JÖNKÖPING UNIVERSITY
School of Engineering

Forecasting With Feature-Based Time Series Clustering

PAPER WITHIN *Computer Science*
AUTHORS: *Conrad Tingström, Johan Åkerblom Svensson*
TUTOR: *Florian Westphal*
JÖNKÖPING *July 2023*

Examiner:	Vladimir Tarasov
Supervisor:	Florian Westphal
Scope:	30 credits
Date:	2023-07-05

Abstract

Time series prediction plays a pivotal role in various areas, including for example finance, weather forecasting, and traffic analysis. In this study, time series of historical sales data from a packaging manufacturer is used to investigate the effects that clustering such data has on forecasting performance. An experiment is carried out in which the time series data is first clustered using two separate approaches: k-means and Self-Organizing Map (SOM). The clustering is feature-based, meaning that characteristics extracted from the time series are used to compute similarity, rather than the raw time series. Then, A set of Long Short-Term Memory models (LSTMs) are trained; one that is trained on the entire dataset (global model), separate models trained on each of the clusters (cluster-based models), and finally a number of models trained on individual time series that are proportionally sampled from the clusters (single models). By evaluating the LSTMs based on Mean Absolute Error (MAE) and Mean Squared Error (MSE), we assess their consistency and predictive potential.

The results reveal a trade-off between the consistency and predictive performance of the models. The global LSTM model consistently exhibits more stable performance across all predictions, showcasing its ability to capture the overall patterns in the data. However, the cluster-based LSTM models demonstrate potential for improved predictive performance within specific clusters, albeit with higher variability. This suggests that certain clusters possess distinct characteristics that allow for better predictions within those subsets of the data. Finally, the single LSTM models trained on individual time series, showcase even wider spreads of scores.

The analysis suggests that the availability of training data plays a crucial role in the robustness (i.e., the ability to consistently produce similar results) of the forecasting models, with the global model benefiting from a larger training set. The higher variability in performance seen for the models trained with smaller training sets indicates that certain time series may be easier or harder to predict. It seems that the noise that comes with a larger training set can be either beneficial or detrimental to the predictive performance of the forecasting model on any individual time series, depending on the characteristics of that particular sample.

Further analysis is required to investigate the factors contributing to the varying performance within each cluster. Exploring feature scores associated with poorly performing clusters and identifying the key features that contribute to better performance in certain clusters could provide valuable insights. Understanding these factors might aid in developing tailored strategies for cluster-specific prediction tasks.

Keywords— demand forecasting, time series prediction, time series clustering

Acknowledgements

We would like to thank Assistant Professor Florian Westphal for his support and guidance throughout the thesis process. Additionally, we would like to express our appreciation to our contact at the stakeholder company for daily discussions providing valuable feedback and suggestions, as well as for his effort in curating the data that was used in this project. Finally, we would like to acknowledge the entire team at the stakeholder company, for letting us do our thesis with them and for enabling our work by providing the necessary hardware.

Contents

1 Introduction **1**
1.1 Scope and limitations 1

2 Background **3**
2.1 Time series categorization 3
2.2 Data 3
2.3 Clustering methods 3
2.3.1 K-means 4
2.3.2 Self-organizing map 4
2.3.3 Clustering evaluation metrics 4
2.4 Forecasting methods 5
2.4.1 Long Short-term Memory 5
2.4.2 Forecasting evaluation metrics 6

3 Related work **7**
3.1 Time series forecasting 7
3.2 Time series clustering 7

4 Approach **9**
4.1 Data 9
4.2 Feature extraction 10
4.3 Clustering 12
4.3.1 K-means 12
4.3.2 Self-organizing map 14
4.4 Time series prediction 17

5 Experiment **18**

6 Results **19**
6.1 K-means 19
6.2 Self-organizing map 20

7 Brief discussion and conclusions **23**
7.1 Future work 24
7.1.1 Feature extraction 24
7.1.2 Hyperparameter tuning 24
7.1.3 Classifying time series for prediction 25
7.1.4 Time series segmentation 25

Appendices **29**

Appendix A SOM evaluation scores **30**

Appendix B k-means evaluation scores **31**

Appendix C k-means MAE, MSE per cluster **32**

Appendix D SOM MAE, MSE per cluster **33**

1 Introduction

As companies continue to develop and integrate demand forecasting into their operations, the need for accurate and efficient forecasting methods becomes increasingly pressing. While time series forecasting using machine learning has shown promising results in predicting demand for certain product categories (Ensafi et al., 2022), it can be challenging to apply this approach to a large number of distinct time series, such as the demand for individual products. The computational cost of training separate models for each individual time series can be prohibitive, and it may not be practical to have a unique model for every product or customer (Bandara et al., 2020). To address this challenge, there is a need to develop scalable methods that can efficiently handle large amounts of distinct time series data, for example clustering similar time series together and training models on the cluster and thereby reducing the number of models needing to be maintained.

Clustering is a technique in unsupervised learning that groups similar objects together based on some similarity metric. In the context of time series data, clustering can be used to identify patterns and similarities in data that may not be obvious through visual inspection or traditional statistical methods (Li & Liu, 2018). Time series data is sequential in nature, with each data point representing a measurement taken at a specific point in time. Therefore, clustering time series data is challenging because of the need to capture both temporal and spatial dependencies, which increases the risk of similarities between time series not being easy to capture.

The project herein described addresses the issue of clustering time series in such a way that a forecasting model trained on a subgroup of time series can provide forecasting results that are not significantly worse for each of the involved time series, compared to models trained on individual time series. This problem is important for large manufacturers as it could positively impact both revenue and environmental footprint by streamlining the use of physical storage, reduce material waste, and assist production planning. The goal of this project was to investigate different approaches to clustering time series. These methods were used to extract subgroups of time series from the data set based on similarity. For evaluation, such a cluster of time series that were similar was used to train a number of forecasting models, using the same parameters. One model was trained on all of the time series in the cluster together and separate models were trained for each of the individual times series. The model trained on the combined time series was then compared to those trained separately in predicting the future of the individual time series. Here, the edge case of training a separate model for each individual time series was expected to perform the best, as clustering was assumed to introduce noise and thereby inhibit the prediction accuracy in comparison. The other edge case would be training a single prediction model for the entire data set. Thus, for evaluation, this was done as well.

This project was conducted in cooperation with a manufacturer of plastic and metal products, that at the time of writing managed a system of approximately 40,000 unique items. With the company's AI effort, one of its goals was to explore demand forecasting. Using the above-described idea of clustering time series on similarity it was important that the forecasting result for a single product by a model trained on a group was comparable to that of a model trained on a single product. Additionally, approximately a quarter of the products in the system were produced at the time of writing, while some products had changed over the years or were no longer produced, making it challenging to predict demand accurately.

1.1 Scope and limitations

The purpose of this thesis is to work towards enabling demand forecasting of individual items for a manufacturer of many different items. More specifically, the study's focus lay on the clustering of time series more so than the prediction thereof, and the following question guided the project:

In the presence of many time series, how can individual time series be grouped together for a forecasting model trained on such a subgroup to not significantly underperform on each of the individual time series in the subgroup, compared to separate forecasting models trained on the individual time series?

Many different methods exist that are used to predict time series, with one of them being recurrent neural networks (RNNs). Specifically, a type of RNN called long short-term memory (LSTM) was found to frequently appear in related literature (Kiefer et al., 2021; Lipton et al., 2015; Ramalingam et al., 2022). Due to its prevalence, this was the method of prediction used in this thesis. Note that the focus of the study was on clustering and the prediction was primarily used as a means of comparing the clustering methods. Thus, while it may have inhibited the generalizability of the results, this is the only prediction method that was used.

Similarly to prediction methods, there are many ways to approach time series clustering such as hierarchical clustering, fuzzy clustering, k-means, self-organizing map (SOM), and more (Aghabozorgi et al., 2015). In this thesis, Two such methods were implemented and subsequently compared by means of the prediction accuracy of LSTMs trained on time series according to the resulting clusters. The explored methods are k-means and SOM. Besides different clustering methods, there are also different ways to deal with the data, such as clustering the time series whole (Looij & Ariannezhad, 2021) or spatially segmented (Castán-Lascorz et al., 2022), or clustering on features extracted from the time series (Candelieri, 2017) rather than using the raw values. In this thesis, the latter of these approaches was used.

With this, the aim of the thesis was to attempt to answer the following questions in terms of prediction accuracy, using both clustering methods:

After clustering time series based on the similarity of extracted features, how does a single LSTM trained on all time series in a cluster compare to:

1. *Separate LSTMs trained on the individual time series in the cluster?*
2. *A single LSTM trained on the whole dataset?*

The research questions were formed based on two critical assumptions. Firstly, that training LSTMs on the data used in the project would be possible. Secondly, that there was a way to cluster the time series together that would not significantly reduce the performance (in terms of prediction accuracy) of the LSTMs trained on whole clusters when compared to training separate LSTMs on each individual time series contained therein. Also expected was that training LSTMs on clusters as per described would result in more accurate predictions than would training a global model on the whole data set. These assumptions were tested by comparing the performance of a global model and a model trained on a subgroup of time series to the performance of separate models trained on each individual time series in the subgroup.

It was expected that the used clustering methods, namely SOM and k-means, might provide different numbers of optimal clusters based on appropriate cluster validity indices. Also expected was that a lower number of clusters would affect the performance of the forecasting model trained on the clusters in a negative way. This follows the assumption stated in the above paragraph that an LSTM trained on a larger number of time series would show worse prediction performance compared to one trained on fewer time series.

2 Background

This section presents the data used in the project, as well as the clustering methods used to group the time series. Also, it describes the forecasting method and how the trained models were evaluated.

2.1 Time series categorization

Time series data of demand or sales can be categorized into four different categories, lumpy, intermittent, smooth, and erratic (Syntetos et al., 2005). The behavior of the different categories is explained by Lolli et al. (2017). A lumpy time series exhibits large and irregular spikes. The spikes can occur randomly and do not have to follow a pattern. Intermittent data consist of long low or zero periods, with large spikes occurring. Smooth data changes steadily over time without any sudden changes. Erratic time series have random spikes and drops without any clear patterns or trends. Placing a time series into one of these categories is done by calculating the squared coefficient of variation and the average inter-demand interval and using cut-off values for these.

2.2 Data

The data set contains sales data from 6 different manufacturing companies, and is collected from an Enterprise Resource Planning (ERP) system, with a focus on manufacturing information shared across all the affiliated companies. The raw data contains sales of plastic and metal products with around 130 columns of order information. From these 130 columns, 4 sets of information are needed, these are, (1) when an order is placed (2) how much of a specific product is ordered (3) who placed the order, and (4) what product is ordered. The time series data contains sales values aggregated monthly from 2013 to 2022. Thus, the data set contains around 27000 unique time series with an average length of about 55 data points (i.e., number of months from first to last entry)

Using the cut-off values for demand occurrence and demand magnitude as defined by Syntetos et al. (2005), a clear majority of the data used in this project is classified as intermittent or lumpy, meaning that the average inter-demand interval is quite large and the size of demand shows high variation.

It is worth pointing out that the raw data contained daily values, rather than monthly, and that aggregating the data was a means to reduce the size of the average inter-demand interval, by reducing the portion of zero-valued data points. Despite doing this means losing details in the data, it was done to make the time series easier to cluster and predict. However, the raw data was such that even with heavy aggregation the data was still mostly intermittent. Also, increasing the level of aggregation would make the project less relevant to the industry partner, as they were mostly interested in forecasting at lower aggregation levels.

2.3 Clustering methods

When working with a large number of time series, it is inefficient to create a model for each time series. Instead, a potentially more viable approach is to group time series together and create a general model which can predict multiple time series without being retrained. Two ways to cluster time series are presented here: SOMs and k-means.

2.3.1 K-means

K-means clustering is a technique that partitions data points into K clusters based on how similar they are. The algorithm iteratively updates the center of each cluster until all points are partitioned into K clusters so that the sum of the squared distance between each data point and its cluster center is minimized (Vo et al., 2016). With time series of different lengths, k-means can cluster similar time series based on their feature representations. The distance measure used to calculate the distance between features is the Euclidean distance.

2.3.2 Self-organizing map

A self-organizing map (SOM) is an artificial neural network that can be used to cluster high-dimensional data. SOMs, also referred to as Kohonen maps, were first introduced by Kohonen (1990). SOMs map high-dimensional data to a lower-dimensional output grid, typically in two dimensions. Each node n in the output layer is associated with its respective weight vector w_n, which is of equal length to the input vectors (see Figure 1). The algorithm trains the network iteratively by first sampling one vector v from the set of input vectors and computing the distances between this and the weight vectors of all nodes in the output matrix. The output node with the weight vector that is the most similar to v is then considered the best-matching unit (BMU) of that particular input vector. Finally, after assigning v to its BMU, the algorithm adjusts the values of the weight vectors, making them more similar to v. The more similar a weight vector is to v, the more significant the adjustment is.

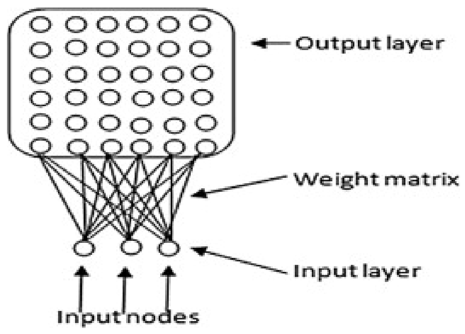


Figure 1. Architecture of self-organising map (Ismail et al., 2011)

In the context of time series data, SOM can identify patterns and similarities between multiple time series. This can be done using raw time series data as input, which would group the time series by the similarity of the actual data points. However, this approach might not always be viable. Particularly since the SOM requires input vectors of equal lengths, which might not be the case for the time series in the dataset. Another method is to first extract the features of the time series and then use the resulting set of feature vectors as input. The extracted features can be basic metrics such as mean and standard deviation, or more complex metrics such as autocorrelations and Fourier transforms. Similar to k-means, the standard distance measure on SOM is Euclidean.

2.3.3 Clustering evaluation metrics

- Silhouette
  The silhouette metric was introduced by Rousseeuw (1987) to measure how well an element fits inside its assigned cluster compared to other clusters. The average distance of elements within one cluster compared to the average distance between elements in a different cluster. The score is

between -1 and 1, where a higher score means that the element is well-placed. The silhouette is defined as:

$$s_i = \frac{1}{N}\sum_{i=1}^{N} \frac{b_i - a_i}{\max\{a_i, b_i\}} \tag{1}$$

where a_i is the average distance between data point i and all other data points in the same cluster, and b_i is the minimum average distance between data point i and all other data points in different clusters.

- Davies-Bouldin
  The Davies-Bouldin index is an evaluation metric for clustering algorithms and was first introduced by Davies and Bouldin (1979). This method is used to find the optimal number of clusters by looking at the average ratio of within-cluster and between-clusters distance. The optimal number of clusters has a low score, which also means that the elements within a cluster are similar to each other and dissimilar to elements within other clusters.

  The Davies-Bouldin index is defined as:

$$\mathrm{DB}(K) = \frac{1}{K}\sum_{i=1}^{K} \max_{j \neq i} \left(\frac{s_i + s_j}{d(c_i, c_j)} \right) \tag{2}$$

  where K is the number of clusters, c_i is the centroid of cluster i, s_i is the average distance of all points in cluster i to the centroid c_i, and $d(c_i, c_j)$ is the distance between the centroids of clusters i and j.

2.4 Forecasting methods

Time series forecasting is a technique used to predict future values based on historical values. In this thesis, univariate data is used. Univariate data use values from a single variable that is often collected on a regular frequency, (e.g, daily, weekly, monthly) to find features and patterns to learn and predict from. From the literature study conducted in Section 3, the method Long Short-Term Memory (LSTM) was selected for this research.

2.4.1 Long Short-term Memory

Long Short-term Memory (LSTM) is based on the Recurrent Neural Network(RNN) architecture. While RNNs were built to process sequential data, such as time series data, RNNs have a problem with handling long-term dependencies in sequential data. For this reason, LSTM was created to handle long sequential data in a better manner (Hochreiter & Schmidhuber, 1997). The LSTM layer consists of a cell state with 3 different gates, an input gate, forget gate, and an output gate (Figure 2).

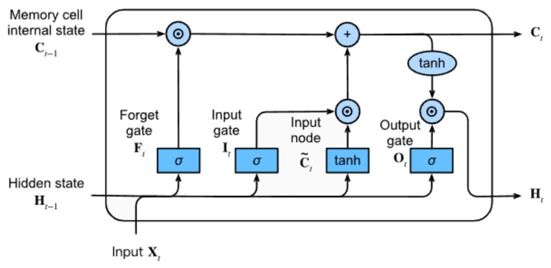


Figure 2. Architecture of a Long Short-Term Memory Unit (Zhang et al., 2021)

These gates are what control the information flow into and out of the cell. With these gates, the LSTM network can selectively forget or retain information that is needed. Within the cell, there is also a cell state which passes information along from one step to the next. With the control of information with the cell state, LSTM is a neural network that works well with handling long-term dependencies in sequential data. (Hua et al., 2019)

This study uses the LSTM configuration of the DeepAR algorithm developed by Salinas et al. (2020). DeepAR is an effective solution for forecasting multiple time series while being capable of making individual-level predictions. It leverages a recurrent neural network architecture that consists of separate LSTM networks, with each network dedicated to modeling a specific time series. This architecture allows the model to capture the distinct characteristics and patterns of each time series while also learning from the global patterns observed across the dataset. By training on multiple time series and learning the global characteristics, the model can leverage the shared patterns and relationships to make informed predictions even for time series with sparse or short histories.

By default, DeepAR employs probabilistic forecasting. However, by specifying the loss function as Mean Squared Error (MSE), the model can be trained to directly minimize the discrepancy between the predicted values and the actual observed values. This modification allows the model to generate point predictions instead of probabilistic forecasts (Herzen et al., 2022).

2.4.2 Forecasting evaluation metrics

For evaluating a forecasting model, it is important to have a clear evaluation of the performance and accuracy of the model. This helps to determine how efficient a model is but also provides information to compare it to other models tested under the same circumstances. In this study, two common metrics for assessing the performance of forecasting models were used, namely the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). Below, these are briefly explained:

- MSE
  MSE is the average squared difference between the predicted and true values. To calculate MSE, the sum of the squared difference between the predicted and true values is divided by the number of observations. As seen in Equation 3 a larger difference between the predicted and true value results in a larger impact than a smaller difference. Overall, the larger MSE score, the worse the performance of the model.
  $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3}$$
- MAE
  Unlike MSE, where the difference value is squared, MAE takes the absolute difference value instead. The absolute differences are then averaged to obtain the mean absolute error. Because MAE does not square the difference, outliers do not have the same impact. Both of these metrics are useful because they consider the magnitude of errors without considering their direction.
  $$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4}$$

  For both Equation 4 and Equation 3 the following applies:
  n is the total number of observations or predictions.
  Y_i represents the actual value of the i-th observation.
  $\hat{Y}_i$ represents the predicted value of the i-th observation.

3 Related work

3.1 Time series forecasting

While LSTM can be seen as a baseline for forecasting problems, extensions of LSTM can solve other issues. These issues can be that when training on multiple time series, they do not always come in the same length. In reference to the data used in this thesis, new products can be introduced and removed, resulting in time series of unequal length. In the paper by Rick and Berton (2022), working with multiple time series of unequal lengths is explored by proposing an extension of LSTM. The proposed model is a combination of a convolutional neural network, LSTM, and auto-encoder. This model is compared to a Temporal convolutional network and does show a better performance on the four energy datasets used.

Depending on the use case, LSTM is not always the best forecasting method. In Ensafi et al. (2022), LSTM is outperformed by both convolutional neural network and Prophet (a parametric forecasting model developed by Facebook which focus on capturing seasonal factors in data (Taylor & Letham, 2017)) on forecasting seasonal data. The models are evaluated using Root Mean Square Error (RMSE) and Mean Absolute Error (MAPE), where LSTM performance is better than statistical methods.

Similarly, in Ramalingam et al. (2022), demand forecasting of pharmaceutical products is explored with deep neural networks, shallow neural networks, and statistical methods. LSTM and stacked LSTM are compared to probabilistic neural networks, generalized regression neural networks, and radial basis function neural networks. The performance of LSTM compared to shallow networks on eight different datasets is mixed, where it is better on two, equal on three, and significantly worse on three. Compared to ARIMA, LSTM's performance is close to equal on all datasets based on the percentage of error.

Each data category requires unique models and hyperparameters, as demonstrated in Kiefer et al., 2021 LSTM is found to be a good option for forecasting on lumpy and intermittent data, outperforming deep learning method multilayer perceptron as well as machine learning methods such as random forest, XG-Boost, and auto-support vector machines, based on the evaluation metrics, mean absolute square error (MASE) and Stock-keeping-oriented Prediction Error Costs (SPEC).

While LSTM is a commonly used method and serves as a good baseline for forecasting, its performance varies depending on the specific characteristics of the data being analyzed (Ramalingam et al., 2022). However, it has been shown that LSTM can show promising results when forecasting on lumpy data (Kiefer et al., 2021) which is the case of this thesis.

3.2 Time series clustering

When attempting to predict sequential data, of which time series is an example, LSTMs are often used owing to their ability to model sequences of different lengths and to capture long-range dependencies (Lipton et al., 2015). Bandara et al. (2020) proposed a prediction model that used time series clustering techniques to identify subgroups of similar time series, which could then be used with RNNs such as LSTM. The authors evaluated their methodology using various clustering algorithms, including K-Means, DBScan, Partition Around Medoids (PAM), and Snob, and found that training separate LSTM models for the resulting clusters consistently outperformed the baseline LSTM model (a single model trained on all-time series) in terms of mean sMAPE accuracy.

Clustering time series can be a computationally demanding task in the context of large datasets. This is particularly true when employing dynamic time warping (DTW) as the distance metrics, owing to the way it considers spatial shift when computing the similarity of time series. The study by Looij and Ariannezhad (2021) examines various clustering methods and distance metrics applied to data processing. The study reveals that replacing DTW with the Euclidean distance measure and applying a single exponential smoothing significantly reduces computational time while retaining the accuracy score. Furthermore, the performance of these methods is compared with the SOM clustering approach. The SOM clustering methods outperform hierarchical clustering methods that use either exponential smoothing or DTW as distance metrics, without requiring any additional pre-processing steps. Evaluation of the clustering performances is conducted using the Calinski-Harabasz and Davies-Bouldin methods, which is shown to be two of the better clustering validation indices by Arbelaitz et al. (2013). The other clustering validity indices that were deemed highly are, silhouette, generalized Dunn, Closeness of Points, and Simplified Davies-Bouldin index. While forecasting performance is assessed using the RMSE metrics, which is preferable when dealing with zero values in the data. The study results demonstrate that SOM clustering, with its superior performance and reduced computational time, is an optimal choice for clustering large datasets. Additionally, the SOM clustering method proposes a more significant number of distinct clusters than hierarchy clustering does.

Candelieri (2017) discusses how k-means clustering is used to classify daily load profiles of building water demand patterns into subgroups. By creating these subgroups with similar time series, a forecasting model's performance should be increased. The clustering method is evaluated by using the silhouette and Calinski-Harabasz scores, however, using the silhouette score might not be viable when working with a large amount of data due to the high computational demand of O(n^2). The Calinski-Harabasz metric also works on the assumption that there are clear clusters with enough distance from each other to have a good distribution (Rachwał et al., 2023). The use of k-means, k-means variations, or combinations is commonly used as a baseline to explore the clustering capabilities. This is also seen where an auto-fuzzy k-means algorithm is used to analyze real-time electricity consumption data. (Li & Liu, 2018). By using validity indices to find the optimal amount of cluster the methods could also successfully identify electricity consumption patterns.

This study done by Shabani et al. (2018) researches the usage of k-means clustering to classify daily load profiles and to group similar building energy demand patterns into subgroups. The results of this work can be used to train different learning models on the subset of data and predict short-term building energy consumption. The proposed approach is expected to improve the performance of predictor models.

In summary, related literature suggests that SOM is a well-performing method for clustering time series and that k-means, an older and more basic approach, is often used as a baseline in comparisons between different methods. However, as pointed out by Liao (2005), both of these methods face limitations when time series are of different lengths. With k-means, the concept of centroids becomes unclear if a cluster contains time series of unequal length. For SOM the problem is similar; the dimension of the weight vectors of the network needs to be defined as the shape of the time series, and so if these are of varying lengths this becomes difficult. To circumvent these challenges inherent to clustering on raw time series, an option is to do feature-based clustering instead. This approach not only enables the use of clustering methods that otherwise would require time series of equal lengths, but also has other benefits such as dimensionality reduction and increased noise-resistance (Wang et al., 2006).

4 Approach

This section provides a description of how the previously introduced methods are implemented in practice. It also outlines the data processing steps that were taken to prepare the data for the models.

4.1 Data

The data was split into training and testing sets with 90 percent for training and the remaining 10 percent for testing. The training data was used to train the clustering models SOM and k-means, whereas the test data was labeled by the clustering methods.

From the test set 10 percent was taken from each cluster to be used for testing the forecasting models, resulting in 1 percent of the data being the final test set. This approach means that 90 percent of the first hold out set was not used in evaluating the forecasting models. However, the size initial split was still necessary, as it needed to be large enough so that the hold out set was representative of the training set. This, in turn, ensured that the distribution of time series across the clusters in the test set was proportional to how the training data was clustered. This also reduced the amount of single models that need to be trained.

Finally, each time series in the sampled 1 percent was split into two parts, one for training and one for testing, with the test data further divided into two sections: 12 points were used as input for the predictor, and 3 points serve as the target for evaluation.

This approach ensures that during training, neither the clustering models nor the forecasting models had access to the test data during training. Prior to the data being split, standardization was applied based on the training data.

Figure 3 shows the steps of how the test data was split. The filled boxes are groups of time series that are clustered in the second step. The last step was the 10% taken to do the last split.

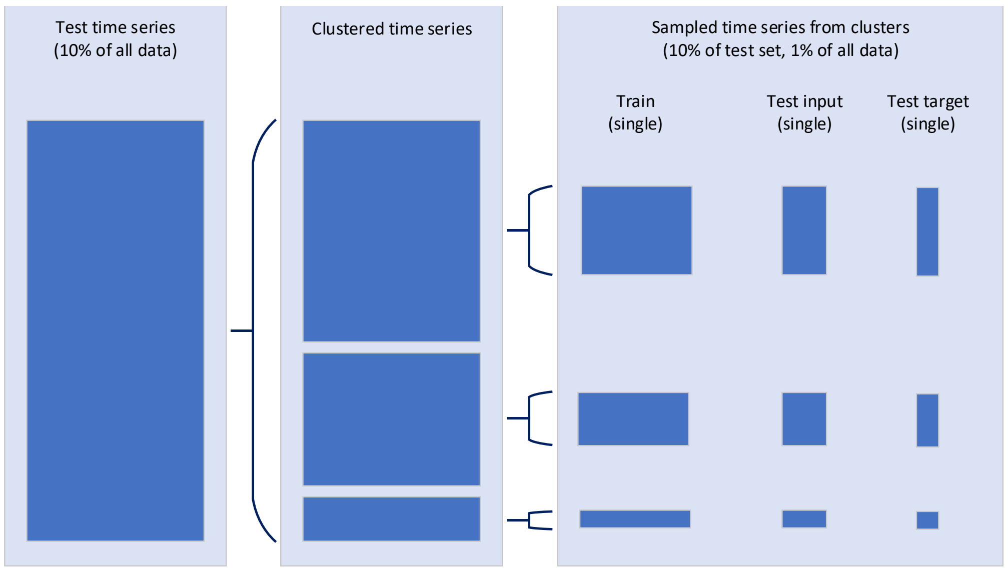


Figure 3. Illustration of sampling and splitting data into train and test sets.

To ensure that every single model had a sufficient amount of data, a length limit of 30 data points was applied during the sampling process. The single models were trained on test data "train single". The cluster-based models were trained on the time series assigned to their cluster and evaluated on the test data sampled from their respective clusters. The global model was trained on all the training data and the data the single models were trained on, and tested on all the test sets.

4.2 Feature extraction

Because of challenges inherent with clustering raw time series of unequal lengths, a feature-based approach was used. Thus, instead of having to compute the similarity of vectors of varying lengths and with potential spatial shift of matching sections of the time series, features were extracted. This achieved a few things to simplify the clustering process. Primarily, instead of dealing with raw time series of varying lengths, the data to be clustered now instead consisted of feature vectors of equal length. As a side effect, this also reduced the dimensionality of the data, reducing the computational load. Also, since the points in the vectors that were to be clustered were now feature values instead of values in time, there was no longer any need to consider spatial shift, and Euclidean distance could be used as the similarity metric.

The feature extraction was primarily done using a python implementation (https://github.com/Nixtla/tsfeatures) of the R package *tsfeatures* originally developed by Hyndman et al. (2023). In addition to the 21 features computed through *tsfeatures*, an additional 11 were initially considered for a total of 32 features. Figure 4 shows the distribution of values for each of these features, across all time series in the data set. The very tight distributions of some of the features mean that most of the time series have roughly the same value in these. Because of this features 21, 22, 23, and 27 were assumed to not be very significant and were therefore discarded from the continuation of the study. While the same could arguably be said for at least feature number 7, this feature came from the *tsfeatures* library from which everything was used. Equally arbitrary was the thresholding for deciding which features had tight enough distribution to be discarded, which was not formally defined. Arguably, there are additional features that are candidates to be removed as well (e.g., 9, 10, 29).

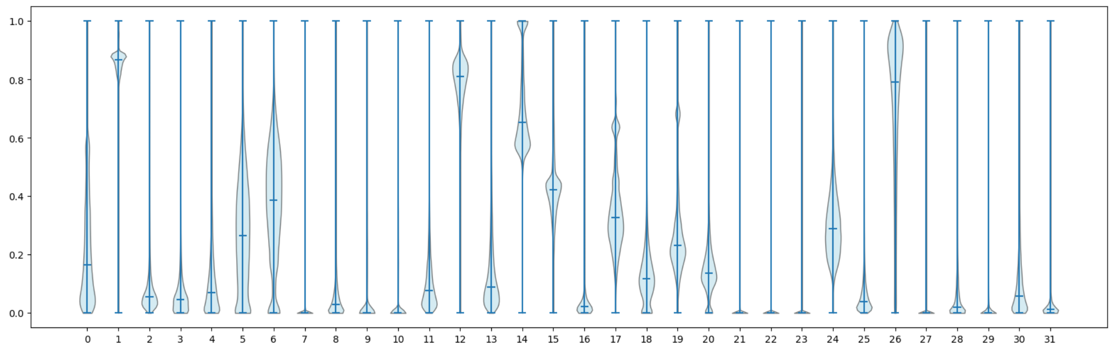


Figure 4. The values of all initially considered features time series in the entire dataset, normalized between 0 and 1. X-axis corresponds to indices in Table 1.

Table 1 shows the final list of features that are used (totaling 28 features), with brief descriptions. Of these, indices 0-20 are extracted using *tsfeatures*, and the remaining are implemented internally.

Index	**Feature name**	**Description**
0	series_length	The length of the time series
1	unitroot_pp	Statistic for the Phillips-Perron unit root test
2	unitroot_kpss	Statistic for the Kwiatowski et al. unit root test
3	stability	Variance of the means of tiled windows
4	x_pacf5	Sum of squares of the first 5 partial autocorrelation coefficients
5	diff1x_pacf5	Sum of squares of the first 5 partial autocorrelation coefficients of differenced series
6	diff2x_pacf5	Sum of squares of the first 5 partial autocorrelation coefficients of twice-differenced series
7	nonlinearity	10 t**2/len(x) where t is the statistic used in Terasvirta's test
8	lumpiness	Variance of the variances of tiled windows
9	alpha	The smoothing parameter for the level in a ets(A,A,N) model fitted to the series
10	beta	The smoothing parameter for the trend in a ets(A,A,N) model fitted to the series
11	flat_spots	The number of flat spots in the series, calculated by discretizing the series into 10 equal sized intervals and counting the maximum run length within any single interval
12	entropy	The spectral entropy of the series
13	crossing_points	The number of times the time series crosses its median
14	arch_lm	Statistic based on the Lagrange Multiplier test of Engle (1982) for autoregressive conditional heteroscedasticity
15	x_acf1	The first autocorrelation coefficient of the series
16	x_acf10	The sum of the squared first ten autocorrelation coefficients of the series
17	diff1_acf1	The first autocorrelation coefficient of the differenced series
18	diff1_acf10	The sum of the squared first ten autocorrelation coefficients of the differenced series
19	diff2_acf1	The first autocorrelation coefficient of the twice-differenced series
20	diff2_acf10	The sum of squared first ten autocorrelation coefficients of the twice-differenced series
21	skew	The skew of the time series
22	kurt	The kurtosis of the time series
23	zero_proportion	The proportion of zeros in the time series as the sum of zeros divided by the length of the series
24	zero_run_length	The mean number of consecutive zeros in the series
25	non_zero_run_length	The mean number of consecutive non-zeros in the series
26	adi	The average demand interval of the series
27	cv2	The squared coefficient of variation for the series

Table 1

Extracted features from the time series with brief descriptions.

The features were extracted after the initial 90-10 split and computed individually on the training and test set. To ensure that the test set is representative of the training data, the features of the respective data sets are normalized and compared. Figure 5 shows the features from the time series in the training set.

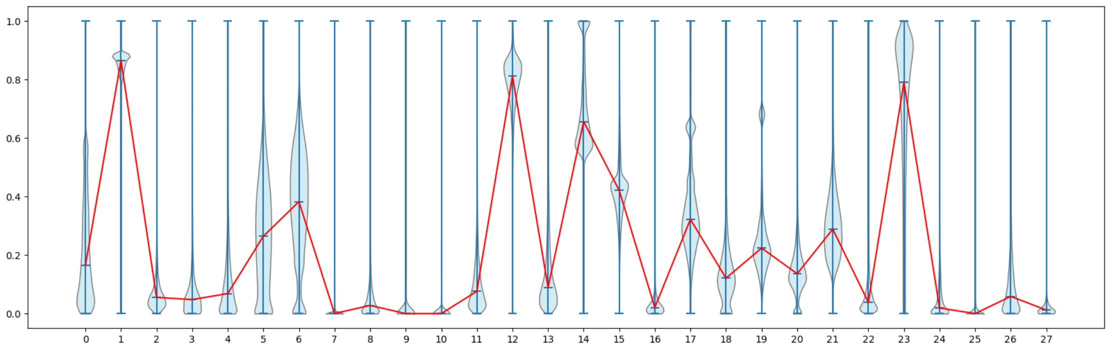


Figure 5. The feature values of all time series in the training data, normalized between 0 and 1. X-axis corresponds to indices in Table 1

Figure 6 shows the features from the test data. Upon visual inspection these were deemed similar enough to those from the training data, meaning that the test set was considered sufficiently representative of the training set.

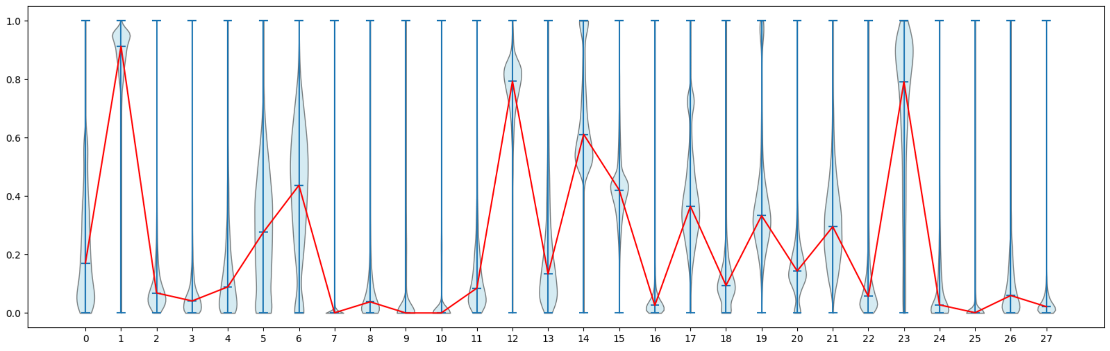


Figure 6. The feature values of all time series in the test data, normalized between 0 and 1. X-axis corresponds to indices in Table 1

4.3 Clustering

4.3.1 K-means

The k-means clustering algorithm was implemented using the tslearn library (Tavenard et al., 2020), with time series features as the input data. Multiple models were trained and evaluated using the k-means algorithm to determine the optimal number of clusters. The range of clusters examined was between 5 and 65, with an interval of 2. The two evaluation metrics described in Section 2.3.3 were used to assess the quality of the clusters. The resulting scores were then normalized between 0 and 1 for ease of interpretation and visualized in Figure 7. The dotted line shows the number of clusters to use in this report. At 35 clusters, both the silhouette and Davies-Bouldin scores are deteriorating indicating that the clusters created at this point are not distinct.

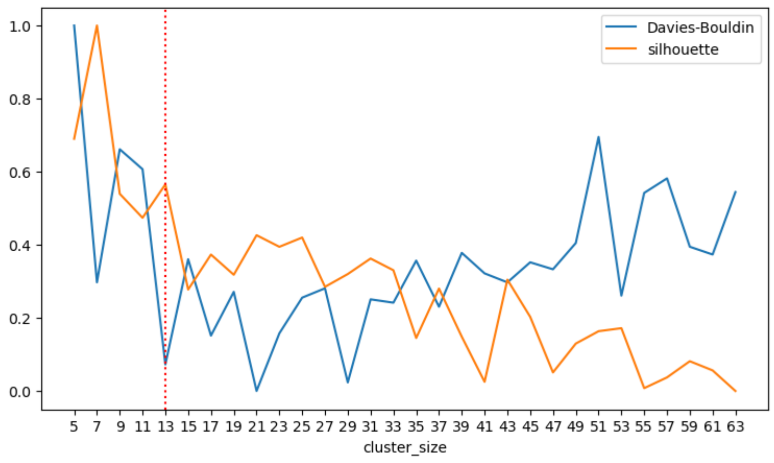
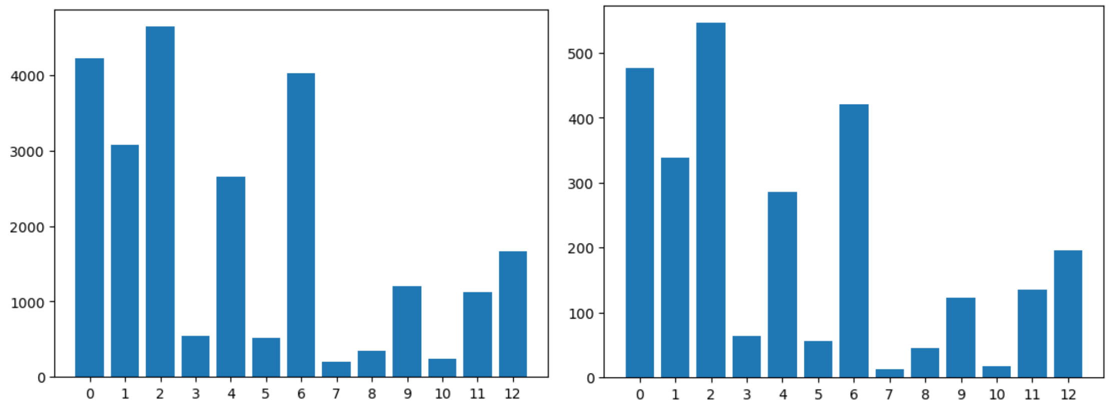


Figure 7. Davies-Bouldin and silhouette score min-maxed between 0 and 1 plotted over different cluster sizes from k-means clustering.

The raw scores for all cluster sizes can be viewed in Appendix B. The top 5 scores for DB and silhouette match on 13 clusters. Therefore the k-means model with 13 clusters was selected as optimal for further analysis. The raw scores do not differ too much, and other cluster sizes such as 7, 21, and 29 are also promising.

The distribution of time series in the clusters can be observed in Figure 8a and Figure 8b. The test set is deemed sufficiently representative of the training data.



(a) Training data *(b)* Test data

Figure 8. Distribution of the time series over 13 clusters on the training and test set made using k-means.

Not all clusters exhibit the same level of differentiation as illustrated by Figure 9 and Figure 10. While these two examples demonstrate significant differences between clusters 4 and 12, it is important to note that the extent of dissimilarity varies across different clusters. The features in both clusters are normalized to values between 0 and 1 using the min-max scaling method. The two clusters exhibit significant differences in multiple features, and the median of the values takes different paths. These differences imply that the two clusters represent distinct patterns or trends in the data. Additionally, the features in Figure 10 are coherent as presented by the narrower range of values displayed in the Figure. In contrast, Figure 9 shows that the features cover a broader area. Meaning that cluster 12 contains time series that have features that create a better tightness within the cluster.

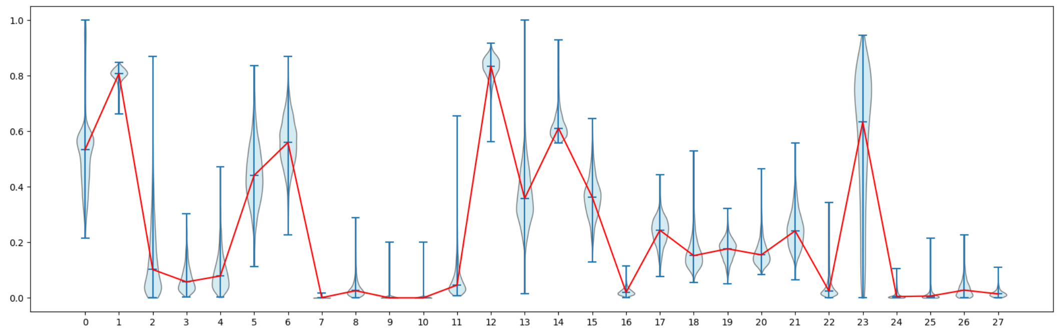


Figure 9. Features density in cluster 4 from k-means clustering (2650 Time series), normalized between 0 and 1. X-axis corresponds to indices in Table 1

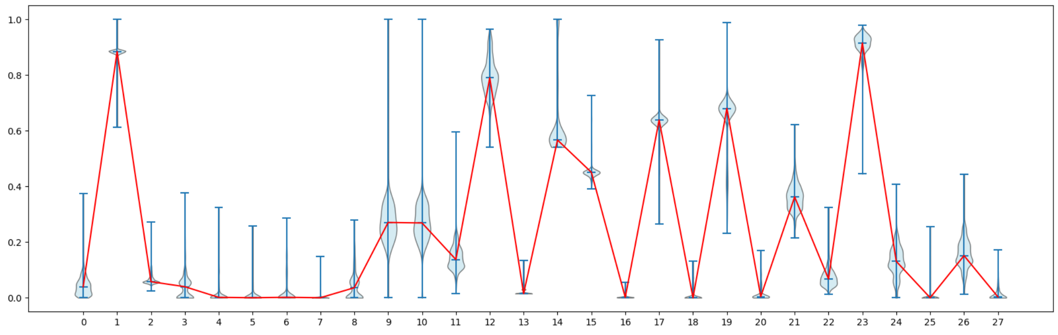


Figure 10. Features density in cluster 12 from k-means clustering (1670 Time series), normalized between 0 and 1. X-axis corresponds to indices in Table 1

4.3.2 Self-organizing map

Self-organizing maps (SOMs) require time series of equal lengths as input, owing to difficulties regarding the definition of the dimension of weight vectors (Liao, 2005). This, however, is a criterion that is not met by the data used in this project since the time series therein are of considerably varying lengths. Thus, features were extracted from the time series as a preprocessing step, prior to clustering on the resulting feature vectors rather than the raw time series, as described in Section 4.2.

Clustering with SOM was implemented using the *MiniSom* library (Vettigli, 2018), using the aforementioned feature vectors representations of the time series as input. Besides defining the input length of the vectors, creating and training the SOM requires a number of hyperparameters to be set. In this project, these used their respective default values where such were available, as set in the implementation that was used. Following is a brief mention of all parameters that were not set to default values:

- *x* and *y*
  The width *x* and height *y* form the dimensions of the two-dimensional matrix output by the SOM and give the number of output nodes $N_{xy} = x \times y$. In this project, each node is treated as a cluster. Because of this, a separate SOM was trained for each output map shape constructed by *x* and *y* ranging from 2 to 8, with a step of 1. This range produces the maps $[(2,2),(2,3),...,(8,7),(8,8)]$ for a total of 49 different shapes. Since this range results in the number of output nodes ranging from 4 to 64, it was chosen since it is similar to the range used for the k-means approach as described in the next section, 4.3.1.

- *random_seed*
  Sets the randomizer for the initialization of the weight vectors. By default, this is set to *none*, which means that the weights are initialized differently each time a new SOM object is created. For the sake of reproducibility, this can be set to a number instead. In this project *random_seed* is set to 2432.

- *num_iteration*
  This value dictates how many times the weight vectors are adjusted. In this project *num_iteration* is set to 1000.

- *use_epochs*
  This parameter can be set to *true* or *false*. If it is set to the former, *num_iteration* is treated as epochs. This means that the weight vectors are adjusted once for each input vector, and this process is repeated *num_iteration* times. If it is set to *false*, the input data is iterated over just once, with the weight vectors being updated *num_iteration* times for each input vector. In this project, this is set to *true*.

The two evaluation metrics described in Section 2.3.3 were used to assess the quality of the clusters. The resulting scores were then normalized between 0 and 1 for ease of interpretation and visualized in Figure 11. The dotted lines show the out map shapes (3, 4) and (4, 3), as they resulted in the lowest respective highest (as well as identical) silhouette and Davies-Bouldin scores. For larger maps, the scores got consistently worse.

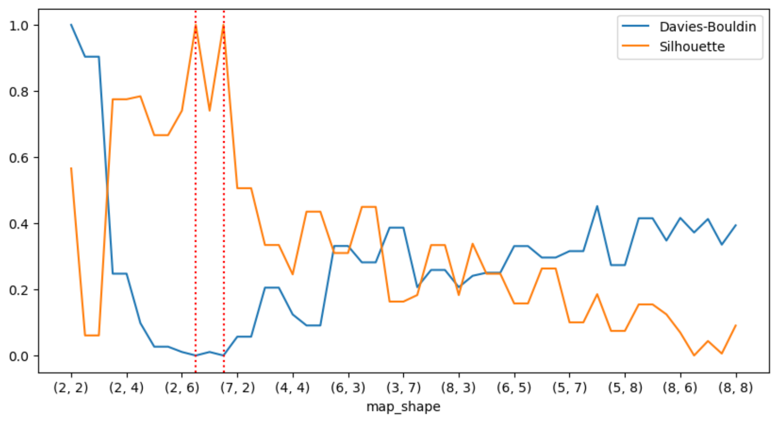
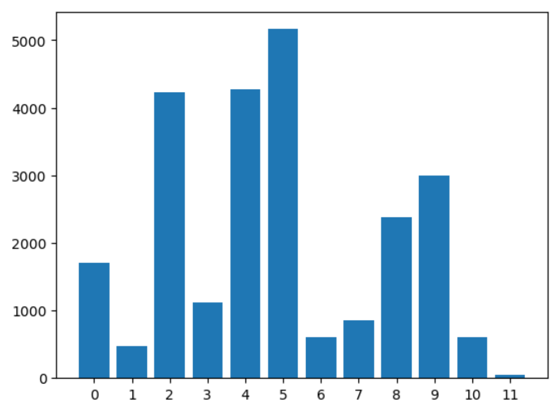


Figure 11. SOM features plotted scores, dotted lines show best map shapes.

Shown in Appendix A are the scores for the different map shapes. Note that like the case with (3, 4) and (4, 3) most other such map-shape pairs showed identical values for both scores. The only two exceptions were the shape pairs (6, 8) and (8, 6), as well as (7, 8) and (8, 7), which display some differences. Bold text shows the top five scores for the two metrics and the best-performing output map shapes in both categories were (3, 4) and (4, 3). Of these two, the former was subsequently used for the continuation of the study.

Using the SOM to cluster the training and test sets achieved the distributions shown in 12a and 12b, respectively. These results indicate that the test set was representative of the training set.


(a) Training data

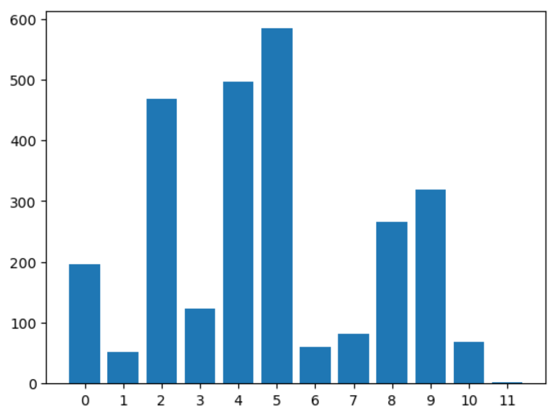

(b) Test data

Figure 12. Distribution of the number of time series over 12 clusters made using SOM on feature vectors.

As an example of the clustering results, Figures 13 and 14 show the features of clusters 4 and 0 respectively; two of the clusters produced by the SOM on the training set with features normalized to values between 0 and 1. The x-axis corresponds to the indices of the features described in Table 1. While these two clusters contain time series in which some features are very similar (e.g., features 7, 25, and 27), they exhibit clear differences in the overall distribution. Notable is also that each of these clusters contains a relatively large portion of the total number of time series while still maintaining a rather tight distribution of values for each feature, especially cluster 0.

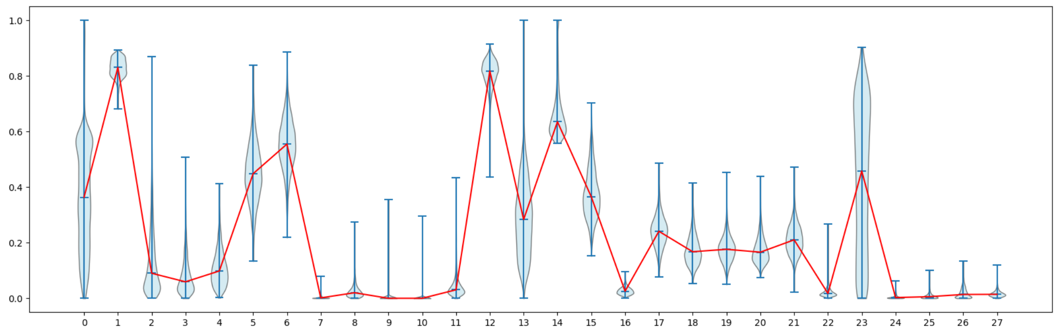


Figure 13. Features from SOM cluster 4 (4268 time series), normalized between 0 and 1. X-axis corresponds to indices in Table 1

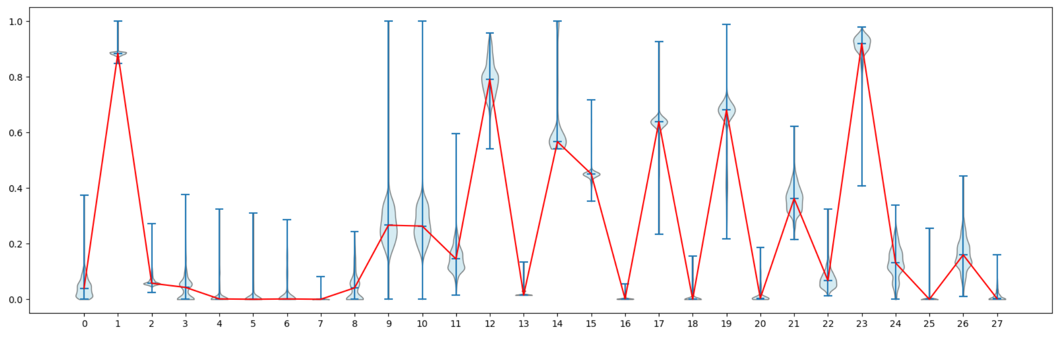


Figure 14. Features from SOM cluster 0 (1700 time series), normalized between 0 and 1. X-axis corresponds to indices in Table 1

4.4 Time series prediction

In order to train global models that could handle multiple time series, the utilization of a suitable algorithm was necessary. DeepAR is such an algorithm that fits the requirements. DeepAR is a deep learning algorithm for time series forecasting, developed by Salinas et al. (2020). DeepAR design lets it handle large amounts of data and produce global models for a large number of products that can be predicted on an individual level. The DeepAR algorithm uses RNN which lets it be used with the LSTM architecture and adopt its strengths. The primary focus of this thesis lies in the clustering aspect rather than producing the optimal forecast. The *Darts* library, developed by Herzen et al. (2022), was employed to use the DeepAR algorithm with the LSTM configuration. *Darts* is a Python library created from Pytorch as a user-friendly forecasting and anomaly detection on time series. The library contains multiple models that are easy to use and their RNNModel is equivalent to the DeepAR in its probabilistic version which supports training on multiple time series and can therefore produce global models which is a requirement for this thesis. While the use of Darts may limit the flexibility in adjusting the forecasting model, it does allow for easy and fast implementation.

The architecture of all the DeepAR models used in this study remained the same, with identical learning rates and a number of epochs. The values for the learning rate and the number of epochs were obtained through parameter optimization with the library Optuna (Akiba et al., 2019) by training single models on a 10 percent sample of the data. The hyperparameters tested were batch size, output length, number of epochs, and learning late. From the testing, the values selected are as follows, batch size 16, output length 1, number of epochs 60, and the learning rate as .058.

The global model was designed to handle all the training data available. During the evaluation phase, the global model processes all the time series in the test set and generates predictions for each of them. The cluster-based models are trained on all the time series data within a specific cluster and then used to generate predictions for the time series IDs that belong to the same cluster. A single model was trained for each of the time series that were sampled when creating the test sets and predicts on the respective test time series. The model prediction on each of the time series in the test set was saved along with the metrics described in Section 2.4.2.

5 Experiment

With the steps involved in the study described in more detail in the previous section, this section serves to condense these and more concisely present an overview of the experimental pipeline.Figure 15 shows the high-level steps involved in the experiment, and in which order they were carried out. For further detail see the related subsections in Section 4, here is a brief description:

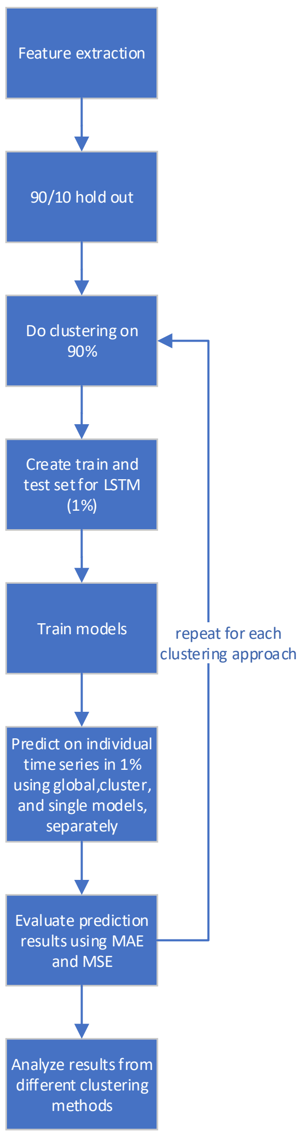

Feature extraction
90/10 hold out
Do clustering on 90%
Create train and test set for LSTM (1%)
Train models
repeat for each clustering approach
Predict on individual time series in 1% using global,cluster, and single models, separately
Evaluate prediction results using MAE and MSE
Analyze results from different clustering methods

Figure 15. Flowchart of the experiment pipeline.

1. **Feature extraction**
   Extract and select features from the time series. Features that show exceedingly low distribution are removed.
2. **90/10 hold-out**
   The data is first split into a 90/10% training and test set as described in Section 4.1.
3. **Train global LSTM on 90%**
   The global model is trained on the 90% training and saved for later use. The hyperparameters used are specified in Section 4.4.
4. **Do clustering on 90%**
   Clustering is done with one method at a time.
5. **Create train and test set for LSTM (1%)**
   A test set is created with 10% from each cluster created in the previous step, resulting in a 1% test set. creating a test set here results in different test sets for the methods used. (Since we are not testing how the clustering approaches compare to each other, but rather how the clustering performs against the global and single models, this does not matter.)
6. **Train models**
   A DeepAR LSTM model is trained for each cluster with time series in the train set labeled to the corresponding cluster. Single models are trained on the time series belonging to the 1% with 15 points being reserved for input and evaluation. The global model is trained on all the training data plus the data the single models are trained on.
7. **Predict on individual time series in 1% using global, cluster and single models, separately**
   The global model predicts on all of the time series in the 1% test set. The single models predict on the series it was trained on. The cluster-based models predict on the time series in the test set labeled to the corresponding model.
8. **Evaluate prediction results using MAE and MSE**
9. **Repeat steps 3-7 for each clustering method**
10. **Analyze results from different clustering methods**

6 Results

In this section, the results from the global, cluster, and single LSTMs that were trained for each clustering approach are presented and interpreted.

6.1 K-means

The forecasting models were evaluated using the metrics MAE and MSE. The scores were used to calculate the median, mean, standard deviation, maximum, and minimum values. The scores are presented in Table 2. When comparing the median, mean, and standard deviation across the different models, the global model performed the best. It is worth noting that both the cluster-based and single models produce a better MAE score on some occasions. The presence of outliers can be observed by examining the maximum and standard deviation values. To account for this, the median value was used instead of the mean.

	global mae	cluster mae	single mae
median	0.439896	0.508154	0.627692
mean	1.128365	1.256343	1.342823
std	6.322498	6.480408	6.356929
max	83.244719	84.853014	84.022116
min	0.026945	0.024322	0.024055

(a) *MAE scores from global, cluster-based and single models*

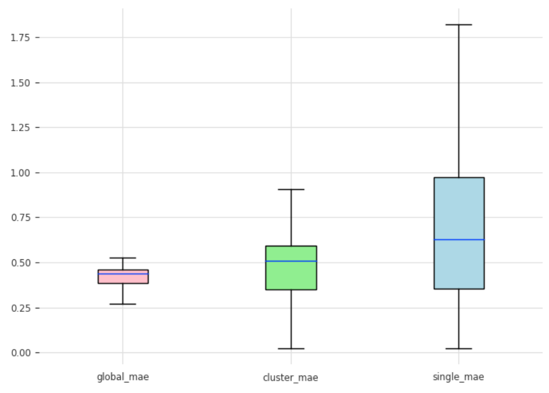
	global mse	cluster mse	single mse
median	0.195744	0.264993	0.504811
mean	47.873314	50.701110	49.092852
std	543.832559	566.528736	551.558192
max	7839.159348	8205.494202	7967.014469
min	0.000768	0.000894	0.000628

(b) *MSE scores from global, cluster-based and single models*

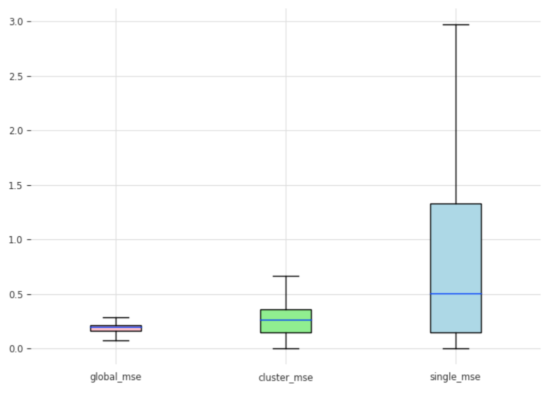
Table 2

MAE and MSE scores from forecasting based on the k-mean clusters.

Figure 16 shows two box plots based on the raw metric values for each time series. The left boxplot represents the MAE scores, while the right boxplot represents the MSE scores. In both images, the leftmost box represents the global model, the center box the cluster-based models, and the rightmost the single models. To increase readability, the outliers were removed from the plots due to the high maximum values reported in Table 2.



(a) K-means MAE

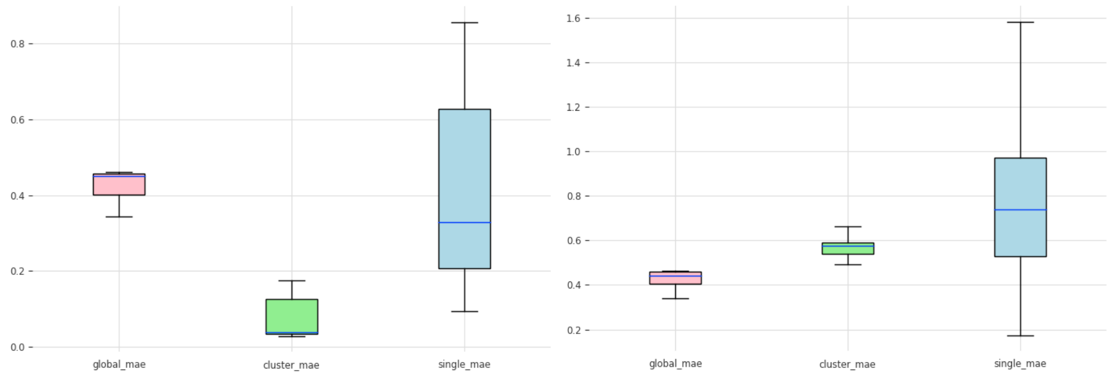


(b) K-means MSE

Figure 16. MAE and MSE scores over all test time series.

The global models consistently provided more consistent results, as indicated by the tightness of the leftmost boxes. The cluster-based models exhibited larger quantile spreads, with the majority of values relatively centered within the error range and a slight skew towards the lower values compared to the global model. The single models showed an even wider spread of both MAE and MSE values, but with the majority of values further skewed towards the lower end.

Figure 17 shows the forecasting MAE scores of the time series in two individual clusters (cluster 11 and cluster 2) generated by the k-means method. The global model exhibited consistent tightness for both clusters, representing the overall performance across all clusters. The cluster-based model, while not as consistent as the global model, demonstrated varying performance depending on the time series within each cluster. For example, the cluster-based model performed better than the global model in cluster 11. However, when considering the median score across all clusters, the cluster-based model often underperformed compared to the global model. The performance of the single models varied significantly from cluster to cluster.



(a) K-means cluster 11 forecasting scores *(b)* K-means cluster 2 forecasting scores

Figure 17. MAE scores from two individual clusters, 11 and 2.

The median scores per cluster are calculated and displayed in Appendix C. The results indicate that the model performance in cluster 8 is the worst among all the cluster-based models. The global model consistently achieved a median MAE score of around 0.44 ± 0.02, indicating a stable performance. On the other hand, the cluster-based models showed a range of scores, with the best score at 0.038 and the worst score at 6.9. The behavior of the single model was similar, with a best score of 0.18 and a worst score of 1.3. Similar patterns were observed in terms of MSE, where the cluster-based models exhibited higher scores compared to the global and single models.

6.2 Self-organizing map

The forecasting metrics MSE and MAE were calculated based on the clusters and the test set created using the self-organizing map (SOM) method. Table 3 presents the median, mean, standard deviation, maximum, and minimum values for both metrics. The median and mean MAE values show an increase from the global model to the cluster-based and single models. While the maximum values remain similar, there is a significant difference in the minimum values. The cluster-based model's minimum value is approximately one-third of the global model, indicating the potential for better performance, but overall, it is worse than the global model.

To visualize the overall performance of the global, cluster-based, and single models, a box plot is presented in Figure 18. As mentioned in the previous section, the leftmost box in each of the images represents the scores from the global model, the center box represents the scores from the cluster-based

	global mae	cluster mae	single mae
median	0.367254	0.456406	0.621381
mean	0.642120	0.880984	0.906903
std	1.448199	1.693108	1.435699
max	16.321427	16.373581	15.350472
min	0.063248	0.023641	0.030397

(a) *MAE scores from the global, clustered and single models*

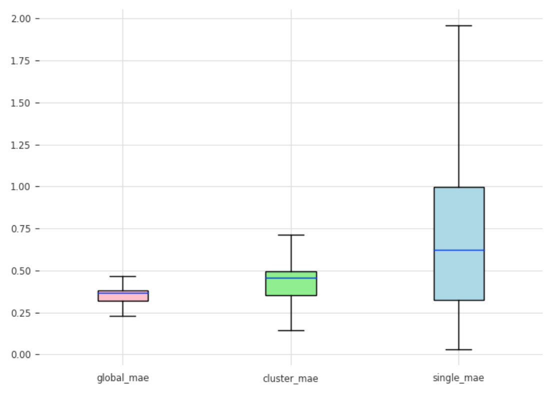
	global mse	cluster mse	single mse
median	0.138066	0.214312	0.495751
mean	3.851880	5.098782	4.461961
std	28.196032	29.586900	26.669330
max	392.739029	400.110181	350.058244
min	0.004416	0.000626	0.001318

(b) *MSE scores from the global, cluster-based and single models*

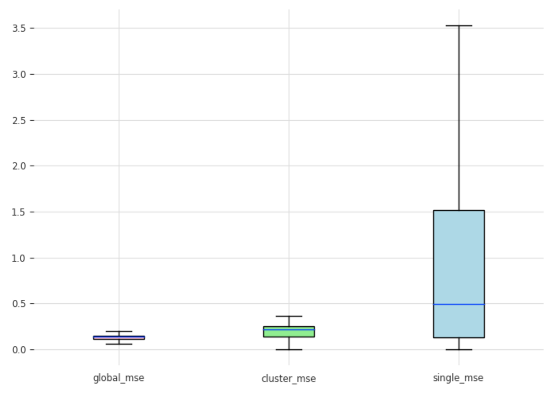
Table 3

MAE and MSE scores from the SOM method across all test.

models, and the rightmost box represents the scores from the single models. The result here is similar to the k-means result, where the performance of the global model is more consistent than the others. The cluster-based models exhibit a wider spread in performance, and the single models even more.



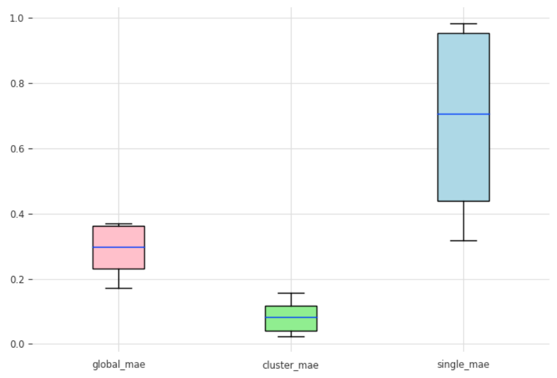
(a) SOM MAE



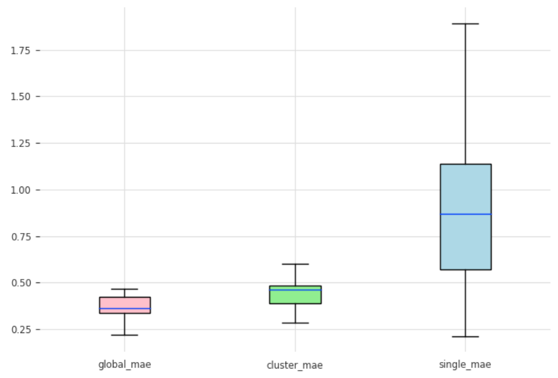
(b) SOM MSE

Figure 18. MAE and MSE scores over all test time series based on SOM.

Figure 19 provides a more detailed view of the MAE scores from two different clusters generated by the SOM method (cluster 7 and cluster 9). In both clusters, the performance of the global model remains consistent without significant variation in the top and bottom values.



(a) SOMs cluster 7 forecasting scores



(b) SOM cluster 9 forecasting scores

Figure 19. MAE and MSE scores from two individual clusters based on SOM, 7 and 9.

However, the cluster-based models demonstrate more variability. For example, in cluster 7, the MAE score ranges from 0 to 0.2, while in cluster 9, it ranges from 0.25 to 0.60.

The median MAE and MSE scores from the global, cluster-based, and single models are also analyzed on a cluster-wise basis, as shown in Appendix D. The median scores for the global model per cluster do not vary significantly, whereas both the cluster-based models and single models exhibit more variability. Similar to the results presented in Appendix C, a particular cluster stands out with much higher/worse scores in the cluster-based model. In this case, it is cluster 3, where the MAE is 5 and the MSE is 25.

7 Brief discussion and conclusions

First of all, the two clustering approaches showed very similar results overall in terms of the predictive performance of their respective follow-up LSTMs. Also, the clustering models were trained over similar ranges of different numbers of clusters and found numbers as close to each other as possible (12 and 13, respectively; each number not being available in the range of the other method). Finally, the data was split into training and test sets in such a way that the two approaches were tested on different test sets which, upon brief inspection, roughly had a 20 percent overlap. A quick look at the time series in the overlap (i.e., time series that were present in both test sets) it seemed that both k-means and SOM had distributed them similarly over clusters. While this would not necessarily have been the case for the entire test sets, had they been the same, all these things in conjunction seem to point towards the two methods finding roughly the same clusters. Because of this high similarity in results between the two clustering approaches, the research questions are not separately answered for the two clustering methods, as was originally intended.

With that said, circling back to the research questions stated in Section 1.1 these were to answer the following in terms of prediction accuracy, for both k-means and self-organizing map (SOM):

After clustering time series based on the similarity of extracted features, how does a single LSTM trained on all time series in a cluster compare to:

1. *Separate LSTMs trained on the individual time series in the cluster?*
2. *A single LSTM trained on the whole dataset?*

Arguably, these questions are not entirely unexpectedly answered trivially with *it depends*, but there are some interesting takeaways to be had.

Based on the presented results it seems like the global model is able to show consistent performance across all predictions to a considerably larger extent than the cluster-based and single models. However, while the variability in the global model's performance is quite low, its potential to predict well also seems to be lower than the cluster-based models, which in turn appear to have similar predictive potential compared to the single models judging by the results. Since both the upper and lower extremes in the predictive performance seem to get pushed outward the fewer time series a model has to train on, it is reasonable to think that some time series were easier for the LSTMs to train on and predict, and some were harder. To an extent, this is in line with the expectation stated in the introduction that the single models would show the best predictive performance as the results indicate that this might be true for the potential performance albeit not in general. It appears that the noise that is introduced by training on additional data can be either detrimental or beneficial to the prediction, depending on the difficulty of the particular target. This leads to questions regarding which type of forecasting model (i.e., global, cluster, or single) is optimal for which time series, in regard to predictive performance.

The cluster-based models show potential for improved performance compared to the global model in certain clusters, but there is also a higher degree of variability and an overall higher median prediction error. Additionally, the cluster-based models display considerably less variability than the single models while showing similar potential predictive performance. Finally, to reiterate, it is notable that the different types of forecasting models (global, cluster, and single), perform very differently for different clusters. Further investigation is required to understand the factors contributing to this difference in performance for each cluster.

The categorization of the data revealed that intermittent and lumpy data were the most prevalent categories. These categories pose significant challenges for prediction, making it interesting to perhaps employ a two-step clustering approach. The first step would involve clustering the data by these categories, which would be followed by clustering based on other features. This approach could prove beneficial by allowing the clustering algorithm to focus on more important features. Not unlikely, data with more non-zero values may enhance the model's performance. Thus, this two-step clustering methodology could shed light on differences in performance when forecasting different time series categories. It is plausible that the method in this thesis may be suitable for smooth data, but not for intermittent data.

Another noteworthy aspect of data preprocessing is the aggregation of daily sales into monthly sales and combining products with customers. These actions result in fewer data points and an increase in zero values, respectively. Although this was done to meet the stakeholder's requirements, it may have been advantageous to forecast product sales at a lower aggregation level, such as weekly. Also, while the DeepAR with LSTM approach was selected to handle numerous time series, it might not have been the most suitable choice. LSTM models are typically effective for longer time series, whereas in this project the individual time series were relatively short (around 50 data points) but numerous.

Although the primary goal of this paper was not to develop the best forecasting models or achieve optimal performance, it is important to note that the forecasting results obtained were far from exceptional. Consequently, implementing these results into a decision-support system at this stage would likely not be viable. While time constraints limited the possibility of conducting hyperparameter optimization on all models, this step could have improved the forecasting results significantly. If this method were to be used in production, hyperparameters would surely be tuned for each prediction model in order to get the best possible predictive performance, and so this would need to be done in the continuation of this work. This is important because it would not only prove a more accurate picture of the predictive capabilities but also of expected training time which could then be used in further evaluation of the approach. Nonetheless, the results presented in this work do provide a comparison of the performance among the three different forecasting models (global, cluster, and single).

Regarding the data split, the 90/10 split could have been performed with a restriction to ensure consistency in the test sets (1%) for the different models. This would have increased comparability between the models and improved the understanding of how the data is divided. It is important to note that the consistent behavior observed in the results of SOM and k-means reinforces the non-random nature of the findings generated in this study.

7.1 Future work

7.1.1 Feature extraction

Improve feature extraction to find the most relevant features for the particular dataset. Currently, the features that are used are simply all features returned from using the *tsfeatures* library, along with a number of other, mostly arbitrarily chosen, features. While some thought was given to considering the relevance of the features that were used (as explained in Section 4.2), this was very limited. Fulcher (2017) discusses the difficulties with defining the optimal set of features. As optimal depends on the task and the dataset, no general solution to finding this exists. Thus, as more might not necessarily be better regarding the set of features and not all features are equally relevant, it is likely beneficial to better investigate what features to use. This could, and possibly should, include working with domain experts for defining the most relevant features, which is how it is often done.

7.1.2 Hyperparameter tuning

While some hyperparameter tuning was done for the LSTMs (see Section 4.4) the obtained values are not necessarily optimal for all the models that were trained. If the desire was the achieve the best possible prediction results, it might be relevant to tune the hyperparameters for each individual model that is trained. This, of course, takes considerably more time than tuning on a subset of the data and using the mean values, but might both produce better prediction results and present a more accurate image of the time training would take for the different types of models in production. Thus, doing more extensive hyperparameter tuning might produce results that are both more interesting and more relevant to the company that the study was done in cooperation with.

7.1.3 Classifying time series for prediction

For our dataset, our results seem to indicate that while the global LSTM appears more robust (i.e., has the tightest prediction error interval), the cluster-based and single LSTMs might perform better for some time series. This might warrant further investigation to see if common characteristics of time series can be found that dictate which of the approaches (i.e., global, cluster, or single LSTM) might be more appropriate. This is interesting since if such defining characteristics can be found, it might be possible to split the data set into subsets for which the different types of LSTMs can be trained in order to optimize predictive performance.

7.1.4 Time series segmentation

Time series segmentation is a process where a time series sequence is split into segments. This can be done rolling, by set time intervals, or by dynamic approaches. The goal of this technique could be to find patterns for each segment and group these together by similarity, or simply to deal with the problem of time series having varying lengths. To classify future segments, a classification model can be used to fit the data into the most relevant cluster (Castán-Lascorz et al., 2022). This approach could be employed instead of feature extraction from the time series. Extracting features may result in information loss, whereas segmenting the time series into segments can increase the dataset size and retain the data representation.

References

Aghabozorgi, S., Seyed Shirkhorshidi, A., & Ying Wah, T. (2015). Time-series clustering – a decade review. *Information Systems*, *53*, 16–38. https://doi.org/https://doi.org/10.1016/j.is.2015.04.007

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J. M., & Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition*, *46*(1), 243–256. https://doi.org/https://doi.org/10.1016/j.patcog.2012.07.021

Bandara, K., Bergmeir, C., & Smyl, S. (2020). Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Systems with Applications*, *140*, 112896. https://doi.org/https://doi.org/10.1016/j.eswa.2019.112896

Candelieri, A. (2017). Clustering and support vector regression for water demand forecasting and anomaly detection. *Water*, *9*, 224. https://doi.org/10.3390/w9030224

Castán-Lascorz, M. A., Jiménez-Herrera, P., Troncoso, A., & Asencio-Cortés, G. (2022). A new hybrid method for predicting univariate and multivariate time series based on pattern forecasting [Clustering time series segments by similarity (classification model). train a forecasting model on each segment]. *Information Sciences*, *586*, 611–627. https://doi.org/10.1016/J.INS.2021.12.001

Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-1*(2), 224–227. https://doi.org/10.1109/TPAMI.1979.4766909

Ensafi, Y., Amin, S. H., Zhang, G., & Shah, B. (2022). Time-series forecasting of seasonal items sales using machine learning – a comparative analysis. *International Journal of Information Management Data Insights*, *2*(1), 100058. https://doi.org/https://doi.org/10.1016/j.jjimei.2022.100058

Fulcher, B. D. (2017). Feature-based time-series analysis.

Herzen, J., LÃ¤ssig, F., Piazzetta, S. G., Neuer, T., Tafti, L., Raille, G., Pottelbergh, T. V., Pasieka, M., Skrodzki, A., Huguenin, N., Dumonal, M., KoÅ›cisz, J., Bader, D., Gusset, F., Benheddi, M., Williamson, C., Kosinski, M., Petrik, M., & Grosch, G. (2022). Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research*, *23*(124), 1–6. http://jmlr.org/papers/v23/21-1177.html

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hua, Y., Zhao, Z., Li, R., Chen, X., Liu, Z., & Zhang, H. (2019). Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, *57*(6), 114–119. https://doi.org/10.1109/MCOM.2019.1800155

Hyndman, R., Kang, Y., Montero-Manso, P., Talagala, T., Wang, E., Yang, Y., & O'Hara-Wild, M. (2023). *Tsfeatures: Time series feature extraction* [R package version 1.1.0.9000]. https://pkg.robjhyndman.com/tsfeatures/

Ismail, S., Shabri, A., & Samsudin, R. (2011). A hybrid model of self-organizing maps (som) and least square support vector machine (lssvm) for time-series forecasting. *Expert Syst. Appl.*, *38*(8), 10574–10578. https://doi.org/10.1016/j.eswa.2011.02.107

Kiefer, D., Grimm, F., Bauer, M., & Dinther, C. V. (2021). Demand forecasting intermittent and lumpy time series: Comparing statistical, machine learning and deep learning methods. https://hdl.handle.net/10125/70784

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78*(9), 1464–1480. https://doi.org/10.1109/5.58325

Li, M. J., & Liu, W. (2018). Load pattern shape clustering analysis for manufacturing [fuzzy k-means clustering of time series]. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10699 LNCS*, 67–80. https://doi.org/10.1007/978-3-319-73830-7_7/TABLES/3

Liao, T. W. (2005). Clustering of time series data—a survey. *Pattern Recognition*, *38*(11), 1857–1874. https://doi.org/https://doi.org/10.1016/j.patcog.2005.01.025

Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2015). Learning to diagnose with lstm recurrent neural networks. https://doi.org/10.48550/ARXIV.1511.03677

Lolli, F., Gamberini, R., Regattieri, A., & Balugani, E. (2017). Single-hidden layer neural networks for forecasting intermittent demand. *International Journal of Production Economics*, *183*, 116–128. https://doi.org/10.1016/j.ijpe.2016.10.021

Looij, T. v. d., & Ariannezhad, M. (2021). Cluster-based forecasting for intermittent and non-intermittent time series [Cited by: 0]. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13114 LNAI*, 139–154. https://doi.org/10.1007/978-3-030-91445-5_9

Rachwał, A., Popławska, E., Gorgol, I., Cieplak, T., Pliszczuk, D., Skowron, Ł., & Rymarczyk, T. (2023). Determining the quality of a dataset in clustering terms. *Applied Sciences*, *13*(5). https://doi.org/10.3390/app13052942

Ramalingam, R., Abdul Rahman, A. A., Dhamodharavadhani, s., Meero, A., & G, Y. (2022). Demand forecasting model for time-series pharmaceutical data using shallow and deep neural network model. *Neural Computing and Applications*, *35*. https://doi.org/10.1007/s00521-022-07889-9

Rick, R., & Berton, L. (2022). Energy forecasting model based on cnn-lstm-ae for many time series with unequal lengths. *Engineering Applications of Artificial Intelligence*, *113*, 104998. https://doi.org/https://doi.org/10.1016/j.engappai.2022.104998

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65. https://doi.org/https://doi.org/10.1016/0377-0427(87)90125-7

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191. https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.07.001

Shabani, A., Eski, A., Panxhi, D., & Zavalani, O. (2018). Identification of daily patterns in building energy consumption, 19 (6 pp.)–19 (6 pp.) https://doi.org/10.1049/cp.2018.1851

Syntetos, M., Boylan, J., & Croston, J. (2005). On the categorization of demand patterns. *Journal of the Operational Research Society*, *56*. https://doi.org/10.1057/palgrave.jors.2601841

Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., Payne, M., Yurchak, R., Rußwurm, M., Kolar, K., & Woods, E. (2020). Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, *21*(118), 1–6. http://jmlr.org/papers/v21/20-091.html

Taylor, S., & Letham, B. (2017). Forecasting at scale. https://doi.org/10.7287/peerj.preprints.3190v2

Vettigli, G. (2018). Minisom: Minimalistic and numpy-based implementation of the self organizing map. https://github.com/JustGlowing/minisom/

Vo, V., Luo, J., & Vo, B. (2016). Time series trend analysis based on k-means and support vector machine. *35*, 111–127.

Wang, X., Smith, K., & Hyndman, R. (2006). Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery*, *13*(3), 335–364. https://doi.org/10.1007/s10618-005-0039-x

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*. https://doi.org/10.48550/ARXIV.2106.11342

Appendices

Appendix A SOM evaluation scores

Output map shape	Number of clusters	Davies-Bouldin	Silhouette
(2, 2)	4	2.030977	0.131160
(2, 3)	6	1.983800	0.103089
(3, 2)	6	1.983800	0.103089
(2, 4)	8	1.662129	**0.142776**
(4, 2)	8	1.662129	0.142776
(3, 3)	9	1.588830	**0.143285**
(2, 5)	10	**1.553892**	**0.136750**
(5, 2)	10	1.553892	0.136750
(2, 6)	12	**1.546053**	**0.140869**
(6, 2)	12	1.546053	0.140869
(3, 4)	12	**1.540821**	**0.155282**
(4, 3)	12	1.540821	0.155282
(2, 7)	14	**1.568724**	0.127831
(7, 2)	14	1.568724	0.127831
(3, 5)	15	1.641406	0.118286
(5, 3)	15	1.641406	0.118286
(2, 8)	16	**1.585347**	0.123891
(8, 2)	16	1.585347	0.123891
(4, 4)	16	1.601689	0.113348
(3, 6)	18	1.703179	0.116944
(6, 3)	18	1.703179	0.116944
(4, 5)	20	1.678917	0.124690
(5, 4)	20	1.678917	0.124690
(3, 7)	21	1.730384	0.108776
(7, 3)	21	1.730384	0.108776
(3, 8)	24	1.642105	0.109874
(8, 3)	24	1.642105	0.109874
(4, 6)	24	1.667656	0.118273
(6, 4)	24	1.667656	0.118273
(5, 5)	25	1.658965	0.118506
(4, 7)	28	1.663545	0.113460
(7, 4)	28	1.663545	0.113460
(5, 6)	30	1.703007	0.108470
(6, 5)	30	1.703007	0.108470
(4, 8)	32	1.686031	0.114337
(8, 4)	32	1.686031	0.114337
(5, 7)	35	1.695571	0.105285
(7, 5)	35	1.695571	0.105285
(6, 6)	36	1.762313	0.110024
(5, 8)	40	1.674825	0.103850
(8, 5)	40	1.674825	0.103850
(6, 7)	42	1.744297	0.108302
(7, 6)	42	1.744297	0.108302
(6, 8)	48	1.711202	0.106630
(8, 6)	48	1.744853	0.103570
(7, 7)	49	1.723186	0.099715
(7, 8)	56	1.705162	0.100037
(8, 7)	56	1.743246	0.102149
(8, 8)	64	1.733807	0.104745

Table 4

Silhouette and Davies-Bouldin scores from SOM clustering on features using Euclidean distance. Bold values are the top 5 scores for each metric.

Appendix B k-means evaluation scores

Number of clusters	**Davies-Bouldin**	**Silhouette**
5	1.697786	**0.173996**
7	1.519505	**0.197160**
9	1.611879	**0.162746**
11	1.598028	**0.157844**
13	**1.462478**	**0.164683**
15	1.535553	0.143173
17	**1.482508**	0.150329
19	1.512893	0.146189
21	**1.444065**	0.154285
23	**1.484121**	0.151911
25	1.508860	0.153807
27	1.515343	0.143726
29	**1.450009**	0.146340
31	1.507679	0.149512
33	1.505367	0.147117
35	1.534574	0.133262
37	1.502582	0.143375
39	1.539952	0.133523
41	1.525707	0.124304
43	1.519481	0.145189
45	1.533403	0.137594
47	1.528539	0.126224
49	1.546771	0.132119
51	1.620455	0.134657
53	1.510238	0.135273
55	1.581581	0.123005
57	1.591676	0.125206
59	1.544193	0.128516
61	1.538803	0.126636
63	1.582182	0.122429

Table 5

Raw silhouette and Davies-Bouldin scores for each cluster size from k-means clustering.

Appendix C k-means MAE, MSE per cluster

cluster_ID	global mae	cluster mae	single mae
0	0.442446	0.632648	0.616794
1	0.415435	0.535505	0.824209
2	0.440707	0.575719	0.738529
3	0.443215	0.262624	0.931210
4	0.446626	0.467222	0.626802
5	0.463062	0.313886	0.478580
6	0.425095	0.478566	0.945209
7	0.462163	0.140393	1.324182
8	0.443583	6.952641	0.297796
9	0.437433	0.201508	0.332319
10	0.454104	0.110964	0.293615
11	0.450796	0.038681	0.329105
12	0.434664	0.273947	0.189285

(a) *MAE scores from global, cluster-based and single models, per cluster*

cluster_ID	global mse	cluster mse	single mse
0	0.196470	0.405911	0.380678
1	0.172589	0.291202	0.890604
2	0.194776	0.332285	0.756644
3	0.207643	0.134843	0.969718
4	0.200076	0.220040	0.632203
5	0.214900	0.098527	0.279673
6	0.186970	0.240182	1.536370
7	0.214043	0.021039	2.523837
8	0.198106	66.863459	0.089797
9	0.192425	0.041079	0.111028
10	0.206578	0.012364	0.087111
11	0.203666	0.001501	0.114706
12	0.190176	0.076921	0.036847

(b) *MSE scores from global, cluster-based and single models, per cluster*

Table 6

MAE and MSE scores from forecasting based on the k-means cluster.

Appendix D SOM MAE, MSE per cluster

cluster_ID	global mae	cluster mae	single mae
0	0.354531	0.186698	0.194124
1	0.360885	0.351901	0.552539
2	0.377680	0.405257	0.448186
3	0.376849	5.011297	0.285853
4	0.355799	0.470069	0.841826
5	0.367581	0.482983	0.720129
6	0.366965	0.231924	0.341357
7	0.296748	0.083207	0.705479
8	0.362043	0.512725	0.799225
9	0.364064	0.464382	0.868798
10	0.371295	0.453072	0.867552
11	NaN	NaN	NaN

(a) *MAE scores from the global, cluster-based and single models per cluster*

cluster_ID	global mse	cluster mse	single mse
0	0.129899	0.035118	0.039255
1	0.131174	0.124034	0.331739
2	0.144792	0.167387	0.216714
3	0.144221	25.875121	0.082296
4	0.131475	0.224342	0.880203
5	0.137940	0.234081	0.735446
6	0.137220	0.061245	0.139890
7	0.095407	0.008196	0.652399
8	0.133585	0.265677	0.887761
9	0.135642	0.216554	1.111644
10	0.140191	0.221716	1.01040
11	NaN	NaN	NaN

(b) *MSE scores from the global, cluster-based and single models per cluster*

Table 7

MAE and MSE scores from the SOM method per cluster.