state sequence $\mathbf{s}$ as $\mathbf{s}^0 = (s_1^0, \ldots, s_T^0)$.
   (c) For $j = 1, \ldots$ up until $\mathbf{s}^j = \mathbf{s}^{j-1}$:
      i. Fit model parameters, $\boldsymbol{\mu}^j = \arg\min_{\mu} \sum_{t=1}^{T}\|z_t - \boldsymbol{\mu}_{s_t^{j-1}}\|^2$.
      ii. Fit state sequence, $\mathbf{s}^j = \arg\min_{\mathbf{s}}\Big\{ \sum_{t=1}^{T-1}\Big[\ell(\mathbf{z}_t, \boldsymbol{\mu}_{s_t}^j) + \lambda\mathbb{I}\{s_t \neq s_{t+1}\}\Big] + \ell(\mathbf{z}_T, \boldsymbol{\mu}_{s_T}^j)\Big\}$.
   (d) Update weights $\mathbf{w} = \frac{S(\mathbf{a}_+, \Delta)}{\|S(\mathbf{a}_+, \Delta)\|_2}$, where $\mathbf{a} = \sum_{k=1}^{K} n_k\|\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}}\|^2$.

Output: Model parameters $\boldsymbol{\mu}$, state sequence $\mathbf{s}$ and weights $\mathbf{w}$.

4 Robust Jump Models with Feature Selection

In most statistical settings we work under the assumption that the data is in some sense well-behaved. In e.g. parametric problems, this often means that the data is assumed to follow some prespecified probability distribution. However, in typical real-data scenarios the data often contains outlier measurements which could skew the empirical distribution of the data. This results in a degradation of the model performance and gives results that do not reflect what the distribution of the data would be with these outlier removed. One approach to such problems is to inspect the data to try to identify outlier observations and then remove them manually, but this is often a tedious process, especially if the data set at hand is large and/or multivariate.

Another method is to develop robust statistical models for the problem at hand. Robust models are specifically designed to be less sensitive to outliers in the data and often relaxes some of the assumptions made for the standard, non-robust version of the model. Examples of such robust models can be found in e.g. Huber and Ronchetti [2009]. In this section, we will try to develop a more robust variant of the sparse jump model presented in section 3.3 by using a clustering method that allows for flexible choice of dissimilarity measure.

4.1 K-Medoids

The jump model presented in section 3.2 was based on K-means clustering which used squared Euclidean distance to determine which observations belong to which cluster/state. Squared Euclidean distance will give a large dissimilarity value to outlier observations and could even create entire clusters that just contains such outliers. Thus, if we are interested in creating a more robust jump model, an obvious approach would be to replace the squared Euclidean distance with a dissimilarity measure that does not results in as large values for large distances. However, using squared Euclidean distance for the K-means setting has the advantage of the objective function being minimized by the cluster means. This ensures that the optimization over the parameters when fitting the jump model using the iterative approach is trivial. A different dissimilarity measure might not have such a useful property.

Before discussing more robust alternatives to regular K-means, we will define more exactly what we mean by dissimilarity measure. This discussion will be based on material from Hastie et al. [2017]. Denote the dissimilarity measure by $d(\,\cdot\,)$, where this is a function taking value from the set of

observation to the positive real line. Further, we will assume that $d(\,\cdot\,)$ is a combination of dissimilarities between individual features:

$$d(y_i,\, y_j) = \sum_{p=1}^{P} d_p(y_{ip},\, y_{jp})$$

where $d_p(\,\cdot\,)$ denotes the dissimilarity between feature p of the observations. We can further generalize this by weighting each of the individual feature dissimilarities

$$d(y_i,\, y_j) = \sum_{p=1}^{P} w_p d_p(y_{ip},\, y_{jp}).$$

If we want to make sure that each feature contributes equally to the total dissimilarity, we can let the weight for feature p be equal to that features contribution to the average pairwise dissimilarity

$$w_p = \Big(\frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T} d_p(y_{ip},\, y_{jp})\Big)^{-1}.$$

In the case where we use squared Euclidean distance as our dissimilarity measure, we get that

$$w_p = \Big(\frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T} (y_{ip} - y_{jp})^2\Big)^{-1} = \frac{1}{2\mathrm{Var}(\tilde{y}_p)}$$

where $\tilde{y}_p$ denotes the p'th feature vector. This means that in the case of using squared Euclidean distance, we can achieve equal contribution by simply standardizing each feature to get their z-scores.

The dissimilarity measure we have used thus far in this thesis is squared Euclidean distance (or squared L^2 norm), given by

$$d^2_{L^2}(y_i,\, y_j) = (y_{i1} - y_{j1})^2 + \ldots + (y_{iP} - y_{jP})^2 = \|y_i - y_j\|_2^2.$$

The square root of this value will give the squared length of the straight line drawn between the two points y_i and y_j. While Euclidean distance (either squared or not) is the most commonly used dissimilarity measure when dealing with numerical data, it is not always the most appropriate. A common alternative is the Manhattan distance (or L^1 norm), which we write as

$$d_{L^1}(y_i,\, y_j) = |y_{i1} - y_{j1}| + \ldots + |y_{iP} - y_{jP}| = \|y_i - y_j\|_1.$$

The difference between squared Euclidean and Manhattan distance can be illustrated by plotting them in the setting with a single feature, which we

have done in figure 2. From this figure, we notice that squared Euclidean distance assigns a smaller distance to observations that are close to each other compared to the Manhattan distance. On the other hand, when the observations are far apart, the squared terms in the squared Euclidean distance will result in much larger distance values being assigned compared to Manhattan distance. Thus, we draw the conclusion that the Manhattan distance is more robust to outliers, but might perform worse in situations where we have more "well-behaved" data when compared to squared Euclidean distance. The above mentioned dissimilarity measures are examples of so-called L^p norms, which can be written as

$$d_{L^p}(y_i, y_j) = (|y_{i1} - y_{j1}|^p + \ldots + |y_{iP} - y_{jP}|^p)^{1/p} = \|y_i - y_j\|_p.$$

So far, we have only discussed dissimilarity measure when the data is numeric. However, using the dissimilarities framework allows us to deal with categorical data as well. For example, given a data set where each observation has P features that in turn can take on M distinct, categorical values. Assume, for sake of simplicity, that each category is labeled by numbers from 1 to M. Then, a commonly used dissimilarity measure is

$$d_{L^0}(y_i, y_j) = \mathbb{I}\{y_{i1} \neq y_{j1}\} + \ldots + \mathbb{I}\{y_{iP} \neq y_{jP}\}.$$

In fact, we will retrieve the dissimilarity d_{L^0} by taking the limit of the L^p norm, as p goes to zero:

$$d_{L^0}(y_i, y_j) = \lim_{p \to 0} \|y_i - y_j\|_p.$$

This can be modified so that different differences between pair of categorical labels have different dissimilarity values. Additionally, if we are dealing with ranked categorical data, further modifications are necessary.

An alternative to K-means which allows for more flexible choice of dissimilarity measure is called K-medoids and is discussed in Hastie et al. [2017]. This method modifies the objective function to be minimized from WCSS used in standard K-means to

$$\sum_{t=1}^{T} d(y_t, y_{t_{s_t}})$$

where we now minimize over the state sequence $\mathbf{s} = \{s_1, \ldots, s_T\}$ and indices $\{t_i\}_{i=1}^K$. Here our cluster centres, or medoids, are restricted to be one of our observations, which is what allows for more flexibility when choosing dissimilarity measure. Thus, we could for example use Manhattan distance

if we suspect that there are outliers in the data, or the "L^0" norm if we have binary data.

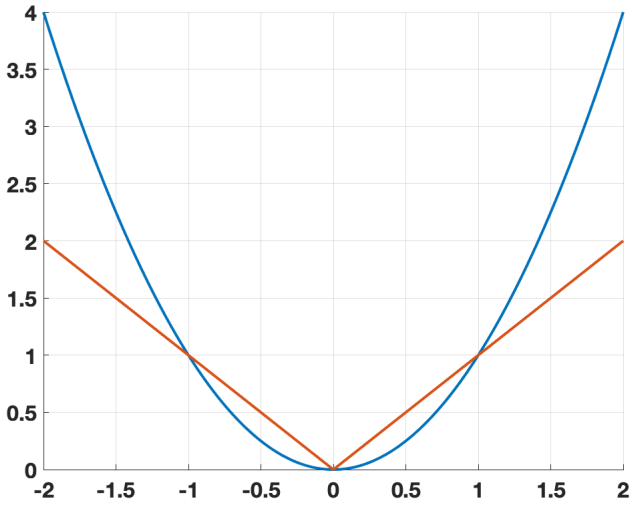

Figure 2: Blue graph is squared Euclidean distance from the origin in the one-dimensional setting. Red graph is Manhattan distance from the origin in the one-dimensional setting.

Optimizing this objective function can be achieved in several different ways. One such approach, referred to as Partitioning Around Medoids (PAM), was presented by Kaufman and Rousseeuw [1990]. This optimization algorithm starts by finding initial medoids using what is called the BUILD step. The first medoid is picked to result in the lowest value for the objective function, the second is picked amongst the remaining observations that results in the biggest reduction in the value of the objective function. This is repeated until we have found K initial medoids. Once the BUILD step is finished, PAM tries to improve on the initial selection of medoids by swapping between each medoid and the other non-medoid observations, evaluating the resulting objective function for each swap. PAM then updates the chosen medoids by picking the swap that results in the biggest reduction in the value of the objective function, continuing this process until there are no further swaps that results in a reduced value. This second part of the algorithm is referred to as the SWAP step.

Another, naive, heuristic method very similar to the K-means algorithm is based on iterating between minimizing the objective function holding $\mathbf{s}$ fixed and then minimizing it holding $\{t_i\}_{i=1}^K$ fixed. This method is presented in Park and Jun [2009], where they show that it performs comparably to

PAM. However, a different comparison between the heuristic iterative algorithm and PAM (along with other algorithms for the K-medoids problem) is done in Schubert and Rousseeuw [2021]. They state that the heuristic algorithm indeed is faster, but it is not as exhaustive as PAM and misses many potential medoids that could improve the objective function. Also, the heuristic algorithm has a tendency to get stuck at a local optimum. Thus, we conclude that for smaller data sets PAM is to be preferred, but if we are dealing with large data sets it might not be viable due to computational constraints and we would have to use the heuristic algorithm instead.

To illustrate the potential robustness property (depending on the dissimilarity measure we pick) of K-medoids, we will apply regular K-means and K-medoids with L_1-norm, using PAM, to a simulated data set of two-dimensional t-distributed data and compare how they perform. We simulate data from a mixture of three t-distributions with degrees-of-freedom $\upsilon = 2$ and the following three location vectors

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} -5 \\ -6 \end{pmatrix}, \quad \mu_3 = \begin{pmatrix} 6 \\ -5 \end{pmatrix}$$

and scale matrices

$$\rho_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \rho_2 = \begin{pmatrix} 1 & 0.13 \\ 0.13 & 1 \end{pmatrix}, \quad \rho_3 = \begin{pmatrix} 1 & 0.67 \\ 0.67 & 1 \end{pmatrix}.$$

We simulate 50 observations from each distribution, the results being visible in figure 3. Notice that there is an outlier observation from the distribution with μ_2 and ρ_2 in the lower right corner of the plot. Then, we cluster this data using 3-means and 3-medoids with L_1-norm, with the resulting clusters visible in figures 4 and 5, respectively. Here, 3-means performs quite poorly, devoting an entire cluster to just the outlier observation. Meanwhile, 3-medoids is much less sensitive to the outlier and manages to recreate the real clusters fairly well. This very simple example illustrates the importance of robustness of the statistical methods we use.

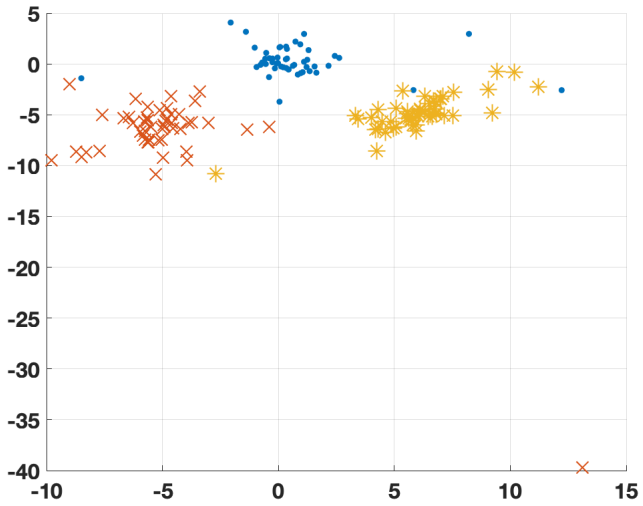

Figure 3: Simulated mixture of three different t-distributions. Points in blue belong to the distribution with mean μ_1 and correlations ρ_1, points in red belong to distribution with mean μ_2 and correlations ρ_2, points in green belong to distribution with mean μ_3 and correlations ρ_3.

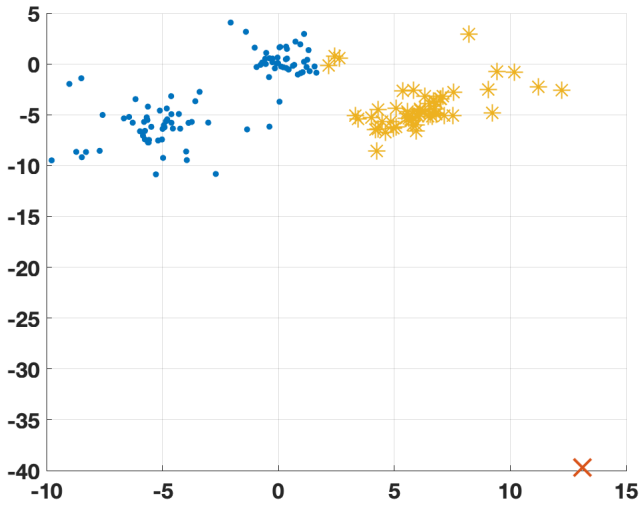

Figure 4: The data from figure 3 clustered using 3-means. Notice the single red observation in the lower right corner.

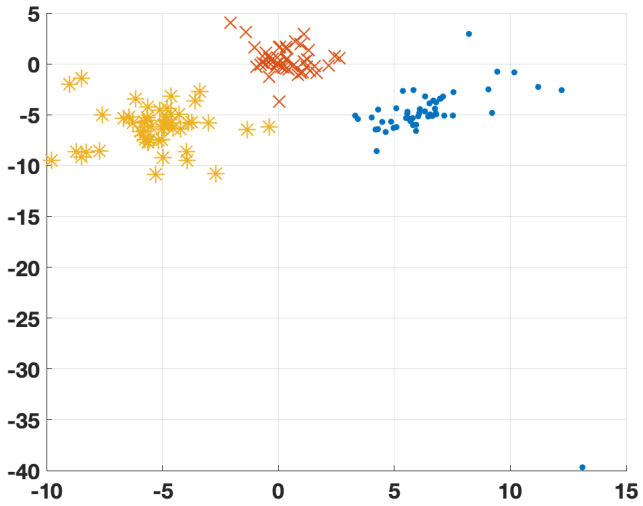

Figure 5: The data from figure 3 clustered using 3-medoids with L_1-norm and PAM.

4.2 Medoids-Based Jump Models

In section 3.2, we defined the jump model by minimizing the the fitting objective given by equation (2) over the cluster centroids and state sequence. In this setting we deliberately did not specify the loss function ℓ in an effort to allow for more flexible model choice. Thus, in an effort to arrive at a more robust jump model, we could use a loss function different from the squared Euclidean one that we used throughout section 3.3. However, optimizing fitting objective (2) over the parameters (keeping the state sequence fixed) is not always computationally feasible for every choice of loss function, and in some cases analytical solution (which we have in the case of the squared Euclidean distance) to this optimization problem might not even exist. In this subsection we will make our own contribution to the literature on statistical jump models by attempting to solve this problem using an algorithm based on K-medoids clustering.

To solve this issue, we can start by viewing the K-medoids model presented in section 4.1 as a special case of the jump model framework from Bemporad et al. [2018]. To do this, let $X = Y$, $r(\theta) = \mathcal{L}(\mathbf{s}) = 0$, $\theta_i = t_i$ and $\ell(x_t, y_t, \theta_i) = d(y_t, y_{t_i})$. Further, we can modify this slightly by letting $\mathcal{L}^{\text{trans}}(i, j) = \lambda \cdot \mathbb{I}\{i \neq j \neq 0\}$. Doing this, we arrive at a model of the form

presented in equation (2). This can be written explicitly as

$$\sum_{t=1}^{T-1} \Big[d(y_t,\, y_{t_{s_t}}) + \lambda \mathbb{I}\{s_t \neq s_{t+1}\} \Big] + d(y_T,\, y_{t_{s_T}}). \tag{6}$$

If we, for example, use the squared Euclidean distance as our dissimilarity, we arrive at a model very similar to the regular K-means based jump model, with the crucial difference being that we have now restricted our choice of centroids to members of our original data set.

Optimization of fitting objective (6) is done as for objective (2), where we iterate between minimizing holding the state sequence fixed and minimizing holding the parameters fixed. The former step, where we optimize over the parameters, can be done by letting the medoid for each cluster be the observation in that cluster that minimizes the sum of dissimilarities to all other observations in the cluster. This is the main advantage of using the medoids-based jump model; since our parameters space is now finite, optimization when keeping the state sequence fixed is trivial regardless of choice of dissimilarity. The latter step can as before be done using dynamic programming. This optimization strategy is very similar to the heuristic, alternating optimization method for regular K-medoids presented in section 4.1. Thus, we have constructed a jump model of the form given by equation (2) that allows for a more robust choice of loss function, while at the same time being realistic to implement from a computational perspective.

In addition, we would like to allow for automatic feature selection in such medoids-based jump models. Witten and Tibshirani [2010] note that it is possible to rewrite the regular K-medoids problem as maximizing a fitting objective on the form of equation (3)

$$\sum_{p=1}^{P} \Bigg(\sum_{t=1}^{T} \big[d_p(x_{tp},\, x_{t_0,p}) - d_p(x_{tp},\, x_{t_{s_t},p}) \big] \Bigg)$$

where t_0 denotes the index of the observation that minimizes distance to all other observations in the data set, i.e. the medoid for the whole data set. This fitting objective can be viewed as maximizing the between-cluster sum of point-to-medoid distances. Then, we arrive at the medoids-based jump model by subtracting the state sequence loss and maximizing the resulting fitting objective

$$\sum_{p=1}^{P} \Bigg(\sum_{t=1}^{T} \big[d_p(x_{tp},\, x_{t_0,p}) - d_p(x_{tp},\, x_{t_{s_t},p}) \big] \Bigg) - \lambda \sum_{t=1}^{T-1} \mathbb{I}\{s_t \neq s_{t+1}\}.$$

From this, it is then possible to arrive at a sparse version of the medoids-based jump model by maximizing

$$\sum_{p=1}^{P} w_p \Bigg(\sum_{t=1}^{T} \big[d_p(x_{tp},\, x_{t_0,p}) - d_p(x_{tp},\, x_{t_{s_t},p}) \big] \Bigg) - \lambda \sum_{t=1}^{T-1} \mathbb{I}\{s_t \neq s_{t+1}\}$$

subject to the constraints $(\star)$ from section 3.3:

(i) $\|\mathbf{w}\|_2^2 \leq 1$

(ii) $\|\mathbf{w}\|_1 \leq \kappa$

(iii) $w_p \geq 0, \quad \forall p.$

The optimization could be done similarly to before, where we iterate between fitting a medoids-based jump model on weighted data and then updating the weights given the new state sequence by solving the soft-thresholding problem. However, when we run this algorithm, we notice that the feature selection performs poorly.

Instead, we can update the weights by solving the soft-thersholding problem for the following fitting objective

$$\sum_{p=1}^{P} w_p \big(\mathrm{TD}_p(\mathbf{s}) - \mathrm{WCD}_p(\mathbf{s}) \big)$$

given a fixed state sequence $\mathbf{s}$, subject to constraints $(\star)$, where TD denotes the average pairwise dissimilarity of the total data set

$$\mathrm{TD}_p(\mathbf{s}) = \frac{1}{T} \sum_{i,j} d_p(x_{ip},\, x_{jp})$$

and WCD denotes the average within-cluster pairwise dissimilarity

$$\mathrm{WCD}_p(\mathbf{s}) = \sum_{k=1}^{K} \frac{1}{T_k} \sum_{i,j \in \{1 \leq t \leq T;\, s_t = k\}} d_p(x_{ip},\, x_{jp})$$

where T_k denotes the number of observations belonging to state k. We are now updating states and weights using two different fitting objectives. While this results in a model that is less theoretically sound, it also results in feature selection results that are actually useful. We do not know why this is the case.

In our specific case, we are interested in developing a more robust jump model with feature selection. To this end, we will use Manhattan distance as our choice of dissimilarity measure. The optimization procedure for the medoids-based jump model with Manhattan distance and feature selection is summarized by Algorithm 2. The initialization is done as before using K-means++. It is of course possible to extend this algorithm to other dissimilarity measures: instead of computing the weighted data in step 2(a), simply precompute the weighted dissimilarities before running the medoids-based jump model in step 2(c).

Algorithm 2 Fitting sparse medoids-based jump model with Manhattan distance.

Input: Standardized observations $y_1, \ldots, y_T$ of P features, number of hidden states K and tuning parameters λ, κ.

1. Initialize feature weights $\mathbf{w}$ by setting $w_1^0, \ldots, w_P^0 = 1/\sqrt{P}$.
2. For $i = 1, \ldots$ up until $\|\mathbf{w}^i - \mathbf{w}^{i-1}\|_1 / \|\mathbf{w}^{i-1}\|_1 < 10^{-4}$:
   (a) Compute weighted features $z_t = y_t \cdot \mathrm{diag}(\mathbf{w}^{i-1})$, for $t = 1, \ldots, T$.
   (b) Initialize state sequence $\mathbf{s}$ as $\mathbf{s}^0 = (s_1^0, \ldots, s_T^0)$.
   (c) For $j = 1, \ldots$ up until $\mathbf{s}^j = \mathbf{s}^{j-1}$:
      i. Fit model parameters,
      $$t_k^j = \underset{t_k \in \{1 \le t' \le T;\, s_{t'}^{j-1} = k\}}{\arg\min} \sum_{t \in \{1 \le t' \le T;\, s_{t'}^{j-1} = k\}} \|z_t - z_{t_k}\|_1$$
      for $k = 1, \ldots, K$.
      ii. Fit state sequence, $\mathbf{s}^j = \arg\min_{\mathbf{s}} \Big\{ \sum_{t=1}^{T-1} \Big[\|z_t - z_{t_{s_t}^j}\|_1 + \lambda \mathbb{I}\{s_t \neq s_{t+1}\} \Big] + \|z_T - z_{t_{s_T}^j}\|_1 \Big\}$.
   (d) Update weights $\mathbf{w} = \frac{S(\mathbf{a}_+, \Delta)}{\|S(\mathbf{a}_+, \Delta)\|_2}$, where $a_p = \mathrm{TD}_p(\mathbf{s}) - \mathrm{WCD}_p(\mathbf{s})$.

Output: Model parameters $\{t_i\}_{i=1}^K$, state sequence $\mathbf{s}$ and weights $\mathbf{w}$.

---

5 Simulations and Empirical Study

The fact that we are free to pick dissimilarity measures in the sparse medoids-based jump models from section 4.2 allows for great flexibility and variety of applications. We could, just to name a couple of examples, use it for feature selection when clustering categorical time series data or use a more robust dissimilarity measure (such as Manhattan distance) when we suspect the data might come from a more fat tailed distribution, which is very common in e.g. financial applications. In this section, we will study how well the sparse medoids-based jump model with Manhattan distance works in the fat tailed setting compared to the other jump models discussed in this thesis. This is done by doing a simulation study, whose set-up will be based on previous simulation studies from Witten and Tibshirani [2010] and Nystrup, Kolm, et al. [2021].

5.1 Previous Studies

In Witten and Tibshirani [2010], they compare sparse K-means to regular K-means, as well as a series of other clustering methods typically used for settings with a high-dimensional feature space, by doing a simulation study. The P-dimensional data was simulated from a mixture of three multivariate normal distributions, that all have identity matrices as their covariance matrices and the following mean vectors

$$\mu_1 = \mu \mathbb{I}_{p \leq q}, \quad \mu_2 = \mathbf{0}, \quad \mu_3 = -\mu \mathbb{I}_{p \leq q}$$

where $\mathbb{I}_{p \leq q}$ is a vector whose first q elements are one and the others are 0, i.e. the first q observations are treated as the relevant features for the clustering problem, while the remaining ones should not be selected by a successful feature selection algorithm. m observations per distribution where simulated, with different combinations of μ, P, q and m. The different clustering algorithms were then applied to these simulations and the resulting classification error rate (CER) compared.

The results of this simulation study show that the sparse K-means algorithm performs worse in terms of CER compared to the regular K-means algorithm when the observations only have relevant features, while it performs significantly better compared to regular K-means for increasing number of irrelevant features. This results hold regardless of choice of μ. In addition, when P, q and m are chosen to be small, the sparse K-means has lower CER compared to the other clustering methods tried. However, for larger P, q and m, regular K-means using the principal components of the data per-

forms better in terms of CER. However this PCA-based K-means algorithm is not able to perform successful feature selection.

In Nystrup, Kolm, et al. [2021] they modify the above simulation setup slightly by deciding which observation will follow which distribution through a Markov chain $\{s_t\}$ taking values in the set $\{1, 2, 3\}$ and with transition matrix

$$\Gamma = \begin{pmatrix} 0.9903 & 0.0047 & 0.0050 \\ 0.0157 & 0.9666 & 0.0177 \\ 0.0284 & 0.0300 & 0.9416 \end{pmatrix}.$$

The transition matrix is taken from a hidden Markov model used to model stock returns, see Nystrup, Lindström, et al. [2020]. Then we can write the conditional distribution of the observation at time t as

$$y_t | s_t = N(\mu_{s_t}, I).$$

Thus, the data is imbued with temporal dependence in this case.

In the simulation setup from Witten and Tibshirani [2010], there was an equal number of simulated observations from each distribution. In the case where the distribution to be simulated from is dependent on a Markov chain, we will not necessarily get such a balanced data set, since one state might be much more prominent than the others. Therefore, CER might give a misleading impression of how well the algorithm is performing and thus, a different way of measuring accuracy is necessary. In Nystrup, Kolm, et al. [2021] they use balanced accuracy (BAC) instead of CER, where BAC is calculated by averaging over the accuracy for each individual state. For example, if we have a total of three states, and all observation belonging to state one and two are correctly clustered but none of the observations belonging to state three is correctly clustered, the BAC will always result in a value of 2/3, while the CER might result in a very low value if the data is imbalanced.

The sparse jump model, along with a series of other cluster methods, are applied to this simulated data for different combinations of μ and P, where we keep $q = 15$ fixed. The results show that the sparse jump model significantly outperforms the regular jump model, as well as most other models that were tried, in most cases. In particular, as P gets larger, the sparse jump model starts performing much better than most of the other models. The results also hold true when correlation is introduced between the irrelevant features.

5.2 Student's t-distributed Features

We are now interested in how well the medoids-based jump model with Mahattan distance performs compared to the other jump models in cases where

there are a significant amount of outliers in the data. The Student's t-distribution is commonly used when we want to develop statistical methods that are robust to outliers and deviations from normality, see for example Lange et al. [1989]. The t-distribution resembles the normal distribution, but with the important differences that the weight of the tails of the t-distribution can be controlled via the degrees-of-freedom parameter υ. In fact, if we let the degrees-of-freedom go to infinity, the t-distribution will be asymptotically equal to the normal distribution. Since the tails can be controlled to be thin/fat, we might not always have that all the moments of the distribution exists. However, in the case of the t-distribution, the expectation exists if $\upsilon > 1$ and in addition the variance exists if $\upsilon > 2$. In Lange et al. [1989] the development of robust methods is done in a parametric setting, by replacing the underlying Gaussian probability distribution with t-distribution. They look mostly at different linear regression problems, but the same has been done for e.g. HMMs in Bulla [2011].

In our case we are working with models that do not assume any specific underlying distribution for the data, so we cannot make any explicit assumptions about our model following the t-distribution. Instead, we will try to use a more robust dissimilarity measure and then slightly modify the simulation setup used in Nystrup, Kolm, et al. [2021], by letting the first 15 features be simulated from a Student's t-distribution with one of the following location parameters for each state

$$\mu_1 = \mu, \quad \mu_2 = \mathbf{0}, \quad \mu_3 = -\mu$$

and degrees-of-freedom equal to υ. The remaining features will be simulated as zero-mean white noise. We will run these simulation for different combinations of μ and υ.

The chosen values for υ are 0.75, 1.5, 3 and 10. The probability density function for t-distributions with these degrees-of-freedom, as well as a standard Gaussian density, have been plotted in figure 6. We notice that the t-distributions have heavier tails for decreasing υ, in fact for $\upsilon = 0.75, 1.5$ the distribution does not even have finite variance. Thus, we expect to get many more, large outliers disturbing the data for smaller υ and therefore the jump models having a harder time estimating the correct state sequence. In the case when $\upsilon = 10$ we expect that the standard jump models will perform comparably to the results obtained in Nystrup, Kolm, et al. [2021].

For the mean parameter μ we will run simulations for the values 0.25, 1 and 2.5. These values represent an increasing degree of separation between the different states, so we expect the jump models to have a harder time estimating the state sequence for $\mu = 0.25$ and it being much easier to cluster

when $\mu = 2.5$. In fact, for $\mu = 2.5$ we are reaching a point were it becomes possible for humans to perform the clustering task with a very high degree of accuracy (at least in the absence of outliers).

We are going to perform the clustering task on the simulated data with four different models: the standard and sparse jump model using squared Euclidean distance and the medoids-based jump model using Manhattan distance with and without feature selection. However, the performance of these models are based on the choice of hyperparameters, the jump penalty λ in all cases, as well as number of features κ for the sparse models. We will perform hyperparameter optimization by using grid search, the same way as in Nystrup, Kolm, et al. [2021]. For the regular and medoids-based jump models we will pick λ among 14 values in the interval $[10^{-2}, 10^{4}]$ that are logarithmically spaced and for the sparse variants of the models we will pick λ from seven values in the interval $[10^{-1}, 10^{2}]$, also logarithmically spaced. For κ, we will pick between 14 equdistant values in the interval $[1, \sqrt{P}]$. This means that in the case of sparse models, we do a 2-dimensional grid-search over $14 \cdot 7 = 98$ pairs of hyperparameter values. The hyperparameters that result in the highest BAC will be picked.

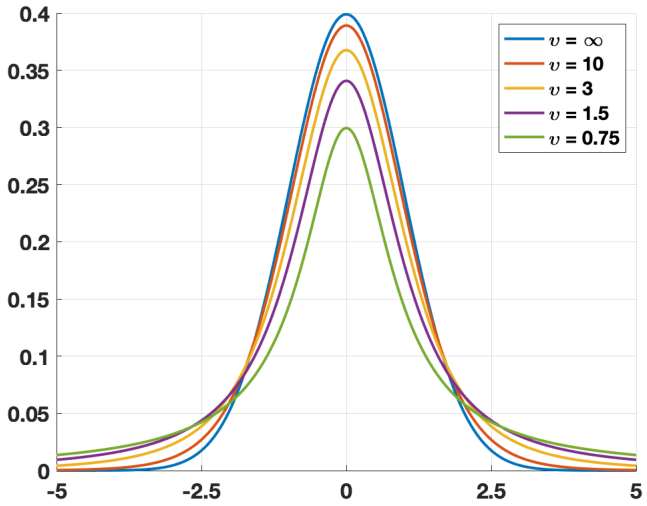

Figure 6: Probability density functions for t-distributions with five different degrees-of-freedom υ (here $\upsilon = \infty$ is the same as a standard Gaussian density).

In addition, we are interested in determining whether the sparse medoids-based jump model performs significantly better compared to the sparse jump model. Typically, if we could assume that sample of the pair of BAC values

were (at least approximately) normally distributed, this could be done using a paired two-sample Student's t-test. However, we do not know the distribution of the paired BAC sample and thus have to use a distribution-free hypothesis test. A commonly used non-parametric hypothesis test for paired samples is the Wilcoxon signed-rank test, see the original article by Wilcoxon [1945]. In the right-tailed case, the signed-rank test will test whether $x - y$ follows a distribution with median bigger than zero, for the paired sample (x, y). Here it is worth noting that the data has to be paired, which in our case means the the pair of BAC values have to result from the different jump models being applied to the same simulated observations, otherwise the comparison will be unfair.

The resulting state sequence from running the jump models might not be sorted in the same way as the real state sequence. In our case with only three states, this issue can be resolved fairly easily be simply running through all permutations of the estimated state sequence and then picking the one that results in highest BAC.

The result of the simulation study with t-distributed features are visible in tables 1, 2 and 3. We notice from theses that in the case where $P = 15$ the jump models using Manhattan distance perform significantly better compared to the models with squared Euclidean distance in the case when $\upsilon = 1.5$ for all values of μ, when $\upsilon = 3$ for $\mu = 0.25$ or 1, and when $\upsilon = 0.75$ for $\mu = 2.5$. This indicates that when there is no feature selection necessary, the Manhattan distance outperforms squared Euclidean distance when there is a moderate amount of outliers given the separation of the states. In the case when $\mu = 0.25$ or 1, degrees-of-freedom $\upsilon = 0.75$ results in too many outliers for the Manhattan distance too handle, but when $\mu = 2.5$ the states seem to be well-separated enough that the Manhattan distance gives better results even when $\upsilon = 0.75$. In the case when $\upsilon = 10$, i.e. the t-distribution is closer to the normal distribution, the medoids-based jump models actually have worse performance. We do not know if this is because of the different choice of metric or due to the use of medoids instead of means.

When we start adding irrelevant features to the simulated observations, we get slightly more mixed results. The medoids-based jump model without features selection seems to deteriorate in performance very quickly as P grows in most cases (the exception being for small values of μ and υ) and the deterioration seems much faster than what happens to the regular jump model. However, the sparse medoids-based jump model manages to retain a significantly higher BAC when $\upsilon = 1.5$ and $\mu = 1$ or 2.5, for both $P = 30$ and 100. In the other cases where the sparse medoids-based model performed significantly better than the standard sparse model for $P = 15$, the sparse medoids-based model eventually becomes just slightly better or worse than

the standard sparse model for big enough P. This is not so strange, considering that $\upsilon = 1.5$ is where the gain from using Manhattan distance seems to be the biggest.

Table 1: Resulting mean BAC values (and their standard deviations in parantheses) of the simulation study described in this section, where a total 500 observations of P features are simulated. The first 15 features are simulated from a t-distribution with location $\mu_1 = \mu$, $\mu_2 = \mathbf{0}$, $\mu_3 = -\mu$ depending on the state, where $\mu = 0.25$, and degrees-of-freedom $\upsilon \in \{0.75, 1.5, 3, 7\}$, the remaining $P - 15$ are simulated from a standard normal distribution. Four different models are applied to the data: standard and sparse jump models with squared Euclidean distance, as well as standard and sparse medoids-based jump models with Manhattan distance. Bolded values indicate that the BAC of the sparse medoids-based jump model is significantly bigger than the BAC of the sparse jump model, using the Wilcoxon signed-rank test at significance level $\alpha = 0.05$.

		$P = 15$	$P = 30$	$P = 100$
$\mu = 0.25$,	$\upsilon = 0.75$			
	Jump	0.35 (0.03)	0.43 (0.10)	0.43 (0.06)
	Medoids Jump	0.35 (0.04)	0.46 (0.10)	0.49 (0.08)
	Sparse Jump	0.35 (0.03)	0.47 (0.11)	0.49 (0.08)
	Sparse Medoids Jump	0.35 (0.02)	0.45 (0.10)	0.46 (0.05)
$\mu = 0.25$,	$\upsilon = 1.5$			
	Jump	0.35 (0.04)	0.41 (0.06)	0.49 (0.10)
	Medoids Jump	0.43 (0.08)	0.47 (0.08)	0.50 (0.08)
	Sparse Jump	0.37 (0.07)	0.40 (0.06)	0.48 (0.09)
	Sparse Medoids Jump	**0.50 (0.07)**	**0.49 (0.08)**	0.50 (0.07)
$\mu = 0.25$,	$\upsilon = 3$			
	Jump	0.47 (0.07)	0.45 (0.05)	0.48 (0.10)
	Medoids Jump	0.52 (0.07)	0.48 (0.07)	0.50 (0.12)
	Sparse Jump	0.48 (0.07)	0.46 (0.06)	0.56 (0.10)
	Sparse Medoids Jump	**0.59 (0.06)**	**0.52 (0.06)**	0.48 (0.10)
$\mu = 0.25$,	$\upsilon = 10$			
	Jump	0.64 (0.09)	0.60 (0.09)	0.55 (0.11)
	Medoids Jump	0.56 (0.07)	0.51 (0.09)	0.50 (0.08)
	Sparse Jump	0.66 (0.12)	0.63 (0.09)	0.64 (0.09)
	Sparse Medoids Jump	0.56 (0.08)	0.51 (0.09)	0.49 (0.05)

We also observe strange behavior for all of the jump models, where their performance in terms of BAC actually increases significantly when we add irrelevant noise features in the case where $\upsilon = 0.75$ for all values of μ, as well as when $\upsilon = 1.5$ and $\mu = 0.25$. We have a hard time explaining these results, but note that even though the BAC values increase, they still perform fairly poorly and remain below 0.5.

Table 2: Resulting mean BAC values (and their standard deviations in parantheses) of the simulation study described in this section, where a total 500 observations of P features are simulated. The first 15 features are simulated from a t-distribution with location $\mu_1 = \mu$, $\mu_2 = \mathbf{0}$, $\mu_3 = -\mu$ depending on the state, where $\mu = 1$, and degrees-of-freedom $\upsilon \in \{0.75, 1.5, 3, 7\}$, the remaining $P - 15$ are simulated from a standard normal distribution. Four different models are applied to the data: standard and sparse jump models with squared Euclidean distance, as well as standard and sparse medoids-based jump models with Manhattan distance. Bolded values indicate that the BAC of the sparse medoids-based jump model is significantly bigger than the BAC of the sparse jump model, using the Wilcoxon signed-rank test at significance level $\alpha = 0.05$.

		$P = 15$	$P = 30$	$P = 100$
$\mu = 1$,	$\upsilon = 0.75$			
	Jump	0.35 (0.03)	0.46 (0.12)	0.48 (0.11)
	Medoids Jump	0.36 (0.06)	0.46 (0.11)	0.50 (0.11)
	Sparse Jump	0.37 (0.06)	0.49 (0.12)	0.51 (0.10)
	Sparse Medoids Jump	**0.38 (0.07)**	0.48 (0.11)	0.50 (0.10)
$\mu = 1$,	$\upsilon = 1.5$			
	Jump	0.44 (0.12)	0.50 (0.10)	0.49 (0.09)
	Medoids Jump	0.72 (0.13)	0.55 (0.10)	0.53 (0.08)
	Sparse Jump	0.45 (0.12)	0.47 (0.10)	0.53 (0.10)
	Sparse Medoids Jump	**0.76 (0.14)**	**0.78 (0.15)**	**0.74 (0.15)**
$\mu = 1$,	$\upsilon = 3$			
	Jump	0.86 (0.14)	0.86 (0.15)	0.83 (0.15)
	Medoids Jump	0.92 (0.11)	0.81 (0.13)	0.60 (0.10)
	Sparse Jump	0.89 (0.13)	0.88 (0.14)	0.89 (0.14)
	Sparse Medoids Jump	**0.96 (0.06)**	**0.97 (0.07)**	0.88 (0.15)
$\mu = 1$,	$\upsilon = 10$			
	Jump	0.98 (0.04)	0.99 (0.03)	0.99 (0.03)
	Medoids Jump	0.96 (0.07)	0.88 (0.12)	0.68 (0.12)
	Sparse Jump	0.99 (0.02)	1.00 (>0.01)	1.00 (>0.01)
	Sparse Medoids Jump	0.97 (0.06)	0.98 (0.03)	0.96 (0.10)

Table 3: Resulting mean BAC values (and their standard deviations in parantheses) of the simulation study described in this section, where a total 500 observations of P features are simulated. The first 15 features are simulated from a t-distribution with location $\mu_1 = \mu$, $\mu_2 = \mathbf{0}$, $\mu_3 = -\mu$ depending on the state, where $\mu = 2.5$, and degrees-of-freedom $\upsilon \in \{0.75, 1.5, 3, 7\}$, the remaining $P - 15$ are simulated from a standard normal distribution. Four different models are applied to the data: standard and sparse jump models with squared Euclidean distance, as well as standard and sparse medoids-based jump models with Manhattan distance. Bolded values indicate that the BAC of the sparse medoids-based jump model is significantly bigger than the BAC of the sparse jump model, using the Wilcoxon signed-rank test at significance level $\alpha = 0.05$.

		$P = 15$	$P = 30$	$P = 100$
$\mu = 2.5$,	$\upsilon = 0.75$			
	Jump	0.35 (0.05)	0.42 (0.05)	0.46 (0.08)
	Medoids Jump	0.43 (0.14)	0.46 (0.07)	0.49 (0.07)
	Sparse Jump	0.35 (0.05)	0.44 (0.07)	0.50 (0.08)
	Sparse Medoids Jump	**0.51 (0.14)**	**0.56 (0.13)**	0.53 (0.09)
$\mu = 2.5$,	$\upsilon = 1.5$			
	Jump	0.68 (0.15)	0.69 (0.13)	0.68 (0.17)
	Medoids Jump	0.94 (0.11)	0.84 (0.16)	0.66 (0.15)
	Sparse Jump	0.72 (0.15)	0.73 (0.16)	0.74 (0.15)
	Sparse Medoids Jump	**0.95 (0.10)**	**0.98 (0.08)**	**0.94 (0.13)**
$\mu = 2.5$,	$\upsilon = 3$			
	Jump	0.98 (0.08)	0.99 (0.04)	0.99 (0.06)
	Medoids Jump	1.00 (>0.01)	0.99 (0.03)	0.90 (0.11)
	Sparse Jump	1.00 (0.01)	1.00 (>0.01)	1.00 (>0.01)
	Sparse Medoids Jump	1.00 (>0.01)	1.00 (>0.01)	1.00 (>0.01)
$\mu = 2.5$,	$\upsilon = 10$			
	Jump	1.00 (>0.01)	1.00 (>0.01)	1.00 (>0.01)
	Medoids Jump	1.00 (0.01)	0.99 (0.03)	0.96 (0.08)
	Sparse Jump	1.00 (>0.01)	1.00 (>0.01)	1.00 (>0.01)
	Sparse Medoids Jump	1.00 (>0.01)	1.00 (0.01)	1.00 (>0.01)

5.3 Example from Financial Application

We have so far discussed the clustering problem and jump models in quite abstract, theoretical terms. It might be of interest to the reader to see how these models work in practice, applied to real data. A common application of jump models is to determine regimes for financial markets and such an application was considered by Nystrup, Kolm, et al. [2021]. There they look at 2516 daily portfolio returns from 49 different stock portfolios and use the sparse jump model to determine different volatility regimes for the data. The mean, the cumulative sum of the mean and the six-day rolling standard deviation of the mean of the portfolio returns are plotted in figure 7.

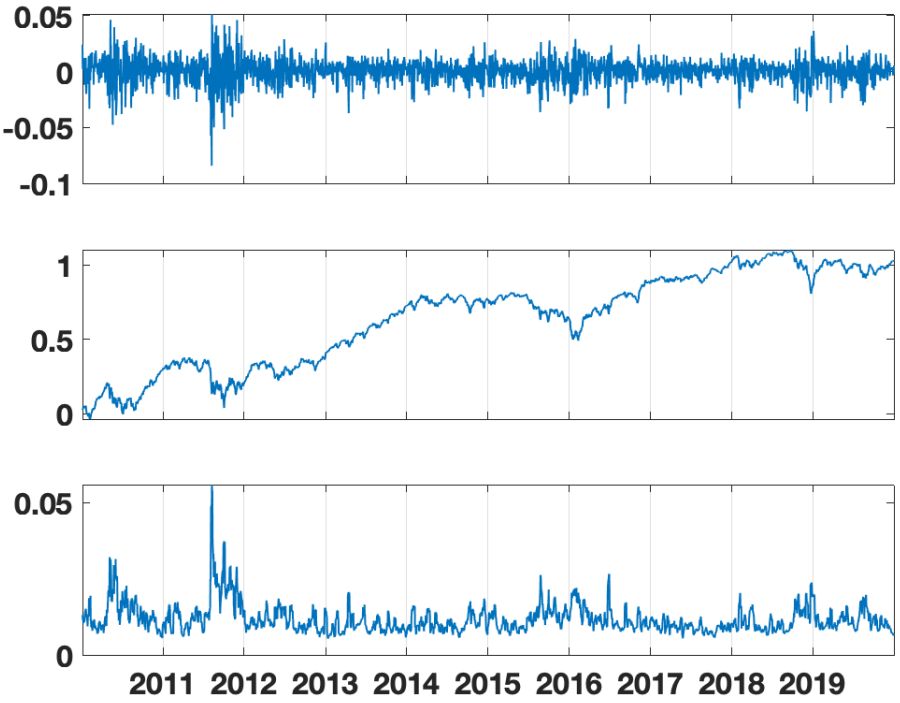

Figure 7: Upper plot: Average daily portfolio returns of 49 different stock portfolios. Middle plot: Cumulative sum of the average daily portfolio returns. Lower plot: Six-day rolling standard deviation of the average daily portfolio returns.

The actual data to be fitted by the sparse jump model is the six-day rolling standard deviations of the portfolio returns. Thus, the sparse jump model will cluster the data into different volatility regimes. In Nystrup et al.

they decided on using $K = 3$ different states, letting the hyperparameters be $\kappa = 5$ and $\lambda = 50$. Additionally, they add 9 different row-permutations of the original data as features, bringing the total number of features up to 490, with only the first 49 being relevant.

The resulting estimated state sequence from using the sparse jump model, along with the feature weights, are visible in figure 8. The resulting state sequence can be interpreted as follows: the state assigned value one in the figure corresponds to a very persistent low-volatility regime, the states with value two corresponds to a more passing mid-volatility regime and the state assigned value three corresponds to a very rare high-volatility regime. We also notice how none of the irrelevant, permuted features are assigned positive feature weights, which mean that the sparse jump model is able to perform the feature selection task well in this case.

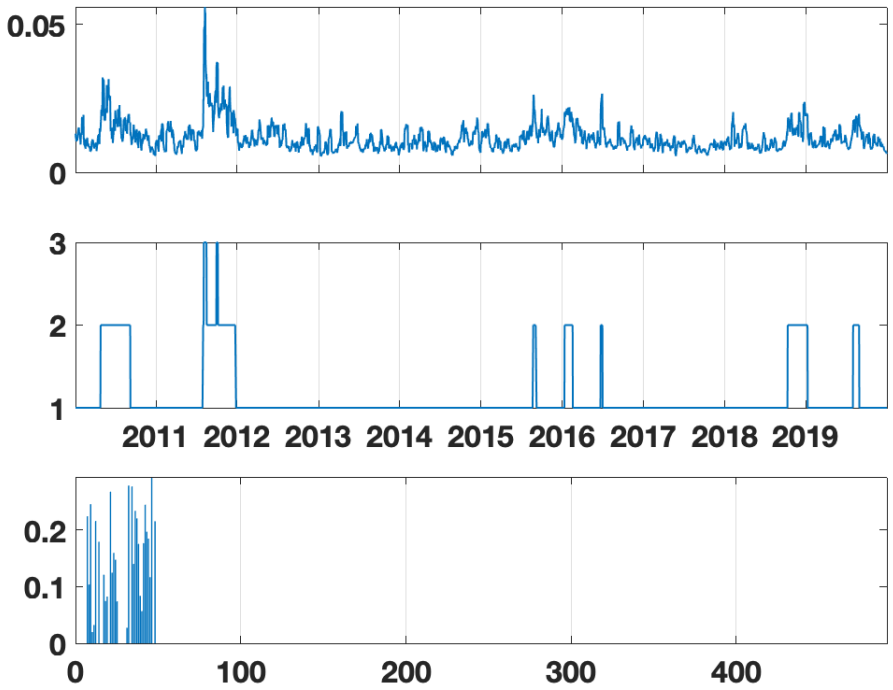

Figure 8: Upper plot: Six-day rolling standard deviation of the average daily portfolio returns. Middle plot: Estimated state sequence obtained from using the sparse jump model. Lower plot: Feature weights resulting from the sparse jump model.

Now, we want to see how our sparse medoids-based jump model performs

in this scenario compared to the standard sparse jump model. First, we will do this using squared Euclidean distance as our dissimilarity measure. Since we are using the same dissimilarity measures a for the regular sparse jump model, we do not expect the estimated state sequence to be much different. The resulting estimated state sequence is visible in figure 9 and it does indeed look similar to what we got using the standard sparse jump model. The main difference seems to be that the middle volatility state seems slightly less common. To get good state sequence, we had to set the hyperparameters to $\kappa = 5$, which is the same as before, and $\lambda = 100$, which is twice as big as before. This might indicate that medoids-based jump models require higher jump penalties compared to when using standard jump models.

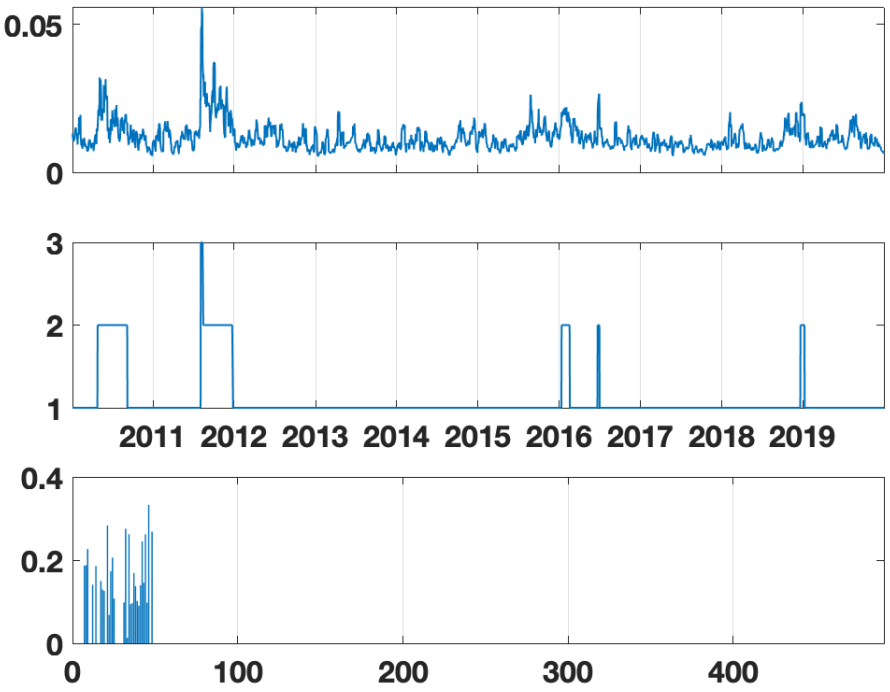

Figure 9: Upper plot: Six-day rolling standard deviation of the average daily portfolio returns. Middle plot: Estimated state sequence obtained from using the sparse medoids-based jump model with squared Euclidean distance. Lower plot: Feature weights resulting from the sparse medoids-based jump model with squared Euclidean distance.

One of the main advantages of the medoids-based jump model is that it allows for flexibility in the choice of dissimilarity measure. To illustrate

this, we decide to use Manhattan distance when fitting the medoids-based model, instead of the squared Euclidean distance which is used by the standard jump model. The hyperparameters are set to $\kappa = 5$ and $\lambda = 50$, the same as before. The state sequence estimated using the sparse medoids-based jump model, along with the feature weights, are visible in figure 10. We notice that the estimated state sequence in this case is quite different, with the mid- and high-volatility regimes being substantially more persistent compared to sparse case. This is probably due to the use of Manhattan distance and the volatility spike occurring in late 2011 having less influence on the estimated state sequence using such a dissimilarity measure. Overall, the sparse medoids-based model performs feature selection well, only putting weights on a subset of the 49 relevant features.

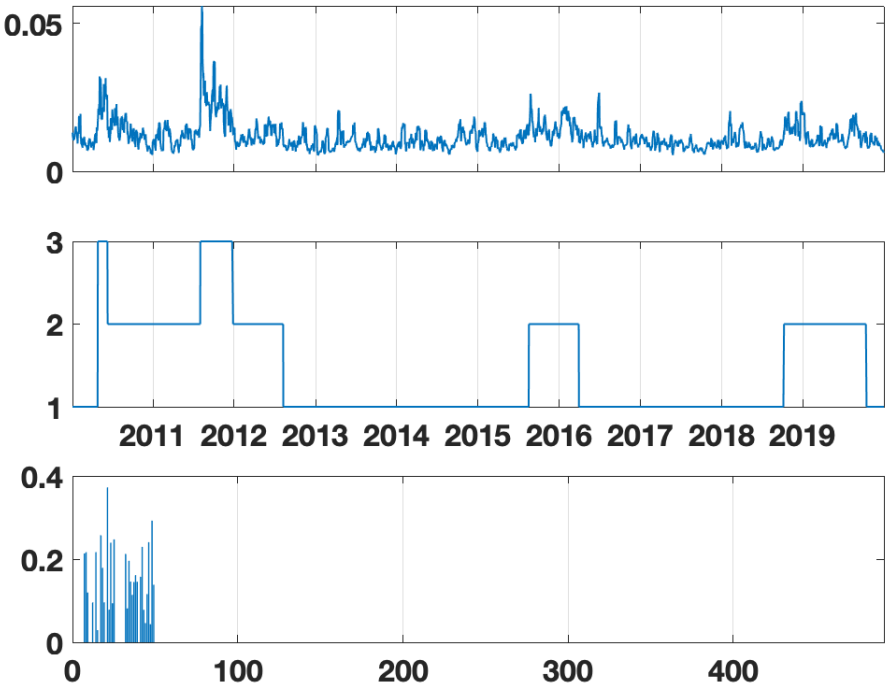

Figure 10: Upper plot: Six-day rolling standard deviation of the average daily portfolio returns. Middle plot: Estimated state sequence obtained from using the sparse medoids-based jump model with Manhattan distance. Lower plot: Feature weights resulting from the sparse medoids-based jump model with Manhattan distance.

6 Conclusion and Further Research

In this thesis, we have managed to show how the K-medoids clustering algorithm can fit into the statistical jump model framework developed by Bemporad et al. [2018]. This has allowed us to add a jump penalty to the K-medoids objective function, analogously to what was done for the jump model based on K-means. Thus, we have developed a clustering method for time series data that allows for flexible choice of dissimilarity measure. This opens up possibilities for using time series clustering with outliers in the data or non-numerical values. Further, we also showed how the medoids-based jump model can be fitted into the sparsity framework from Witten and Tibshirani [2010], allowing for easy and effective feature selection.

The simulation studies show that using Manhattan distance in the medoids-based jump model gives better results when there is a moderate amount of outliers present in the relevant features compared to the regular jump model. This only seems to hold true when the separation between the states is large, however. Additionally, feature selection seems to be working well in the medoids-based jump model, since the sparse medoids-based jump model manages to keep comparable levels of accuracy even when adding several irrelevant noise features.

6.1 Further Research

In our simulation study, we only looked into the case when the relevant features are simulated from fat-tailed distributions and the irrelevant ones come from a standard Gaussian distribution. While the the medoids-based jump model with Manhattan distance manages to do correct feature selection in this case, it would be worthwhile investigating whether the model would also be able to do correct feature selection when the irrelevant noise parameters are sampled from a fat-tailed distribution. This is important, because we don't expect just the relevant features to suffer from outliers when working with real data. It might also be the case that the addition of outliers in only the relevant features made the feature selection problem too trivial, indicating that the good results might be giving a misleading impression.

Something that was noticed from the results of the simulation study, was that the models seemed to increase in performance for ill-separated states with a large number outliers when irrelevant Gaussian noise features where added. We could not come up with a good explanation for this, and it might be worthwhile to make further, more detailed investigations into this to try and find a reasonable explanation.

Further, we notice that feature selection based on the between-cluster

sum of point-to-medoids dissimilarities resulted in poor feature selection. We could not figure out why this was the case. However, the more observations we have access to, the closer the medoids-based model should be to the regular one, indicating that feature selection based on point-to-medoids dissimilarities might perform better when the number of observations increase. This hypothesis would be worth investigating.

Of course, we also would want to apply the model to more scenarios using real data. One potential application could be in the financial setting, which is well-known to typically follow fat-tailed distributions. Even though we did investigate this briefly, we looked at smoothed, portfolio data, which should result in fairly robust results regardless of method chosen. In cases with more volatile, non-smoothed data, using the Manhattan distance in this case could potentially lead to more interpretable results with fewer states necessary. Another application would be to use the medoids-based jump model with categorical time series data.

7 References

Arthur, David and Sergei Vassilvitskii (2007). "K-Means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. Society for Industrial and Applied Mathematics, pp. 1027–1035.

Baker, J. (1975). "The DRAGON system–An overview". In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 23.1, pp. 24–29.

Baum, Leonard E. and Ted Petrie (1966). "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *The Annals of Mathematical Statistics* 37.6, pp. 1554–1563.

Bemporad, Alberto et al. (2018). "Fitting jump models". In: *Automatica* 96, pp. 11–21.

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex Optimization*. 1st ed. Cambridge University Press.

Bulla, Jan (2011). "Hidden Markov models with t components. Increased persistence and other aspects". In: *Quantitative Finance* 11.3, pp. 459–475.

Chopin, Nicolas and Omiros Papaspiliopoulos (2020). *An Introduction to Sequential Monte Carlo*. 1st ed. Springers Series in Statistics. Springer.

Cortese, Federico, Petter Nils Kolm, and Erik Lindström (2023). "Generalized Information Criteria for Sparse Statistical Jump Models". In: *Symposium i anvendt statistik 2023*. Ed. by Peter Linde. Danmark Statistik & Copenhagen Business School, pp. 68–78.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2017). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer Series in Statistics. Springer.

Huber, Peter J. and Elvezio M. Ronchetti (2009). *Robust Statistics*. 2nd ed. Wiley Series in Probability and Statistics. John Wiley & Sons.

Jakobsson, Andreas (2019). *An Introduction to Time Series Modeling*. 3rd ed. Studentlitteratur.

Kaufman, Leonard and Peter J. Rousseeuw (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. 1st ed. Wiley Series in Probability and Statistics. John Wiley & Sons.

Lange, Kenneth L., Roderick J. A. Little, and Jeremy M. G. Taylor (1989). "Robust Statistical Modeling Using the t Distribution". In: *Journal of American Statistical Association* 84.408, pp. 881–896.

Li, Na and Matthew Stephens (2003). "Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data". In: *Genetics* 165.4, pp. 2213–2233.

Llyod, Stuart (1982). "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2, pp. 129–137.

Nystrup, Peter, Petter Kolm, and Erik Lindström (2021). "Feature selection in jump models". In: *Expert Systems with Applications* 184, p. 115558.

Nystrup, Peter, Erik Lindström, and Henrik Madsen (2020). "Hyperparameter Optimization for Portfolio Selection". In: *The Journal of Financial Data Science* 2 (3), pp. 40–54.

Park, Hae-Sang and Chi-Hyuck Jun (2009). "A simple and fast algorithm for K-medoids clustering". In: *Expert Systems with Applications* 36.2, Part 2, pp. 3336–3341.

Piga, Dario, Valentina Breschi, and Alberto Bemporad (2020). "Estimation of jump Box-Jenkins models". In: *Automatica* 120, p. 109216.

Rabiner, L.R. (1989). "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286.

Schubert, Erich and Peter J. Rousseeuw (2021). "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA and CLARANS algorithms". In: *Information Systems* 101, p. 101804.

Sipos, Ivett, Attila Ceffer, and J. Levendovszky (2017). "Parallel Optimization of Sparse Portfolios with AR-HMMs". In: *Computational Economics* 49.

Viterbi, A. (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* 13.2, pp. 260–269.

Wilcoxon, Frank (1945). "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6, pp. 80–83.

Witten, Daniela and Robert Tibshirani (2010). "A Framework for Feature Selection in Clustering". In: *Journal of the American Statistical Association* 105.490, pp. 713–726.

Master's Theses in Mathematical Sciences 2023:E54
ISSN 1404-6342

LUNFMS-3121-2023

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/